Encrypted Messaging App

Oran Keating

Dr Challoners Grammar School

AQA A-Level Computer Science NEA

Contents

Analysis

Meeting the Complexity Requirements:

Technical Skills Group A	**How I am meeting it**
Complex data model in database: ● Cross-table parameterised SQL ● User/CASE-generated DDL script	Cross-table parameterised SQL: Retrieving needed data User/CASE-generated DDL script: Creating databases Other: Normalised relational database
Advanced matrix operations	Matrix multiplication as part of the AES algorithm
Complex mathematical model (GF2^8 and AES)	Multiplication in the Galois field 2^8 as part of the AES algorithm
Complex user-defined algorithms (eg optimisation, minimisation, scheduling, pattern matching) or equivalent difficulty	Advanced Encryption Standard Algorithm
Dynamic generation of objects based on complex user-defined use of OOP model	Server creates a new instance of an object to handle each new user connected. Each page in the app is a new object that is instantiated when the user moves to that page
Complex user-defined use of object-oriented programming (OOP) model, eg classes, inheritance, composition, polymorphism, interfaces	Client and Server inherit from a base class that holds all of their dual functionalities such as sending and receiving data. Base class inherits from AES encrypt and decrypt classes Composition in GUI to allow client and gui to communicate Polymorphism when using the client and server classes as they have slightly different functions
Complex client-server model: ● Server-side scripting using request and response objects and server-side extensions for a complex client-server model	Entire Client Server model
parsing JSON/XML to service a complex client-server model	Data sent between client and server is serialised using JSON and other methods
Defensive programming	Preventing SQL injection

	Try and except statements to catch any errors that may occur.
Scheduling	Used when client or server is sending and receiving data

Background and Problem Definition:

Encrypted messaging applications are commonplace today with whatsapp being the most famous one with over 2 billion users globally. These apps give people security in the knowledge that no one can read their messages except them and those that they are sent to.

Ben is a 6th form student at a secondary school. Encrypted messaging apps are used commonly in schools. Ben and his friends often use whatsapp to communicate when they are not together however he does not always have his phone on him and is frustrated by the lack of easy to use encrypted messaging apps on the computer. He has asked me to make an encrypted messaging app available for computers that allows him to send messages to his friends across the internet. He wants the app to have end-to-end encryption as he does not want anyone else looking at his messages.

Defining the users:

The users will be the general population with the main group benefiting from it being Ben and his friends. They will be able to use it to send messages to people who they are friends with on the app. If the app is rolled out to the general population it will have to meet accessibility standards.

How it will be researched:

The following is a list of methods I will use to research the problem:

- Interviews
- Analysis of existing solutions
- Design mockups to ensure I have understood the requirements
- Research into hashing at a base level
- Comparison of different hashing algorithms to find the most suitable one
- Research into encryption at a base level
- Comparison of different encryption algorithms to find the most suitable one
- Looking at the GDPR to ensure my program complies with it
- Modelling parts of the problem that will inform the design stage

Interview 1:

To allow me to understand my end users requirements I conducted an interview with them. I asked questions about the key features they wanted in the system.

Prerequisite: Ben S has commissioned me to create an encrypted messaging application for computers.

Q: Do you want to send different file types such as MP3 or JPEGs?
A: Yes, I want to be able to send memes and videos and sound effects with this program although it does not need to be the highest quality as long as you can make out what it is.

Analysis: He wants to be able to send different file types such as image files and sound/video files. Videos and MP3 files can be compressed as quality does not matter. If the quality does not matter so much then the ability to send WAV files is not needed.

Q: Do you want there to be message logs?
A: Yeah I want to be able to see all the messages I have sent like in whatsapp.

Analysis: There needs to be an ability to login/create an account if they want to be able to see messages they have sent. Along with account functionality is the forgotten password function. It will not act as a chatroom where people connect and chat but more as a messaging software similar to whatsapp.

Q: Who will you be using the site to mainly message
A: I want to be able to message my friends one on one from anywhere.

Analysis: 'One on one from, anywhere' means it will need to be sent over the internet not an LAN.

Q: You mentioned one-on-one, does this mean you don't want group chat functionality?
A: Yeah, I don't need the ability to make group chats.

Analysis: One-on-one messaging only. No need to implement group chats.

Q: What is the rough time frame you want it to be completed by?
A: There is no rush but I want it to be completed in just under a year.

Analysis: Timeframe - under one year

Q: How secure do you want your messages to be?
A: I want them to be secure enough so that anyone trying to intercept the message will not be able to read them even if they succeed in intercepting them.

Analysis: This means I will need to implement end to end encryption so that even if the message is intercepted it won't be able to be decoded.

Q: Do you want to be able to login with other accounts such as google?
A: I want no one else to have any sort of access to my account, not even google.

Analysis: Does not want the functionality to login with google or other services.

Q: How do you want to make contact with another user?
A: I want to be able to type their username in and then be able to message them without them having to accept any sort of friend request.

Analysis: Anyone can message anyone without the need to accept or send a friend request. Contact is established by typing in the other person's UserID.

Q: Do you want to be able to change your display name?
A: Yes I want to be able to change it to whatever I want.

Analysis: When creating an account the details needed will be username, which will act as the UserID, password which will be hashed and a display name which will be changeable. In addition an email address will be required for verification.

Q: Do you want to be able to delete your account?
A: Yes and when the account is deleted I want no one to know I ever had one.

Analysis: When the account is deleted so is all the data held with it meaning that on chat logs with saved messages the saved message will be replaced with ‘Message Deleted’.

Q: Do you want to be able to delete messages
A: Yeah, sometimes I send messages I regret so I want to be able to delete them.

Analysis: The user needs to be able to delete messages.

Q: Do you want them to be able to see you have deleted a message?
A: Yes, like on whatsapp.

Analysis: When you delete a message on whatsapp it replaces the text with ‘This message was deleted’. Implement that functionality into the application.

Q: Thanks for taking the time to be here and answer my questions. Before we finish, do you have any questions for me?

A: No. Thank you.

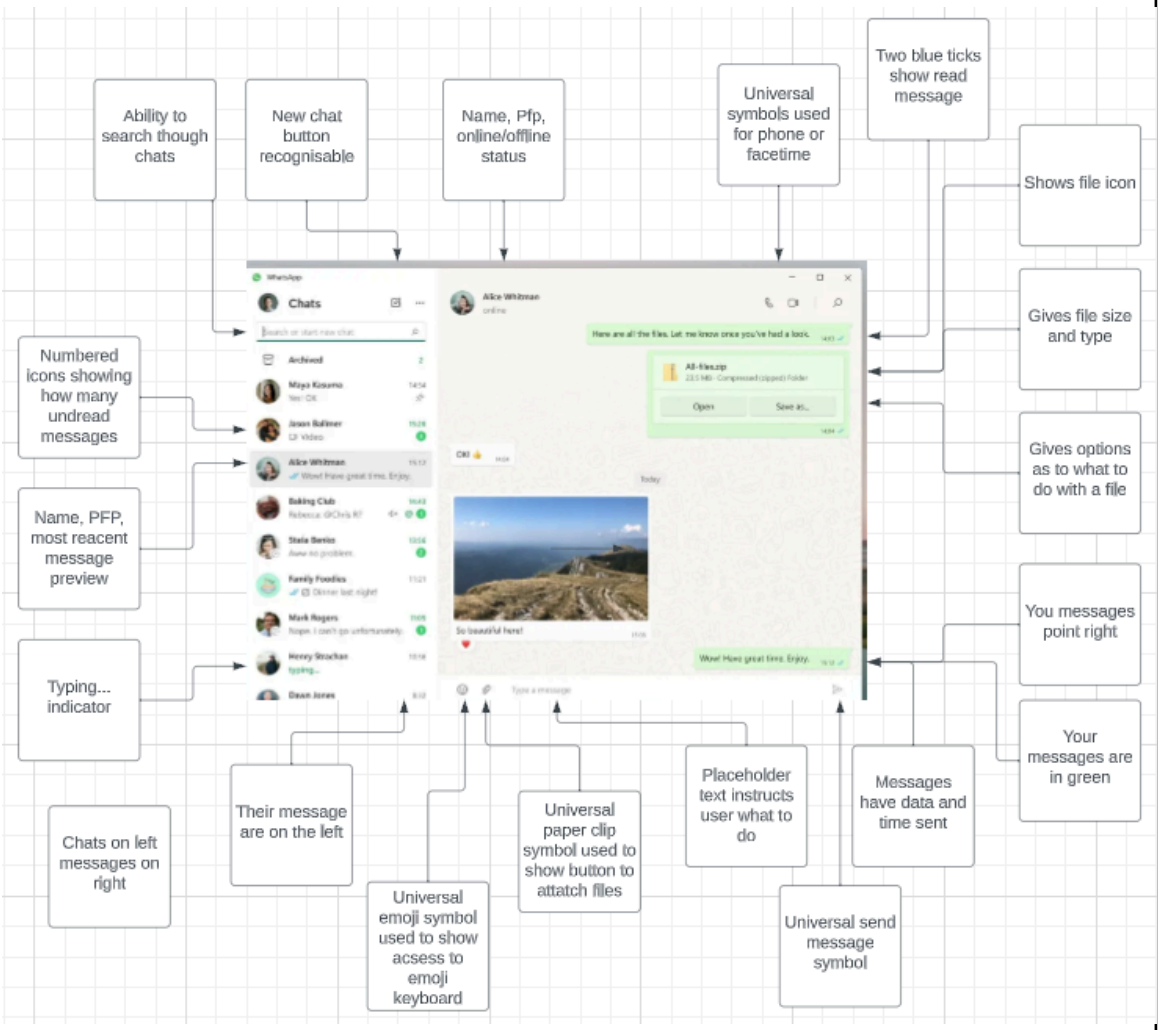
Conclusion: Many of the objectives I state at the end were derived from the answers and analysis of the answers in this interview.

Existing Solutions:

Whatsapp:

UI Analysis:

Analysing an app's UI is important as people will be used to specific conventions and so deviating from them will make my app harder to use. Analysing the app's UI allows me to identify these conventions and use them in my own design. Deviating from these conventions will confuse the user and make using the UI difficult.



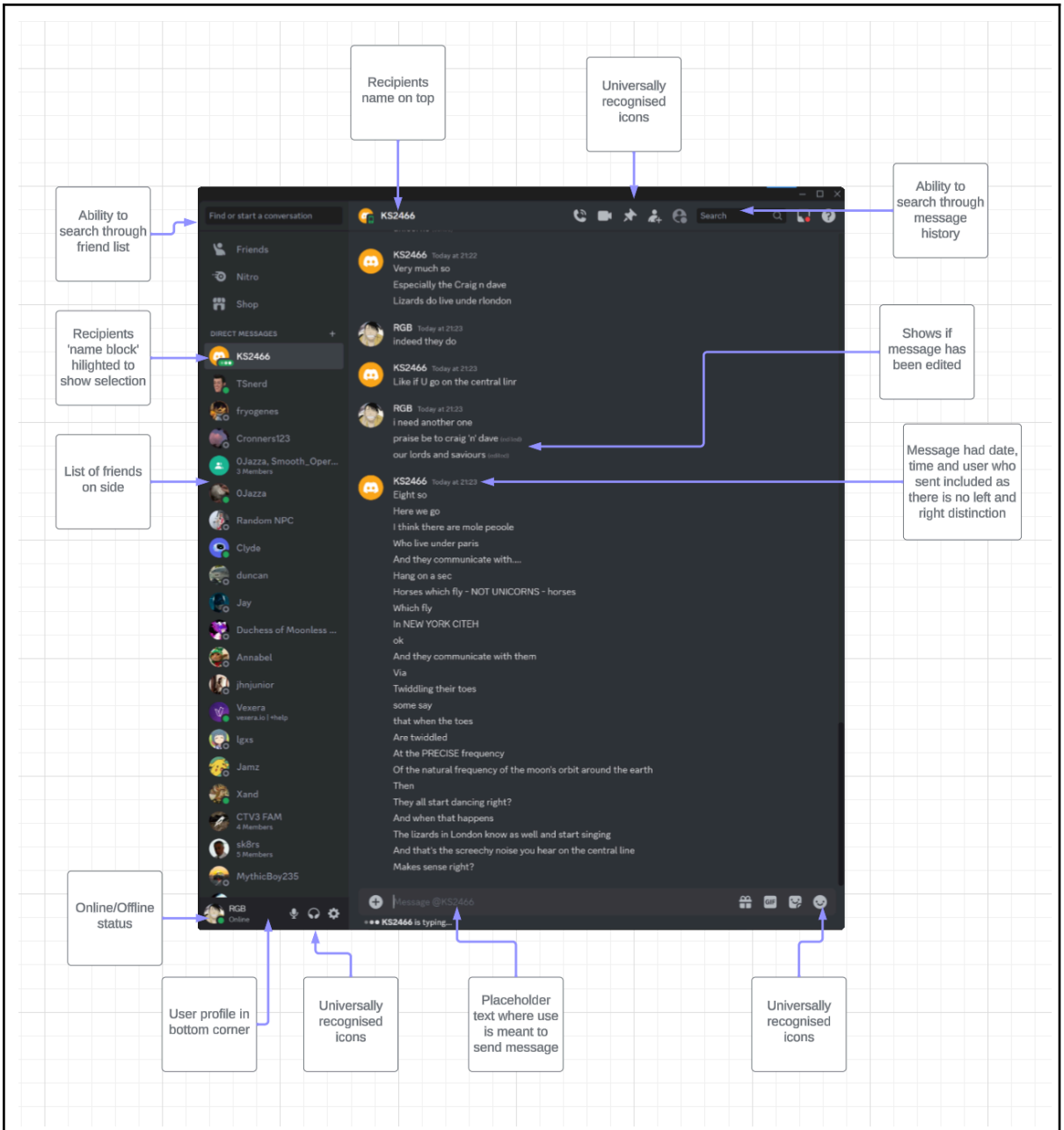
Analysis of Whatsapps UI

Key takeaways:

- Use universal symbols such as paper clip for file attachment as this makes use of app intuitive
- Have list of chats on the left and messages on the right
- Messages should have time sent and date sent included
- Anywhere the user is meant to type should have placeholder text
- There should be a clear distinction between your messages and the recipients messages

Discord

UI Analysis:

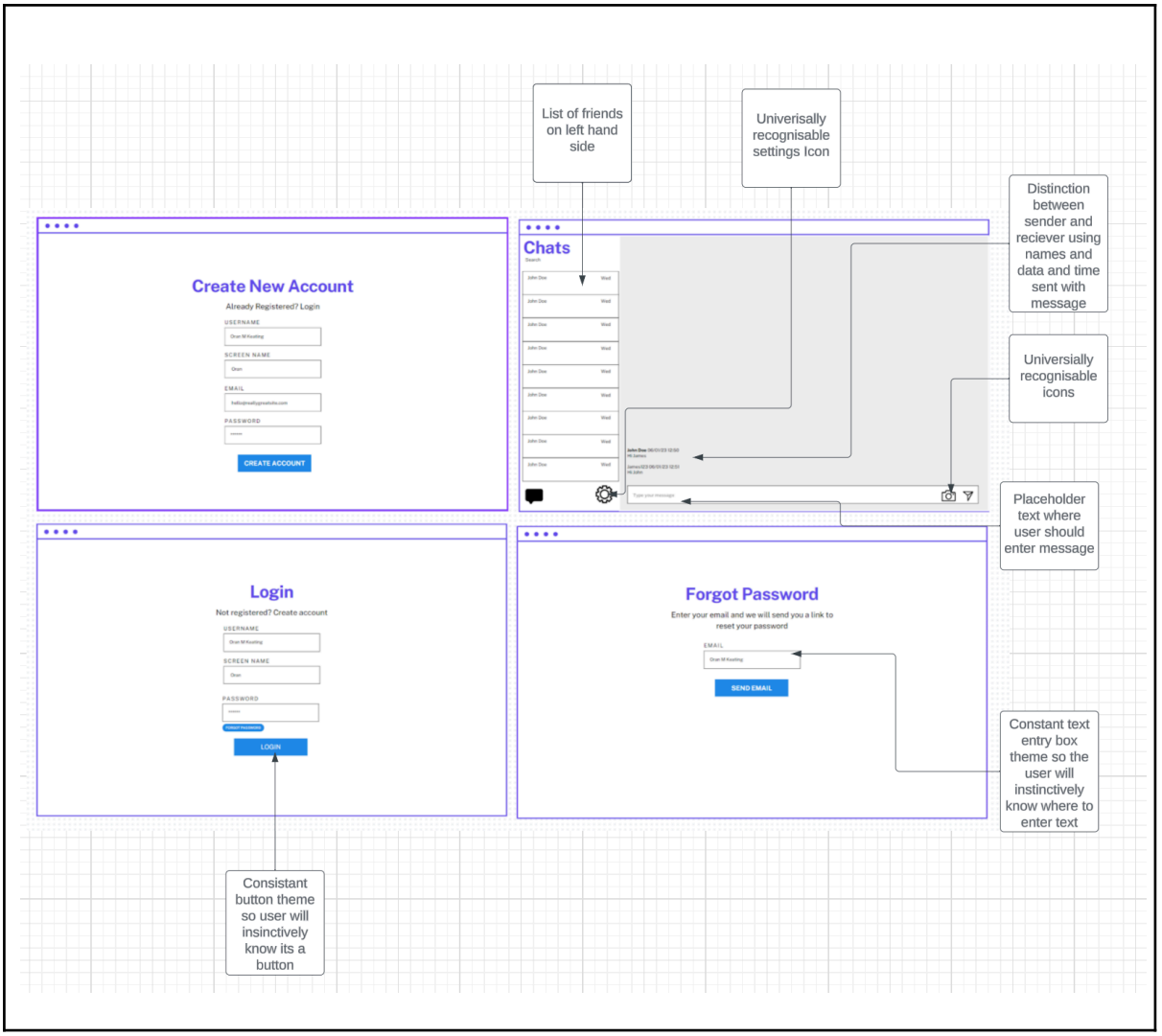


Analysis of Discords UI

Key takeaways:

- Universal symbols used
- If left right sender/receiver distinction unavailable make it clear who is sending the message.
- Placeholder text where user can type
- List of friends on the left hand side
- Recipients name on top of chat window
- User profile in bottom left corner

Annotated UI Mockups V1



Personal critique of UI

- The camera icon is misleading as the user won't be taking a photo but instead will be attaching a file.
- Searching through chats text box is not a text entry box
- It is not clear what the 'add friends icon' is meant to do
- The recipient's name is not on the top of the chat window

Interview 2

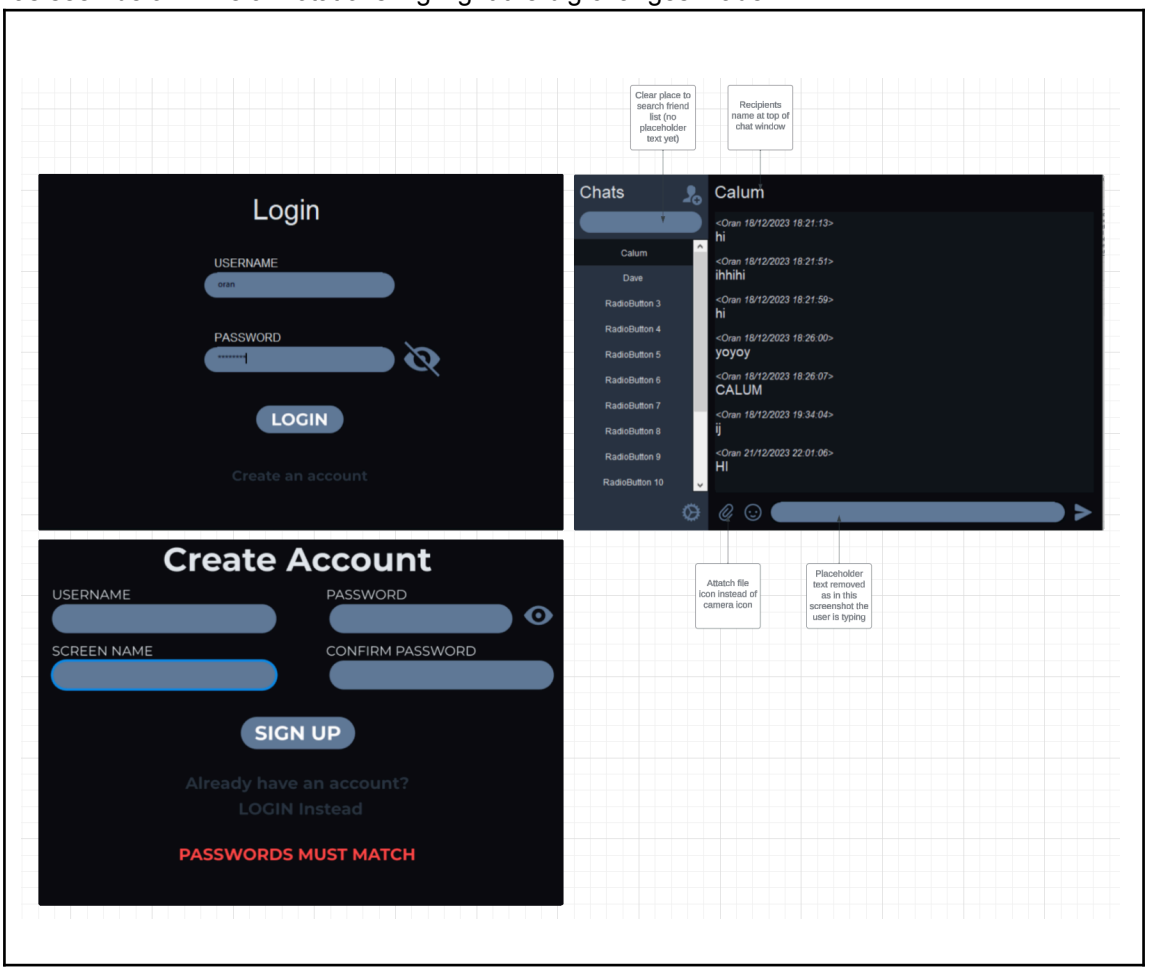
I presented Ben with the annotated mockups and explained the flow of the program and its general functionalities.

Q: Is there any amendments you want to make at this final stage before I start the work.
A: Yeah. I want the forgot password thing to be removed and also the requirement of any email address in the creating account stage. I would like there to be a requirement where you enter the password twice when creating the account in case you mistyped it. Also I would like to be able to send emojis.

Analysis: Email verification and forgot password need to be removed. When creating an account, have the user put in their password twice and ensure they are the same. Add the ability to send emojis.

Annotated UI Mockups V2

After taking into consideration my own personal critique of the old UI and Bens critique I made a new UI as seen below. The annotations highlight the big changes made.



Hashing

A hashing function is only one-way therefore there is no way to ‘decrypt’ a hashed string. Encryption on the other hand is two-way meaning the original plain-text can be retrieved. (*Password Storage - OWASP Cheat Sheet Series*, no date)

The two primary uses of hashing are hash tables and password storage. It is used for password storage as it is impossible to ‘decrypt’ a hash meaning that even if a hacker had access to the stored passwords they would not be able to use them to log in as the victim.

For a hashing function the same input will always result in the same output which will be unique to all other outputs. This makes it perfect for password storage as to check a user's password we can simply hash their entered password and compare it against the hash stored in the database.

While hashes themselves are impossible to ‘decrypt’ , hackers have other methods of finding out users' passwords. Many hashes have special methods in place to protect against these other methods.

Two of these special methods are what's known as salting and peppering the password. (*Password Storage - OWASP Cheat Sheet Series*, no date)

Salt and Pepper

Although hashing is a strong defence against hackers they have ways around it such as brute force attacks, dictionary attacks or making use of rainbow tables. (*Team, 2024*) Salt and Peppers aim is to slow the attacker down even further if they attempt these attacks.

Salting a password refers to appending or prepending a randomly generated string to each password before it is hashed then hashing it with this added ‘salt’. As the salt is unique for each user, if a hacker is trying to brute force the password or use a rainbow table they would have to crack each hash one by one using the unique salt in each case instead of calculating a hash once and comparing it against every other hash in the table. It also means that if two users have the same password the resultant hash will be unique because of the added salt. (*Password Storage - OWASP Cheat Sheet Series*, no date)

A salt needs to be cryptographically strong and credential-specific. That means for each new password created a new salt should be created with it. The salt should be as many bytes as the output of the hash. (*What is a salt and how it boosts security?*, no date)

Peppering is where you encrypt the hashes with a symmetrical encryption key (the key is used to both encrypt and decrypt the text) before storing the hashed password. This key acts as the ‘pepper’. Unlike salt, the same pepper is used for every password. In addition the pepper must not be stored in the database. Pepper should be stored in HSMs (Hardware Security Modules). (*Password Storage - OWASP Cheat Sheet Series*, no date)

Analysis of different hashing Algorithms

In order to successfully weigh up the pros and cons of different hashing algorithms I first need to understand some key terms and concepts.

Collisions: Although before I said for a hashing function the output will always be unique to all other outputs of that same function, sometimes this isn't the case and two different inputs can result in the same output. This is known as a collision and can have disastrous consequences and is the primary reason why SHA-1 is no longer used. (*Isaac Computer Science*, no date)

Side channel attacks: This attack exploits the fact that as processes are executed by the computer the hard drive and other hardware used by the computer emits electrical signals that contain some sort of information. (*Greenberg, 2020*)

GPU attacks: This attack is where the attacker exploits vulnerabilities the graphics processing unit has to gain information they shouldn’t have. (*Infospoint, 2023*)

Why do we want Hashing Algorithms to be slow?

The safety of a hash depends on how fast the hashing function takes to calculate the hash. This is because if someone was attempting a brute force attack or dictionary attack if the hashing function was fast they would be able to complete the attack in mere seconds whereas if each hash takes a second to calculate a brute force or dictionary attack would take much longer. (*Password Storage - OWASP Cheat Sheet Series*, no date)

Hashing Algorithm	Overview	Positives	Negatives
SHA-256	Latest in the family of SHA hashing algorithms. Used for hash tables	Low probability of collisions. Many explanations of algorithms online.	Fast to calculate the hash. (If you are using hash tables for database lookups for example this is treated as a positive)
Argon2	It was the winner of the 2015 Password Hashing Competition. Variant Argon2id should be used as it prevents side channel attacks and GPU cracking attacks.	Slow - if a hacker is performing a brute force attack it will be very slow as the computation behind the hashing function takes a lot of memory to perform. Prevents side-channel attacks.	Extremely complex to code with little documentation or explanations of the maths behind it.
Bcrypt	Password hashing algorithm based on the BlowFish Cypher and the hashing function used by the UNIX password system called crypt. Designed in such a way that it is tuned to run slower on newly available hardware meaning that even if computing power increases the speed stays slow. It should have a minimum work factor of 10	Function requires salt. It can be tuned to be as slow or fast as needed. Has lots of documentation and explanations for the algorithm online.	72 character limit. Less resistant to GPU attacks than Argon2.

Encryption

Encryption is the process of converting a message from plain text into cipher text. (*Isaac Computer Science*, no date b) Its purpose is to ensure that intercepted data cannot be understood by the attacker. There are two fundamental encryption types of encryption algorithms: Symmetric and Asymmetric. ‘Symmetric encryption uses the same key for both encryption and decryption, while asymmetric encryption uses a pair of keys: a public key for encryption and a private key for decryption.’ (*Types of encryption: symmetric or asymmetric? RSA or AES?* | *Prey blog*, 2021)

There are advantages and disadvantages to both methods and choosing one depends on context around how you are using it.

Analysis of different encryption Algorithms

Type	Overview	Pros	Cons
Diffie-Hellman with Advanced Encryption Standard (AES)	Using the Diffie-Hellman procedure to share the shared secret for the AES algorithm. AES is a symmetric encryption algorithm which uses the symmetry of the XOR logic gate to encrypt and expands upon it. Used widely and by governments.	Diffie-Hellman's maths is not too complex and does not seem particularly hard to implement. AES is widely known and will have a lot of documentation which I can learn from to understand how it works. I am already using AES to encrypt the passwords after they have been hashed meaning I can re-use code.	For each new text sent a new key has to be generated and communicated. Vulnerable to Man in the Middle attack. (https://www.geeksforgeeks.org/man-in-the-middle-attack-in-diffie-hellman-key-exchange/)
RSA	Primarily relies on the difficulty of factoring large numbers into their prime factors. It is asymmetric	Cobines encryption, key exchange and signatures all into one. Standard for encrypting data over the internet. The maths of RSA has been explained widely so finding it in the first place is easy. It can be used to create digital signatures. It is relatively fast.	Complex. The decryption process takes a lot of processing power on the receiver's end. The larger the key size the better the security however calculating the key takes more time as the key size increases.
Signal encryption	Symmetric cypher used by whatsapp and signal. Makes use of diffie-hellman to exchange keys and uses the triple version.	Open source so if needed I can look at the source code to get an understanding. Each complex layer is broken down into clearly defined sub-sections.	Has lots of complex layers. At first glance it seems like the most complex.

What encryption algorithm will I use?

I will use AES as I already understand the maths behind Diffie-Hellman and because AES is so well documented I feel I will easily be able to find enough information about it to recreate it myself. In addition, using AES allows me to re-use the function for peppering passwords.

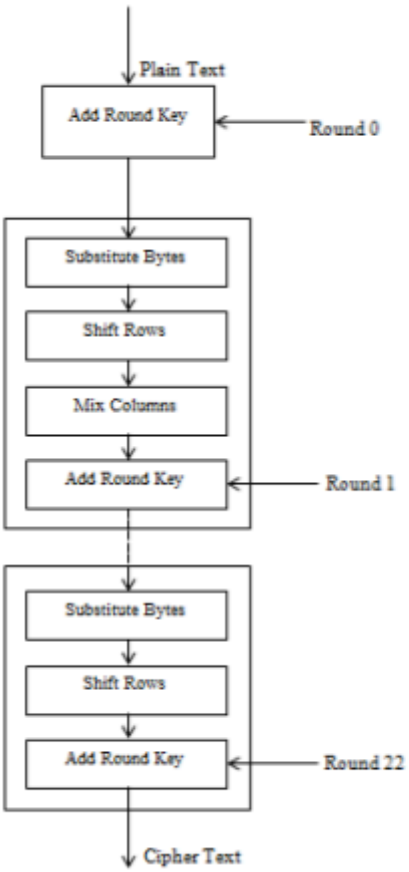
Further research into AES and Diffie-Hellman

AES is a symmetrical encryption algorithm. One very simple example of symmetrical encryption is an XOR gate. XOR gates are fundamental to symmetrical encryption algorithms so understanding, at a basic level, how they work will benefit me in the design stage.

The following is an example of using XOR gates to encrypt and decrypt a nibble.I will use 1010 as the plain text to be encrypted and 1101 as the encryption key.

```
Unset
Plain text = 1010
Key = 1101

1010 ⊕ 1101
-----------
       1010
  XOR  1101
-----------
       0111
-----------

Cipher text = 0111
Key = 1101

0111 ⊕ 1101
-----------
       0111
  XOR  1101
-----------
       1010
-----------

Decrypted Text = 1010 = Plain Text
```

Below is a basic overview of how AES works. This will inform the development of the pseudo code in my design stage.

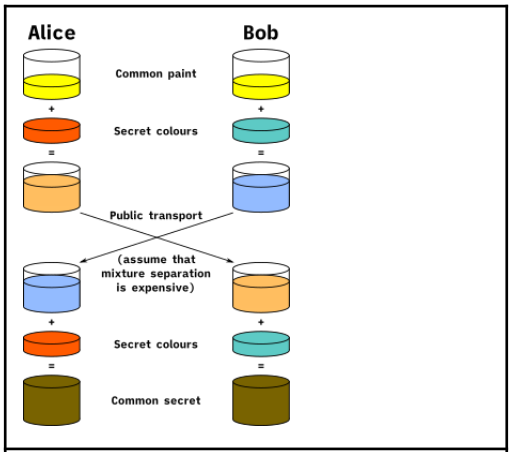


input : Plaintext Block $ptxt_b$, Secret Key sk
output: AES state $state$
$state = InitState(ptxt_b, sk)$
$AddKey(state, sk_0)$
for $i = 1$ **to** $n_r - 1$ **do**
  $SubBytes(state)$
  $ShiftRows(state)$
  $MixColumns(state)$
  $AddKey(state, key_i)$
$SubBytes(state)$
$ShiftRows(state)$
$AddKey(state, key_{n_r-1})$

AES Flowchart

Diffy-Hellman is a way of ‘securely exchanging cryptographic keys over a public channel. It works on the premise that combining numbers in a specific way is very easy to do one way but incredibly difficult to do another way.

Below is a diagramed example of this which will help inform me in my design stage.



Diffie Hellman Key Exchange Diagram

Complying with the GDPR

As my program will store data it needs to comply with the GDPR

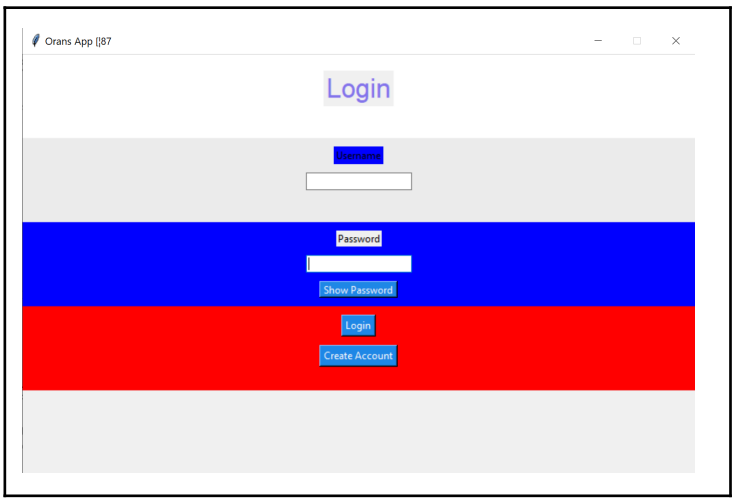
Point	How I will comply
Encrypt personal data wherever possible	All personal data stored will be encrypted
Users can request and receive all information you have on them	Button where all data stored on that user is sent to them
Users can correct or update any inaccurate or incomplete information	Allow user to update their screen name
Easy for customers to request to have their personal data deleted	When user deletes account all personal data held is deleted along with it

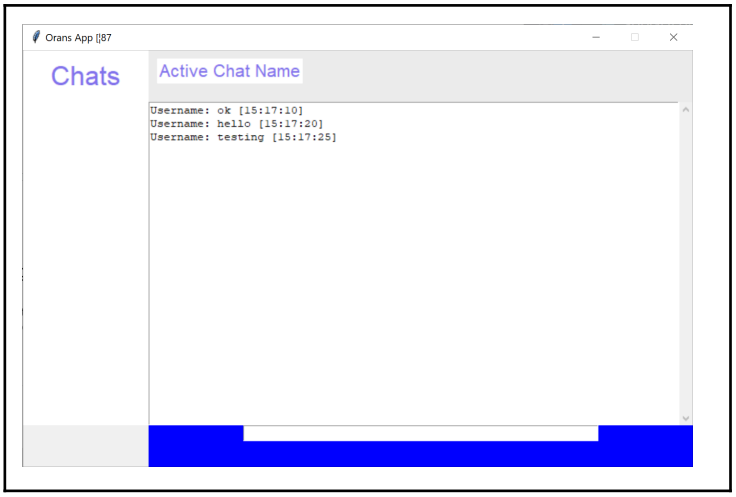
Prototyping

I am prototyping the server, client and GUI to see if the python modules I am thinking of using with them allows me to fulfil all of my objectives.

GUI prototype

(Appendix Analysis, GUI Prototype Code)





GUI prototype findings

1) I am able to link function to buttons:

```
Python
showPasswordButton = tk.Button(passwordFrame, fg=cWhite, bg=cButton,
activebackground=cButtonActive, activeforeground=cWhite, text='Show
Password', command=lambda: self.toggle_password(passwordEntry,
showPasswordButton))
```

2) I am able to have functions manipulate existing elements on the frame to a high level of specificity such as in this show/hide password function:

```
Python
def toggle_password(self, passwordEntry, showPasswordButton):
	if passwordEntry.cget('show') == '':
		passwordEntry.config(show='*')
		showPasswordButton.config(text='Show Password')
	else:
		passwordEntry.config(show='')
		showPasswordButton.config(text='Hide Password')
```

3) Using an OOP will be vital in creating the GUI as it allows me to change frames with a controller class and allows each frame to be its own class making for easier coding

4) While tkinter is versatile it is incredibly time consuming to create a complex UI. To resolve this I have found a program that allows me to create a design in Figma and transform it into code using tkinter which allows me to quickly create a simple structure for the GUI before adding functionality. (ParthJadhav, no date) For the output code of tkinter designer see **(Appendix, Analysis, Tkinter-Designer Output)**

5) There could be potential issues with setting graphical text to say certain things, for example the user's screen name, as all tkinter elements are created in the init function which is called when the class is created not when it is shown. To resolve this I will have a create_and_place function to place GUI elements and an on_show function which is called when the frame is shown not when it is initiated.

Summarised GUI prototype findings

1) Tkinter is a viable module to use for my GUI
2) I will use Tkinter Designer along with Figma to quickly layout the tkinter designs
3) I will ensure that each class will have a create_and_place function and an on_show function

Client Server prototype

(Appendix Analysis Client Server Prototype Code)

After researching and following tutorials, (*All About Python, 2021b*) and (*Bek Brace, 2020*), on how to build a simple client server app in python I came out with some key findings.

Client Server prototype findings

1) I will need to implement error handling throughout especially in regards to handling a client disconnect otherwise an error will occur

```
Program is running
Server running
Successfully connected to client 127.0.0.1 50891
('Oran', <socket.socket fd=352, family=2, type=1, proto=0, laddr=('127.0.0.1', 80), raddr=('127.0.0.1', 50891)>)
b'Oran~hi'
Exception in thread Thread-2 (listen_for_messages):
Traceback (most recent call last):
  File "C:\Python311\Lib\threading.py", line 1038, in _bootstrap_inner
    self.run()
  File "C:\Python311\Lib\threading.py", line 975, in run
    self._target(*self._args, **self._kwargs)
  File "C:\Users\orank\OneDrive\Desktop\Python Server 2\server.py", line 14, in listen_for_messages
    message = client.recv(2048).decode('utf-8')
              ^^^^^^^^^^^^^^^^
ConnectionResetError: [WinError 10054] An existing connection was forcibly closed by the remote host
```

Client Disconnect Error

2) I need to find a way to serialise data to ensure that any combination of characters can be send and not cause any potential error
3) Printing useful information is vital for debug purposes
4) Currently the only way I am differentiating the users username and the messages sent is when the data is received in the code. For this small model this is ok but before I expand it I will need to find a way of differentiating between many different types of packets.

```python
def client_handler(client):
  # server will listen for clinet message containg the username
  while 1:
```

```
    # First packet recieved will always be treated as a username
    username = client.recv(2048).decode('utf-8')
    if username != '':
      active_clients.append((username, client))
      break

    else:
      print('Client username empty')

  #subsiquent packets are treated as a message
  threading.Thread(target=listen_for_messages,
          args=(client, username,)).start()
```

Current way of receiving data

5) Implementing a protocol for sending and receiving packets will be useful as currently I can only receive a max of 2048 bytes

Summarised Client Server prototype findings

1) Need to find out how to handle disconnect errors
2) Need to find a format for data serialisation
3) Find a way to differentiate the contents of different packets
4) Write a protocol allowing any size message to be received

Objectives

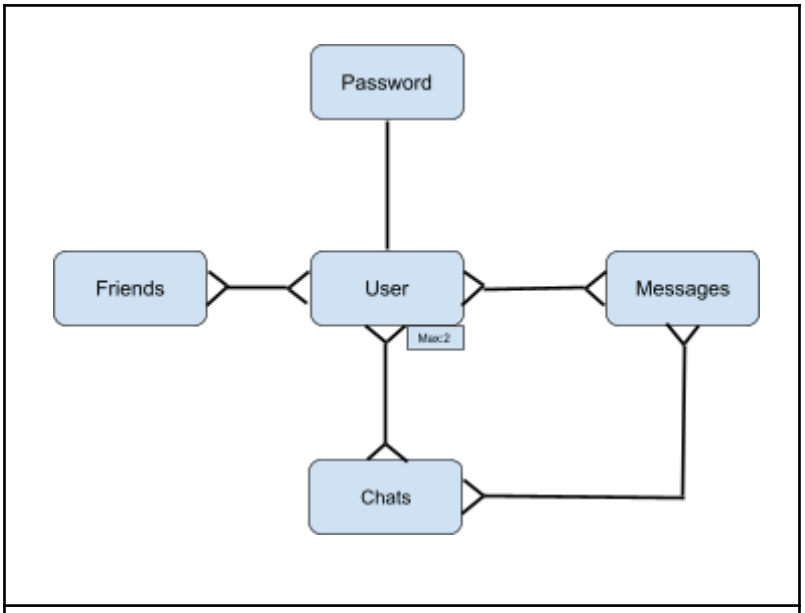
1. All new users must be able to create an account
    a. The new user must complete a form where they enter their Username, Screen-name, Password.
    b. The Username must be unique to all the usernames on the app.
    c. The Screen-name does not have to be unique
    d. Password must be entered twice for verification
    e. The password must be at minimum 8 characters long, at maximum 64 and contain numbers letters and special characters
    f. The user must be able to show/hide their entered password
    g. Passwords must be hashed before they are stored
        i. The hashing process must involve the use of salt and pepper to ensure maximum security
        ii. The algorithm to hash the password must be Bcrypt
        iii. The pepper must be stored securely outside of the database holding the passwords
2. To login the user must enter their username and password
    a. There must be no way to recover or reset a password
    b. The user must be able to show/hide their entered password

3. The user must be able to add a new friend by typing the username of the person into a specific box
    a. The user must be able to block any of their current friends provided they themselves are not already blocked
    b. The user must be able to unblock any of their current friends provided they themselves are not already blocked
4. The user must be able to see their outgoing friend requests
5. The user must be able to see their incoming friend requests
6. The user must be able to permanently delete their account
    a. Other users must not be able to send messages to or see messages between them and a deleted account
    b. The user must be asked twice before they can delete their account
7. The user must be able to request all data with the exception of the hashed password from the server
    a. The user must be able to choose where this data is stored
8. The user must be able to change their screen-name to whatever they want provided it does not interfere with the code in any way.
9. The user must be able to send text, emoji or images to another user
    a. These messages must be encrypted end to end using AES and a key known between the two users
    b. These messages must be stored on the server in an encrypted form (changed to ‘stored locally in an encrypted form’ see: Server Side Storage Approach)
    c. These messages must include date and time of sending
    d. The user must be able to delete any of their sent messages provided they have not blocked the user the messages has been sent to or that user has not blocked them.
    e. The deleted message must me replace with ‘Message Deleted’
    f. The user must not be able to send any messages to or receive any messages from a blocked friend or a friend they have blocked
10. The user must be able to see the message history between them and a friend they have blocked
11. The user must not be able to see the message history between them and a friend who has blocked them.
12. When the user closes the application all changes they have made must persist
13. The system must have an ‘easy to use’ UI where ‘easy to use’ is defined as meeting all sub objectives below
    a. The UI must use understandable symbols wherever it is trying to communicate a message through the use of those symbols
    b. For all actions taken by the user the user must be able to visually see their action has had an effect
    c. There must be placeholder text where the user needs to enter text that is removed when they go to enter text

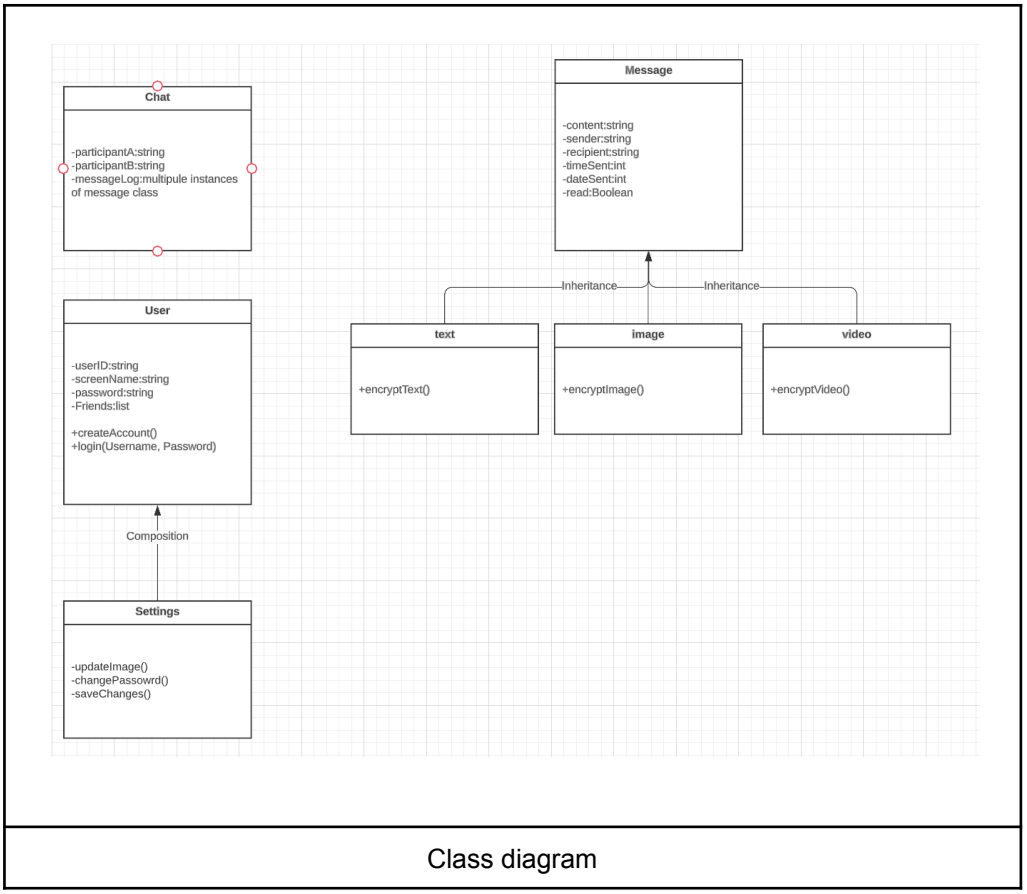
Proposed Solution

I will create a messaging app using python that can be downloaded and run on other devices that complies with all objectives listed above.

Diagrams to inform the Design stage



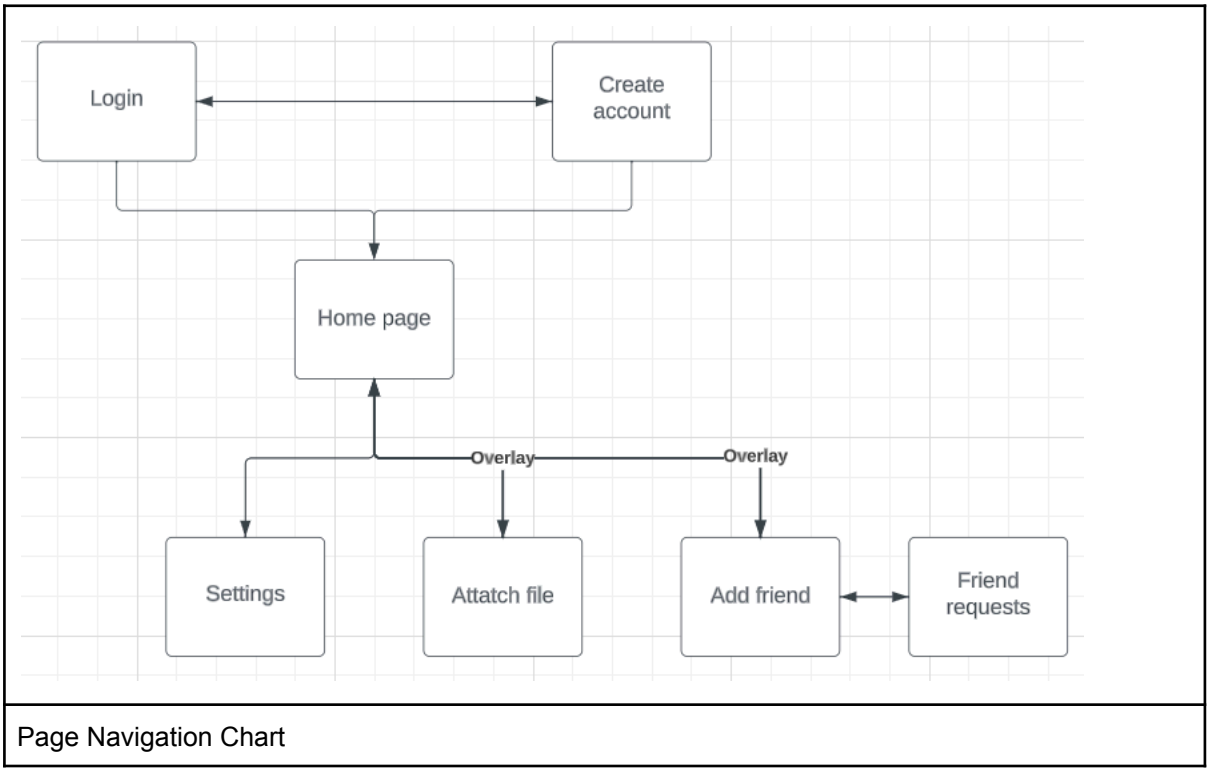
Entity-relationship diagram



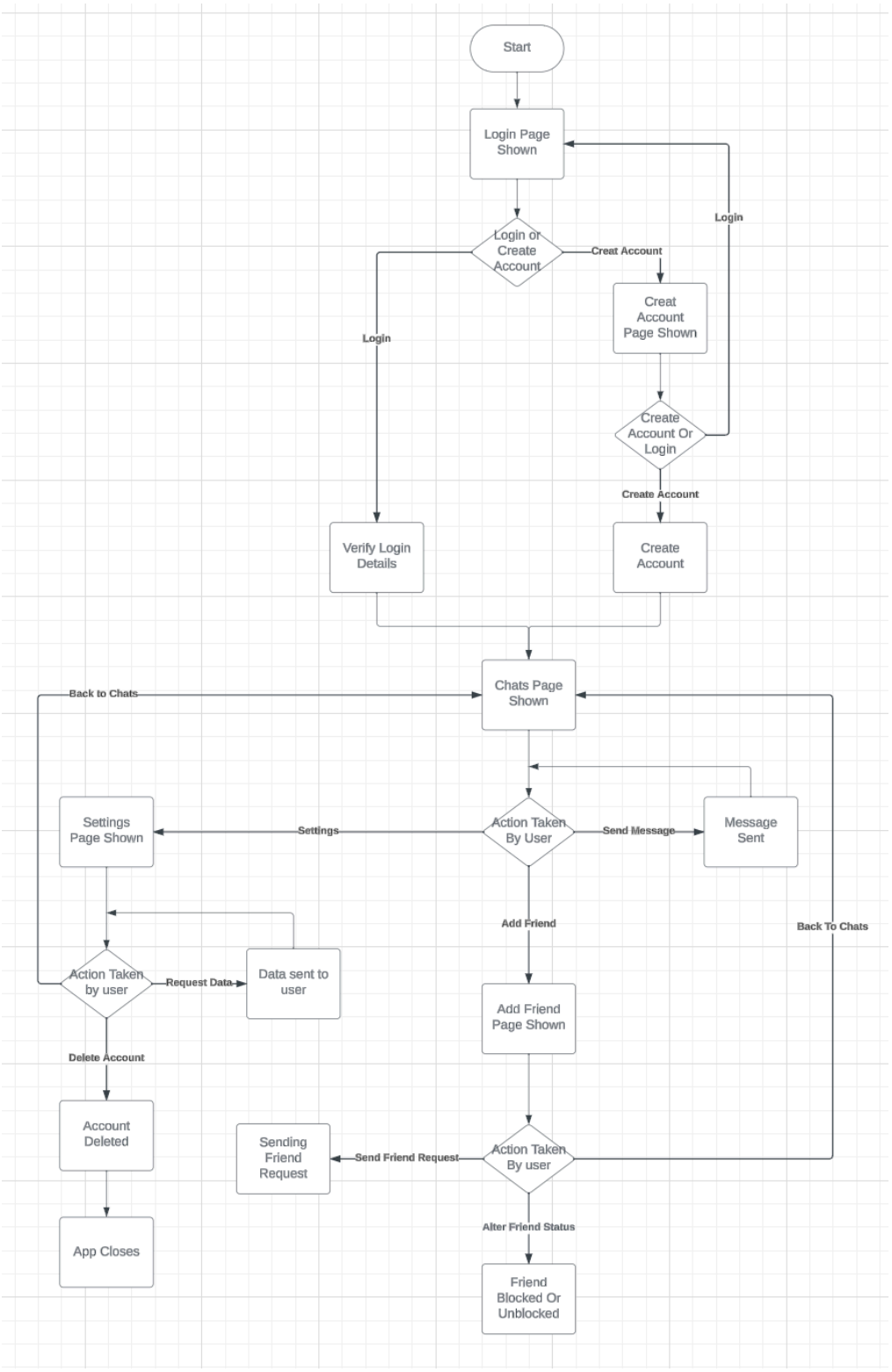
Class diagram

Design

Diagrams



Page Navigation Chart



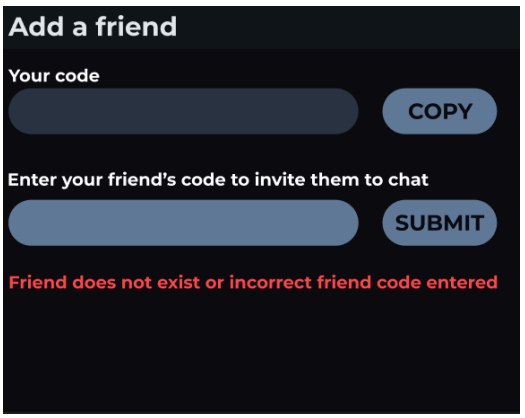
System Flowchart

UI

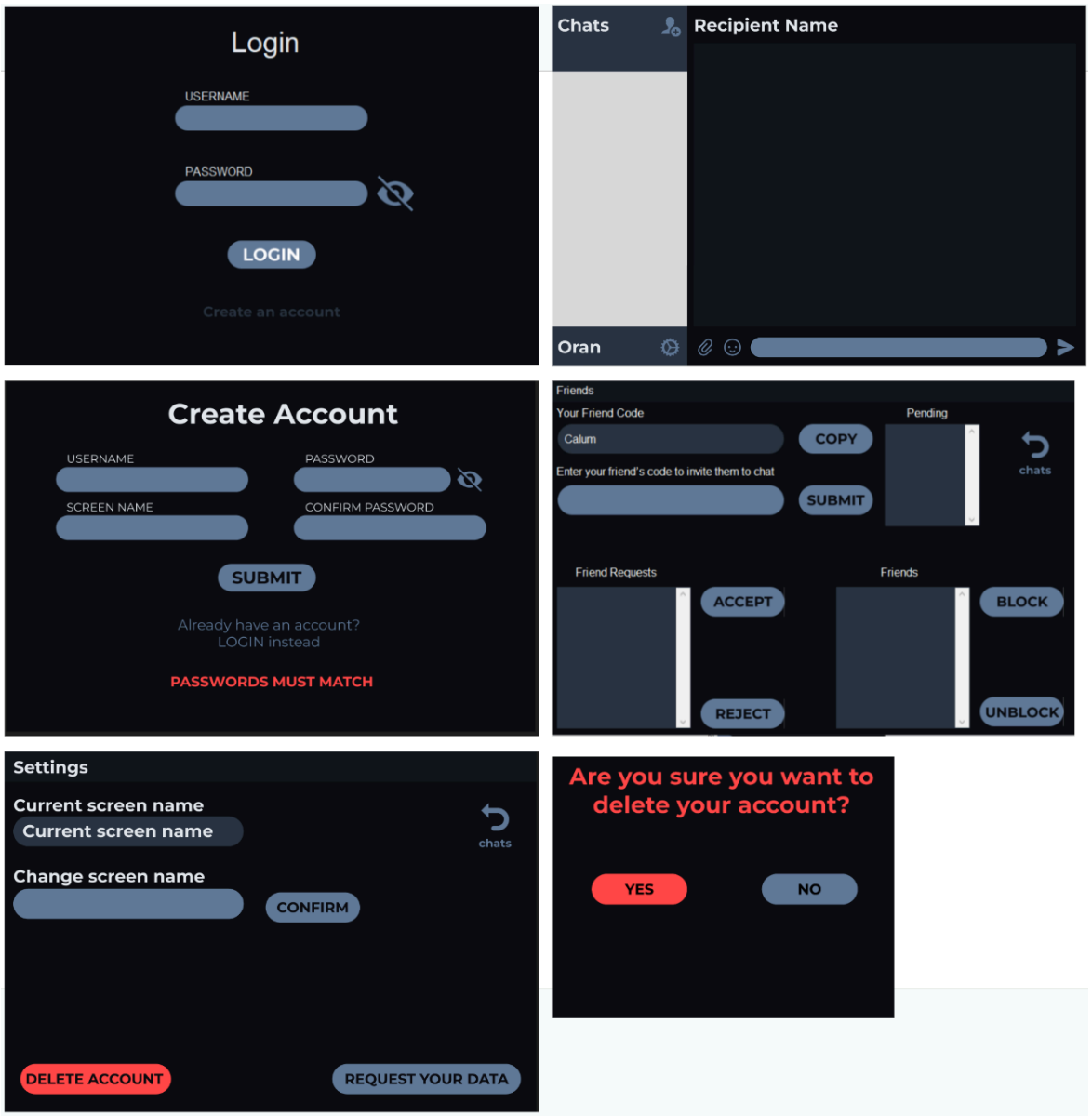
When designing the UI I made sure to keep in mind my objectives and all I had said about the personal critique of my own UI. No temp text is shown in the design as that is added in tkinter.

Once each design was created it did not deviate for the rest of the project with the exception of the Add Friend Page.

Initially the Add friend page was to be a popup window where a user could add a friend. However the major flaw with this design is that the user could not see any incoming friend requests or outgoing friend requests so I updated the design as seen in the collage of all the UI designs below.



Add Friend UI Version 1



All UI Designs

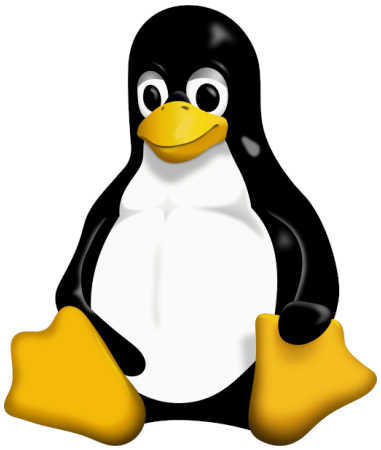
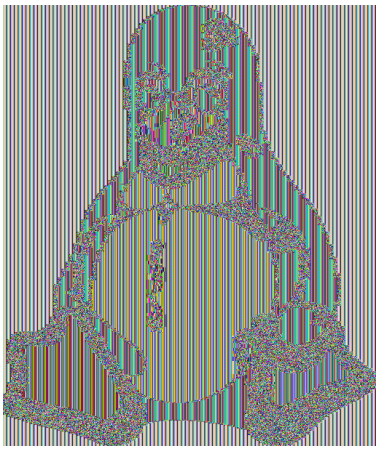
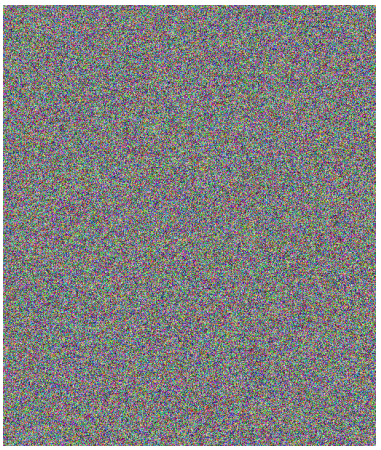
Algorithms

Advanced Encryption Standard

All information unless otherwise referenced was learnt from (*Francisrstokes, no date*)

Advanced encryption standard (AES) is a symmetric block cipher. Symmetric refers to the idea that the same key is used for encryption and decryption. Block refers to how the plain text is encoded.

There are two main types of AES. Electronic code book (ECB) and Cypher block chaining (CBC). ECB is how the algorithm would be used in its most basic form with no additional changes or alterations. The issue with ECB is that in certain cases it can 'leak' some structural information. This is because ECB here is working almost like a hash in that for the same input it will always return the same output. This is because, for each block of data, we are encrypting it using the same key schedule meaning that the output is the same. If we take a very simplified example of the string "beekeeper" and put it through some sort of encryption process where for each of the same inputs the same output is returned the ciphertext could end up looking something like this "pwwjwwxwa". Here we can see that the overall structure is retained and someone using frequency analysis could determine that the w is an e and then go on to decrypt the rest of the text. A more pictographic example is that of the penguin and it demonstrates this weakness visually.

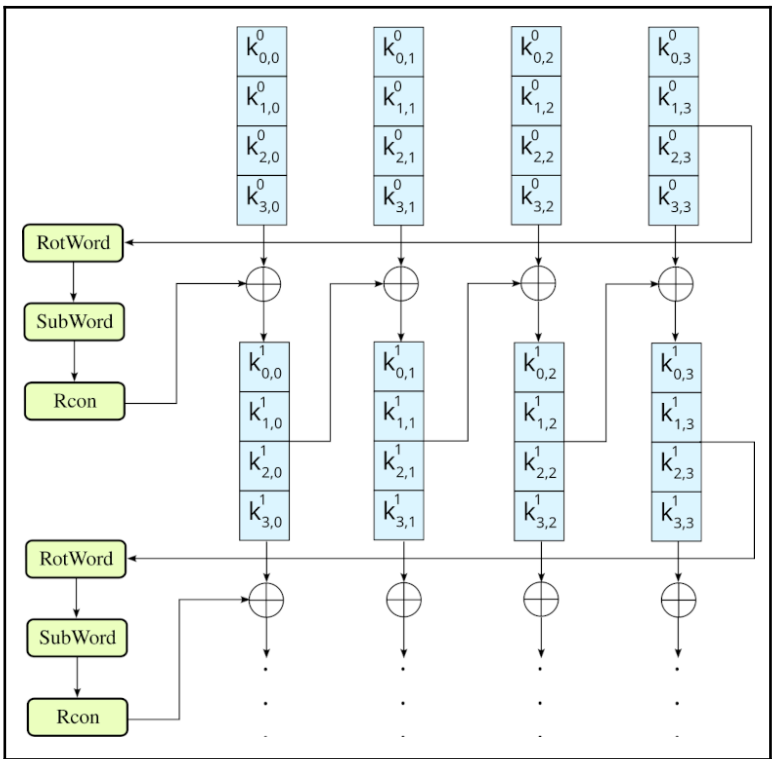
Plaintext (*Francisrstokes, no date*)	ECB Encryption (*Francisrstokes, no date*)	CBC Encryption (*Francisrstokes, no date*)
		

Due to the weakness of the ECB I will use CBC.

Key Expansion
Before any encryption takes place we need a key or set of keys. AES starts with a 128-bit key and expands it into a series of round keys called the key schedule which are then used for encryption. The initial key must be 128 bits long. It can be longer but that would mean having to adjust how many rounds of encryption occur and for the level of security needed in my project I am satisfied with 128 bits. Here are the step of the key expansion process 1) Append the key to the key schedule

2) Split the key into 4 columns of 32 bits
3) Transform the last column through 3 operations
    a) Rotate word
    b) Substitute word
    c) Round constant
4) Take this newly transformed column and XOR it with the 1st column of the previous round key to generate the 1st column key of the next round key.
5) XOR this new column with the 2nd column of the previous key to generate the 2nd column of the next round key
6) XOR the new column we have made with the 3rd column of the previous round key to generate the 3rd column of the next round key
7) XOR the new column we have made with the 4th column of the previous round key to generate the 4th column of the next round key
8) These 4 newly made columns are the new round key. Append them to the key schedule.
9) Take the last column of the newly made key and repeat from step 3.

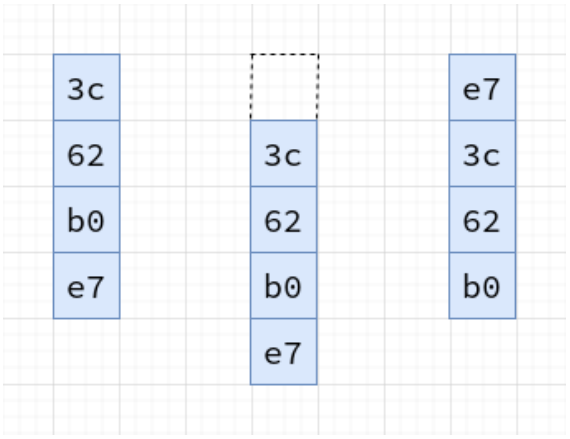
The image below diagrams this process



Key Expansion Diagram
(*Francisrstokes, no date*)

Rotate Word

This function makes use of a circular queue. All items are pushed forward one space and the item at the front of the queue is moved to the back. The only difference is that there are no pointers marking the front and end of the queue.



Rotate Word Diagram
(*Francisrstokes, no date*)

Substitute Word

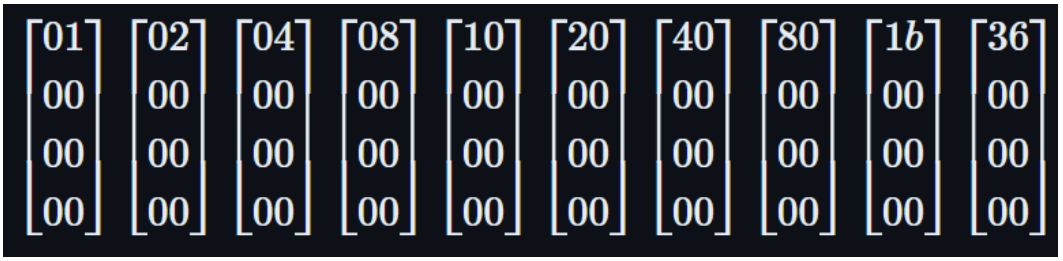
This function uses a lookup table called an S-box.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Sbox Table
(*Francisrstokes, no date*)

Each byte in the word is substituted for the corresponding byte in the S-box. For example if the first byte was 12 (10 02) the corresponding byte in the table would be c9 (c0 09).

Round Constant
The column is XORed with a predefined constant column corresponding to the current round in the key expansion process.



Constant Column
(*Francisrstokes, no date*)

Pseudo code of AES Key Expansion

```
Unset
const s_box = 2D lookup table
const constant_column = Matrix
key = 128 bit string

function key_preperation(key)
	prepared key = ''
	for each character in key
		binary = convertToBinary(unicodeValue(character))
		preapred_key += binary
	return prepared_key



function key_expansion(key)
	key_schedule = empty list
	for i in range(11)
		key_schedule.append(np.array((4,4)))

	#adding the first key to the key_schedule
	for i in range(0, len(key), 8):
		key_schedule[0][every 4][every 1] = key[i:i+8]
	#key expansion stage
	for i in range(10)
		final_column = key_schedule[i][3]
		transformed_column =
round_constant(sub_word(rot_word(final_column)), 1)

		for j in range(4)
			transformed_column = xor(key_schedule[i][j],
transformed_column)
			key_schedule[i+1][j] = transformed_coulumn
```

```
function xor(init_column, transformed_column)
	column = []
	for i in range(4)
		xor = init_column[i] xor transformed_column[i]
		column.append(xor)
	return column


function rot_word(column)
	#circular list
	return column[final number] + column[everything but final number]

function sub_word(column)
	row_nibble = base10 value of column[:4]
	column_nibble = base 10 value of column[4:]
	sub_value = binary(s_table[row_nibble][column_nibble])
	return sub_value

function round_constant(column, i)
	x = int(sub_value)
	y = constant_column[i][0]
	round_constant = x ^ y
	return binary(round_constant)
```

Encryption

Once the key schedule is completed we can encrypt the data.

For each block (16 bytes) of plaintext input the following occurs

1) XOR block with first round key
2) Apply the following operations
    a) Substitute bytes
    b) Shift Rows
    c) Mix Columns
    d) XOR with next round key
3) Repeat step 2 9 times therefore making use of all but 1 round keys
4) Substitute bytes
5) Shift Rows
6) Add final round key

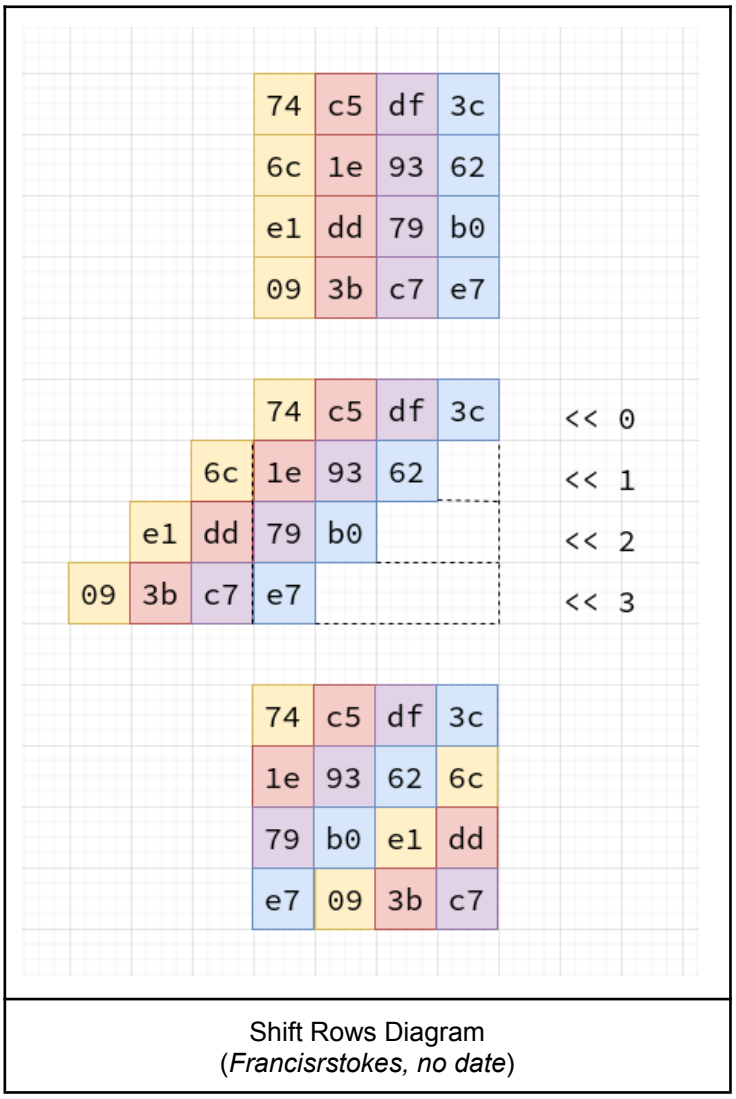
Many of the functions vital to the encryption process are also used in the key expansion process.

Substitute bytes

Same as key expansion

Shift Rows

Each Row is shifted as many bytes as its order -1.



Shift Rows Diagram
(*Francisrstokes, no date*)

This uses the same circular queue idea as the Rotate Word function.

```
Unset
def shift_rows(block):
	for i in range(4):
		block[i] = block[i][i:] + block[i][:j]
	return block
```

Mix Columns

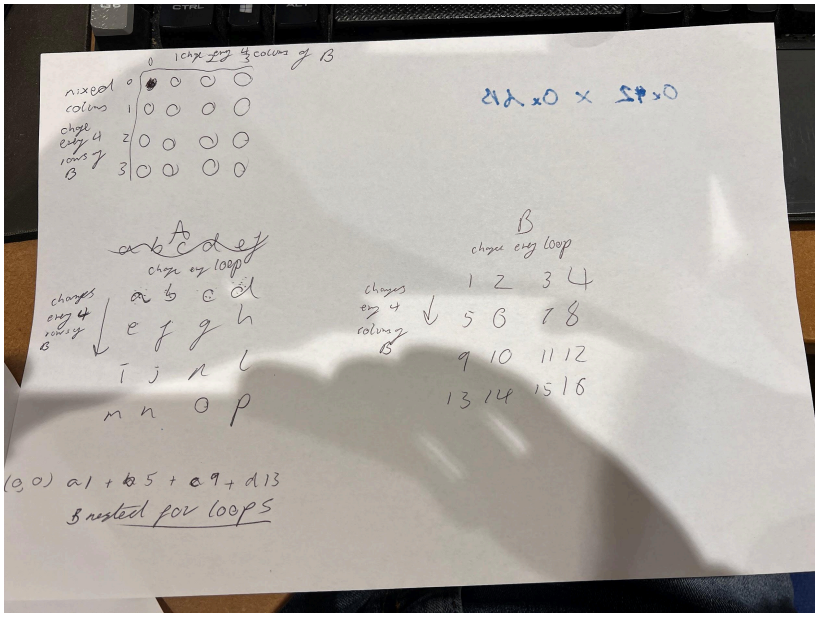
Each column is multiplied with a matrix; however both addition and multiplication here take place in a finite field called GF(2^8).
The point of this function is to provide diffusion to the algorithm. This means that changing a single bit in the plaintext leads to approximately half of the bits in the ciphertext to change.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

Matrix Multiplication Diagram

Matrix multiplication **(Appendix Technical Solution AES Code Version 2(161-174))** was tough to understand but fairly easy to implement.

First I consulted (*Chirag Bhalodia, 2022*) to get a basic understanding of matrix multiplication and then worked through the function stype by step on paper using a drawn out version of the 4x4 blocks.



Matrix Multiplication

This led me to realise that the movement of each row and column for each block was just a repetition and only 3 repetitions were used so I knew I had to use 3 nested for loops each with a range of 4.

Design of mix columns function

```
Unset
MATRIX = Predefined 4x4 2d list
def mix_columns(block):
       mixed_columns = 4x4 2d list of 0s
       for i in range(4):
              for j in range(4):
                     for k in range(4):
                            mixed_columns[i][j] XOR gf_mult(block[i][k],
MATRIX[j][k]
```

GF2^8

This will allow me to multiply two numbers together in the mix columns function.

Having multiplication take place in GF(2^8) has a number of advantages.

1) Computers handle operations is GF(2^8) extremely well due to them being bitwise operations
2) The algorithm is less computationally expensive due to the use of bitwise operations
3) Due to the specific properties of GF(2^8) it provides resistances against specific attacks, the details of which are outside the scope of this NEA

Below is an explanation of how GF(2^8) works:

In GF(2^8) addition is simply an XOR operation but multiplication is a lot more complex.

Firstly we need an understanding of how numbers are represented in GF(2^8)

Our place value system is essentially a polynomial with a series of different coefficients for each number. The coefficients can take the values of 0-9.

For example 123 can be represented as $1 \bullet 10^2 + 2 \bullet 10^1 + 3 \bullet 10^0$ and so on.

Binary works much the same except instead of 10 the base is two and the only coefficients are 0 and 1.

However what happens if we have an arbitrary base called x and the only coefficients are 0 and 1?

This is what GF(2^8) is and the fact that the only coefficients are 0 and 1 means it maps very well onto binary

For example the binary number 10001101 can be written as $x^7 + x^3 + x^2 + 1$ in GF(2^8)

Multiplication, therefore, in GF(2^8) is the multiplication of two polynomials together. Whenever two like terms are added they cancel out as there are only two possible coefficients 0 or 1. Finally you mod this number with the irreducible polynomial, in this case 0x11B, which ensures that all numbers remain within the finite field.

Now I can begin to develop an algorithm to multiply two numbers in GF(2^8)

```
Unset
#Multiplication is repeated addition of the same number so for each bit
set in b we can add (XOR) a to the result
def gf_mult(a,b):
	if b == 1: #If b == 1 the result is a as any number * 1 is itself
	result = a
	for each bit in b:
		if b == 1:
			add (XOR) a to result
		if a_msb == 1:
#Multiplying by the irriducible polynomial here saves us from having to do
it repeateldy with another for loop at the end
			a add (XOR) 0x11b
	return result
```

Client Server

Data serialisation

The best way to serialise data would be using JSON as that is the format comly used when sending data between a client and a server. While I do not yet have a strong idea of exactly what I will need to be sending, JSON can serialise most data types so I feel it is suitable for whatever I may need to send. (*O3DE: JSON serialization of O3DE data Types*, no date)

Differentiating packet contents

The best way to do this would be to use dictionaries with key value pairs. That way I can easily distinguish each item while also maintaining code readability rather than if I added every item I need to send to a list. This also works well as dictionaries can be easily converted into JSON data in python.

Designing the packet_header protocol

The first protocol I had to create was the packet_header protocol. When you need to receive data through a socket you need to specify the maximum amount of data to be received. (*Python socket receive - incoming packets always have a different size*, no date) If I set this to be a constant there is always a chance someone will send a message slightly bigger than the constant. To resolve this I designed the packet_header protocol by looking at how others resolved this and finally settling with a protocol created by (*Tech With Tim, 2020*)

```
Unset
HEADER = 64
```

```
def send_data(data):
	data <- json.dumps(data)
	data <- data.encode('uft-8')
	send_to_server(packet_header(data))
	send_to_server(data)

def packet_header(data): example = 1234
	data_length <- get length of data 3
	send_length <- turn data_length into bytes  b'3'
	get the number bytes to pad
	padding <- HEADER - len(send_length) == HEADER - 1 == 63
	send_length <- send_length + (one byte 'padding' times)

def recive_data():
	data_length <- recieve data from server size(HEADER)
	data_length <- decode data_length from bytes format
	data_length <- int(data_length)
	json_data <- recieve data from server size(data_length)
	data = json.loads(json_data)
```

Using this protocol I can design the structure for the client server code

```python
FORMAT = 'utf-8'
HEADER = 64
class Client:
	def __init__(self):
		self.user_id = None
		self.public_key = None
		self.connection_failed = None
		self.screen_name = None
		self.friends = None
		self.disconnect = False

	def connect(self):
		try:
			#connect to server
		except:
			print("[ERROR MESSAGE]")

	def start_listen_thread(self):
		#start thread on self.listen()

	def send_data(self, data):
		data = json.dumps(data)
```

```
            data = data.encode(FORMAT)
            #send to server(self.packet_header(data))
            #send to server (data)

      def packet_header(self, data):
            data_length = len(data)
            send_length = str(data_length).encode(FORMAT)
            padding = HEADER - len(send_length)
            send_length += b' ' * padding
            return send_length

      def recive_data(self):
            data_length = #wait to recieve data size(HEADER)
            data_length = data_length.decode(FORMAT)
            data_length = int(data_length)
            json_data = #wait to recieve data size(data_length)
            data = json.loads(json_data)
            return data


      def listen(self):
            while self.disconnect != True:
                  data = self.recive_data()
                  #do whatever is needed with data
```

Permanent data storage

Database Normalisation

To create a fully normalised database I am going to create an unnormalized one to begin with and then normalise it step by step until it's fully normalised. This will ensure it finishes as a fully normalised database.

userID	screenName	password	friends	messages
b0b	Bob	password	bob, oran	hi, how are you nice to see yoy
al1c3	Alice	1234	bob, oran	hi, im great, wow, what a nice house
0r4n	Oran	fluffy	bob, alice	hello no thanks but no

First Structure

This table is not in First Normal Form (1NF) as it has multi_valued attributes in columns ‘friends’ and ‘messages’. In addition it matters in what order the data is stored which also violates 1NF

Removing these multi_valued valued attributes would cause a lot of data duplication despite being in 1NF as it is not complying with 2NF.

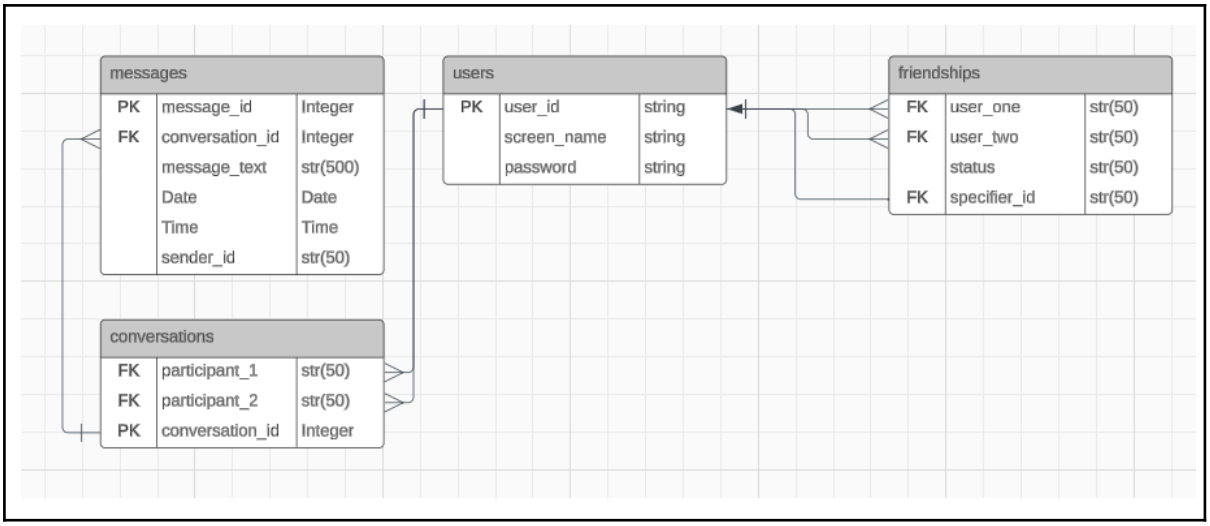
users				friends	
userID	screenName	password		userID	friendID
b0b				b0b	al1c3
al1c3				b0b	0r4n
0r4n				al1c3	0r4n
				al1c3	b0b
messages				0r4n	b0b
				0r4n	al1c3
messageID	userID	messageText			
1	b0b	hi			
2	al1c3	hi			
3	0r4n	hello			

Second Database Design

A database like the above removes the duplicated data and the multi_valued attributes.

After more adjusting I ended up with a database represented in the E-R diagram below

Normalised E-R diagram



Database structure in SQL

```
Unset
CREATE TABLE users (
        user_id text NOT NULL PRIMARY KEY,
```

```
        screen_name text NOT NULL,
         password text NOT NULL,
        UNIQUE(user_id)
        );

CREATE TABLE  conversations (
        conversation_id integer PRIMARY KEY,
        participant_1 text NOT Null,
        participant_2 text NOT Null,
        FOREIGN KEY (participant_1) REFERENCES users(user_id),
        FOREIGN KEY (participant_2) REFERENCES users(user_id),
        CHECK (participant_1 != participant_2)
        );

CREATE TABLE IF NOT EXISTS messages (
        message_id integer PRIMARY KEY,
        conversation_id INT,
        message_text text NOT Null,
        date text NOT NULL,
        time text NOT NULL,
        sender_id text NOT NULL,
        FOREIGN KEY (conversation_id) REFERENCES conversations(conversation_id)
        );
```

Technical Solution

Where Technical Skill Are Demonstrated

Technical Skills Group A	How I am meeting it	Where is it demonstrated
Complex data model in database: • Cross-table parameterised SQL • User/CASE-generated DDL script	Cross-table parameterised SQL: Retrieving needed data User/CASE-generated DDL script: Creating databases Other: Normalised relational database	Server Side Storage Approach
Advanced matrix operations	Matrix multiplication as part of the AES algorithm	Version 3 - OOP Approach
Complex mathematical model (GF2^8 and AES)	Multiplication in the Galois field 2^8 as part of the AES algorithm	Galois field 28
Complex user-defined algorithms (eg optimisation, minimisation, scheduling, pattern matching) or equivalent difficulty	Advanced Encryption Standard Algorithm	Developing the server Developing the Backbone Developing the client
Dynamic generation of objects based on complex user-defined use of OOP model	Server creates a new instance of an object to handle each new user connected. Each page in the app is a new object that is instantiated when the user moves to that page	Handling Connections and Disconnections OOP Approach version 2
Complex user-defined use of object-oriented programming (OOP) model, eg classes, inheritance, composition, polymorphism, interfaces	Client and Server inherit from a base class that holds all of their dual functionalities such as sending and receiving data. Base class inherits from AES encrypt and decrypt classes Composition in GUI to allow client and gui to communicate Polymorphism when using the client and server classes as they have slightly different functions	Developing the server Developing the client Developing the Backbone OOP Approach version 2
Complex client-server model: • Server-side scripting using request and	Entire Client Server model	Developing the server Developing the Backbone Developing the client

response objects and server-side extensions for a complex client-server model • Parsing JSON to service a complex client-server model		
parsing JSON/XML to service a complex client-server model	Data sent between client and server is serialised using JSON and other methods	Data Serialization Developing the Backbone
Defensive programming	Preventing SQL injection Try and accept statements to catch any errors that may occur.	Server and Client side storage approach Defensive Programming
Queues	Message queue	Developing the server

Advanced Encryption Standard

Version 1 - Failed Approach

The way I initially approached AES made coding overly complex. I could tell my key expansion function was not working as desires as I was checking my outputs using: (*CrypTool Portal*, no date)

The Approach

Much of my initial problems when coding AES were in relation to the constant switching of data types and using strings instead of binary values. Not only did it make it much harder to code but also harder to debug as I wasn't able to easily compare my output against the output of an exemplar AES algorithm. Furthermore I didn't fully understand each step of the AES algorithm fully.

Due to my desire to use strings I had to convert every bit of plaintext into binary and then format it to remove the b that python puts at the start of printed binary.

```python
def key_to_bin(key):
  binary = ''
  #{:0>8} means fill with 0 up to a max of 8 characters and align these zeros on the left
  for character in key:
    binary += ('{:0>8}'.format(format(ord(character), 'b')))
  return binary
```

This created additional complexities in the rest of the code as I couldn't use inbuilt bitesize operations but instead had to code my own which invariably resulted in numerous syntax and logic errors.

```python
def xor(init_column, transformed_column):
 column = []
 for i in range(4):
  xor = int(init_column[i], 2) ^ int(transformed_column[i], 2)
  xor = '{:0>8}'.format(format(xor, 'b'))
  column.append(xor)
 return column
```

Problem	Solution
Using strings of binary made it very difficult to code and debug as I had to constantly switch data types which often required loops as I had these strings in lists. In addition, because I was using strings of binary in a format python did not recognize, to convert it back and forth I had to use specific functions which only added to the complexity.	After looking into how *Boppreh (no date)* and *Tasos-Py (no date)* went around implementing AES in python I realised that one version made use of *.encode()*. After experimenting with how it worked I realised it was the perfect function for working with binary as it converted a string into an iterable list of bytes.
If the user was using special characters an error could occur as they would not be part of the ASCII set.	I passed *'utf-8'* as a parameter into *.encode()* meaning that it makes use of the unicode set which allows for a much larger character set.
I didn't fully understand AES yet.	I looked at other explanation of aes (*https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf*) (*https://uomustansiriyah.edu.iq/media/lectures/5/5_2021_06_05!07_10_53_PM.pdf*) (*Computerphile, 2019*) (*Chirag Bhalodia, 2022a*) and re-read (*Francisrstokes, no date*) to get a full understanding.

Afterwards I began to re-code AES ensuring I was using the bytes encoded format rather than a string of bytes for my data.

Inverse S-box

While I did fail to create a working key expansion algorithm I did successfully write code that created the inverse s-box.

Function name: Create Inverse S-Box	**Purpose:** To create the inverse s box needed for the AES algorithm using the predefined s

	box table

Explanation: For the AES encryption I need a few constants. Two of these are lookup tables. Once called s_box used for encryption and another called inverse_s_box used for decryption. I managed to find the initial s_box online however I still needed to create the inverse_s_box. I decided to print it in a list format and then copy and paste it into my actual code for use as a constant.

Code:

```
import numpy as np

s_box = [
    [0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76],
    [0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0],
    [0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15],
    [0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75],
    [0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84],
    [0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf],
    [0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8],
    [0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2],
    [0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73],
    [0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb],
    [0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79],
    [0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08],
    [0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a],
    [0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e],
    [0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf],
    [0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16],
]

inverse_s_box = np.zeros((16, 16))

for a in range(len(s_box)):
    for b in range(len(s_box[0])):
        value = s_box[a][b]
        coordinates = '{:0>2}'.format(hex(value).split('x')[-1])
        row = int(coordinates[0], 16)
        column = int(coordinates[1], 16)

        inverse_coords = '{:0>2}'.format(hex(s_box[row][column]).split('x')[-1])
        inverse_row = int(inverse_coords[0], 16)
        inverse_column = int(inverse_coords[1], 16)
        inverse_s_box[inverse_row][inverse_column] = s_box[a][b]

print(inverse_s_box.tolist())
```

Output:

```
[[82.0, 9.0, 106.0, 213.0, 48.0, 54.0, 165.0, 56.0, 191.0, 64.0, 163.0, 158.0, 129.0, 243.0, 215.0, 251.0], [124.0, 227.0, 57.0, 130.0, 155.0, 47.0, 255.0, 135.0, 52.0, 142.0, 67.0, 68.0, 196.0, 222.0, 233.0, 203.0], [84.0, 123.0, 148.0, 50.0,
166.0, 194.0, 35.0, 61.0, 238.0, 76.0, 149.0, 11.0, 66.0, 250.0, 195.0, 78.0], [8.0, 46.0, 161.0, 102.0, 40.0, 217.0, 36.0, 178.0, 118.0, 91.0, 162.0, 73.0, 109.0, 139.0, 209.0, 37.0], [114.0, 248.0, 246.0, 100.0, 134.0, 104.0, 152.0, 22.0, 212
.0, 164.0, 92.0, 204.0, 93.0, 101.0, 182.0, 146.0], [108.0, 112.0, 72.0, 80.0, 253.0, 237.0, 185.0, 218.0, 94.0, 21.0, 70.0, 87.0, 167.0, 141.0, 157.0, 132.0], [144.0, 216.0, 171.0, 0.0, 140.0, 188.0, 211.0, 10.0, 247.0, 228.0, 88.0, 5.0, 184.0
, 179.0, 69.0, 6.0], [208.0, 44.0, 30.0, 143.0, 202.0, 63.0, 15.0, 2.0, 193.0, 175.0, 189.0, 3.0, 1.0, 19.0, 138.0, 107.0], [58.0, 145.0, 17.0, 65.0, 79.0, 103.0, 220.0, 234.0, 151.0, 242.0, 207.0, 206.0, 240.0, 180.0, 230.0, 115.0], [150.0, 17
2.0, 116.0, 34.0, 231.0, 173.0, 53.0, 133.0, 226.0, 249.0, 55.0, 232.0, 28.0, 117.0, 223.0, 110.0], [71.0, 241.0, 26.0, 113.0, 29.0, 41.0, 197.0, 137.0, 111.0, 183.0, 98.0, 14.0, 170.0, 24.0, 190.0, 27.0], [252.0, 86.0, 62.0, 75.0, 198.0, 210.0
, 121.0, 32.0, 154.0, 219.0, 192.0, 254.0, 120.0, 205.0, 90.0, 244.0], [31.0, 221.0, 168.0, 51.0, 136.0, 7.0, 199.0, 49.0, 177.0, 18.0, 16.0, 89.0, 39.0, 128.0, 236.0, 95.0], [96.0, 81.0, 127.0, 169.0, 25.0, 181.0, 74.0, 13.0, 45.0, 229.0, 122.
0, 159.0, 147.0, 201.0, 156.0, 239.0], [160.0, 224.0, 59.0, 77.0, 174.0, 42.0, 245.0, 176.0, 200.0, 235.0, 187.0, 60.0, 131.0, 83.0, 153.0, 97.0], [23.0, 43.0, 4.0, 126.0, 186.0, 119.0, 214.0, 38.0, 225.0, 105.0, 20.0, 99.0, 85.0, 33.0, 12.0, 1
25.0]]
```

Final comments:
If we now take the first value of this output, 82, and convert it to hex we get 52 and when comparing this to the correct inverse s box table it is the same number. Doing this with different digits also returns the correct hex value for that coordinate.

Later on I tested this with the sub_word function to prove undoubtedly that the table was correct. That code is below.

Version 2 - Functional Approach

Below I outline each function in my second attempt at coding AES and highlight previous errors in the design.

Function name: key_expansion	**Purpose:** To convert a given key to a series of

(Appendix Technical Solution AES Code Version 2 (72-92))	11 round keys.

Errors from version 1: Adding the first key to the key schedule is overcomplicated and confusing due to the orientation of the arrays in the diagrams and how they would need to be in code being different. Additionally I would not need to initialise a 3D array of 0s to begin with but I could instead just append each new round key to a list.

New errors: One problem I had during coding the key_expansion algorithm was adding the newly created key to the key schedule. I would create a temporary round key list and append a list containing the transformed column to it before appending the temporary round key list to the key_schedule.

However when appending lists of items to the round key list it would affect the other elements in the list.

After talking to a classmate and consulting stack overflow I learned that this is due to the way python handels lists. Although it may seem like a list just holds values *[x,x]* it instead holds pointers to the object which holds that value *[pointer to x, pointer to x]* which is why then when you modify x after appending multiple x's to the list the previous items in the list are modified with it.

Here is this effect demonstrated in python:

```python
a = []
b = [1]
a.append(b)
a.append(b)
b.append(5)
print(a)

>>Output [[1, 5], [1, 5]]
```

To resolve this we can append the values within the object rather than the object itself to the list.

Here is the fixed code in python:

```python
a = []
b = [1]
a.append(b[:])
a.append(b[:])
b.append(5)
print(a)

 >>Output [[1], [1]]
```

Here is the this problem manifesting in the specific part of the Key Expansion algorithm:

```python
for i in range(4):
      transformed_column = xor(transformed_column,
key_schedule[current_round][i])
      round_key.append(transformed_column)
    key_schedule.append(round_key)
```

Here I am appending a pointer to the object *transformed_column* meaning that in the next round of the for loop when *transformed_column* is altered the previous items in *round_key* will also be altered.

Here is my amended code:

```python
#line: 90
for i in range(4):
      transformed_column = xor(transformed_column,
key_schedule[current_round][i])
      #CHANGED SECTION
       xor_column = transformed_column[:]
       #END OF CHANGED SECTION
      round_key.append(xor_column)
    key_schedule.append(round_key)
#line: 98
```

Here I am appending the values within *transformed_column* to *round_key* rather than a pointer to the object itself.

Final Code:

```python
#line: 77
def key_expansion(key):
  # key = key.encode(encoding)
  key = bytes_to_matrix(key)
  # Adding first round key
  key_schedule.append(key)
  # key expansion 10 rounds for 128 bit key
  for current_round in range(10):
    round_key = []
    # taking last column and applying set of opperations to it
    final_column = key_schedule[current_round][3]
    transformed_column = round_constant(
      sub_word(rot_word(final_column), True), current_round)
    # using transformed_column to create next round key by xoring with
previous round keys
    for i in range(4):
      transformed_column = xor(
        transformed_column, key_schedule[current_round][i])
```

```
        # copies value in list rather than list itself
        xor_column = transformed_column[:]
        round_key.append(xor_column)
      key_schedule.append(round_key)
    print(f"key_schedule: {key_schedule} \n\n")
#line:99
```

Function name: sub_word **(Appendix Technical Solution AES Code Version 2 (50-60))**	**Purpose:** Takes each byte in the word and replaces it with its equivalent byte in the S_BOX table.
Explanation: This is one of the functions used in the key expansion, encryption, decryption	

Errors and analysis of version 1: (Appendix Technical Solution AES Code Version 1 (53-64))

```python
def sub_word(rot_column):
sub_word = []
for byte in rot_column:
 byte = str(byte)
 row_nibble = int(byte[:4], 2)
 column_nibble = int(byte[4:], 2)
 sub_value = str(bin(int(S_BOX[row_nibble][column_nibble])))
 sub_value = '{:0>8}'.format(sub_value.replace('0b', '')) #formatting sub
value
 sub_word.append(sub_value)
return sub_word
```

Although the function works as desired the code is clunky and confusing to read. I was trying to implement the verbal explanation of this function rather than think of an effective way to implement it computationally.

I realised after re-reading how the function works and what the s_box is I realised that you did not need to treat it as a 2D list with rows and columns but instead as a list. By taking the base 10 value of the bit you were substituting and counting that many spaces through the list you would get the same result.

Galois field 2^8

Function name: gf_mult **(Appendix Technical**	**Purpose:** To multiply two numbers in the GF(2^8)

Solution AES Code Version 2 (144-158))	

Explanation:

Used in the mix_columns function because:

1) Computers handle operations is GF(2^8) extremely well due to them being bitwise operations
2) The algorithm is less computationally expensive due to the use of bitwise operations
3) Due to the specific properties of GF(2^8) it provides resistances against specific attacks, the details of which are outside the scope of this NEA

Initial code, errors and solution:

```python
def gf_mult(a, b):
  # multiplication in gaussian feild 2^8
  if b == 1:
       #if b is one the result is a as a*1 = 1
    result = a
  else:
    result = 0
    for i in range(8):
       #loop through each bit in b, if it is one add to result
      if (b & 1): # if lsb = 1 xor with a
        result ^= a
      a_msb= a & 0x80
      if a_msb:
       #multiply here by irriducible polynomial
        a ^= 0x11b
      b >>= 1 #move to next bit in b
  return result
```

Upon testing this function to see if it worked I encountered a logic error.

Issue with this is each time we xor we just go back to the initial result.

After some research here is the issue I realised the key step I was missing was left shifting A for each iteration of the loop. Below I explain why this is necessary

Let's say the multiplication is $(x^6+x)(x^7+x^5+x^3+x+1)$

What we want the algorithm to do is
result += $(x)(x^7+x^5+x^3+x+1)$
result += $(x^6)(x^7+x^5+x^3+x+1)$

However in its current state it does this
result += $(x)(x^7+x^5+x^3+x+1)$
result += $(x)(x^7+x^5+x^3+x+1)$

This is because it does not recognise the second x as x^6 but instead x as it is not keeping track of how far through the bits of B it is.

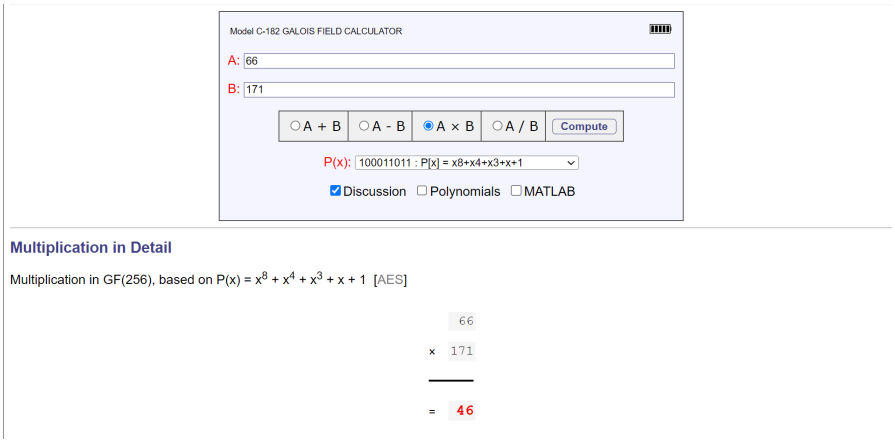
I could adjust it to keep track of what power of B it is and then multiplying by the correct power but that could get very complicated very quickly.

A more elegant approach is to left shift A upwards for each iteration of the loop so that now the algorithm does this:
result += (x)(x^7+x^5+x^3+x+1)
result += (x)(x^{13}+x^{11}+x^9+x^7+x^6)
which is the same as
result += (x)(x^7+x^5+x^3+x+1)
result += (x^6)(x^7+x^5+x^3+x+1)

Final Code:

```python
def gf_mult(a, b):
    # multiplication in gaussian feild 2^8
    if b == 1:
        # if b i one the result is a as a*1 = 1
        result = a
    else:
        result = 0
        for i in range(8):
            # loop through each bit in b, if it is one add to result
            if (b & 1):
                # if lsb = 1 xor with a
                result ^= a
            a_msb = a & 0x80
            a <<= 1  # shift a left to ensure it is multiplied by the
correct power
            if a_msb:
                # multiply here by irriducible polynomial
                a ^= 0x11b
            b >>= 1  # move to next bit in b
    return result
```

Expected output:



Output:

```
Python
print(gf_mult(0x42, 0xAB))

>>Output 46
```

Version 3 - OOP Approach

The final step was to turn my working AES code into an OOP base Approach. This allowed me to easily share functions between the encrypt and decrypt functions and avoid having to repeat code.

The biggest alteration I made here was how the data is inputted and outputted. This was due to how I was serialising my data allowing it to be sent over a network.

Appendix Location	**Category A**		**Objective Met**
(Appendix Technical Solution AES Version 3)	YES - Advanced matrix operations, complex mathematical model, Complex user-defined algorithms		9a
Subroutine/Class/module name	**Category A**	**Lines**	
KeyExpansion class			
key_expansion	Complex user-defined algorithms	45-66	
bytes_to_matrix			
rot_word			
sub_word			

round_constant			
xor			
SharedFunctions class, Inherits from KeyExpansion			
sub_bytes			
rotate			
gf_mult	YES - Complex mathematical model	113-127	
mix_columns	YES - Advanced matrix operations	129-149	
xor_key			
Encrypt class, Inherits from SharedFunctions			
encrypt	Complex user-defined algorithms	157-190	
padding			
encrypt_shift_rows			
Decrypt class, Inherits from SharedFunctions			
decrypt			
remove_padding			
decrypt_shift_rows			

Hybrid Encryption

Diffy Hellman

Although my initial idea was to use the Diffie Hellman Key Exchange Algorithm to share the symmetric key I learned that this was vulnerable to man in the middle attacks thanks to *GfG (2022)* and so had to change my approach.

The other type of encryption I knew about was Public Key Cryptography and I realised this was perfect for what I needed as not only did it remove the vulnerabilities Diffie Hellman but also allows me to use signature to make my app more secure

Public Key Cryptography

Public Key Cryptography uses two keys rather than one. One is the *public key* which is available to everyone and the other is the *private key* which is only known to the persons whose key it is. The two keys are mathematically linked meaning that when the *public key*
encrypts the plaintext; it can only be decrypted by the linked *private key*. This allows us to securely encrypt plaintext as if it is encrypted using someone's *public key* that means that only the person with the linked *private key* can read that message. (*Simply Explained, 2017*)

This provides **confidentiality** as no one else can read the data. However we can not guarantee who it came from (**authentication**) or that it hasn't been altered in transit (**integrity**) . That is what signatures are used for.

Signatures

Information in this section was learnt from (*Practical Networking, 2021*) and (*Practical Networking, 2021a*).

Encrypting with private key provides Authentication and Integrity but NOT confidentiality
Hashing provides Confidentiality

If we encrypt our message with our *private key* it means that only our *public key* can decrypt it. So when someone receives our message it tells them it must have come from us as only our *public key* can decrypt it. This provides **authentication** but not **confidentiality or integrity** as anyone could have decoded it, read it, and changed it along the way as everyone has our *public key*.

If we send the message encrypted with their *public key* and then the message encrypted with our *public key* this provides both **authentication and integrity** but not **confidentiality**. It now provides **integrity** because if the recipient decrypts both messages using the relevant keys and they are not the same we know some alteration must have occurred in transit but not **confidentiality** as we have still sent the message using our *public key* resulting in the same issue as the above paragraph.

If we send the message first hashed and then encrypted with our *public key* (the signature) and the message encrypted with their *public key* (the message) this provides all three.
Authentication because it was encrypted with our *public key*.
Confidentiality because it was encrypted using the recipient's *public key* and the message in the signature has first been hashed which means no one can read it as hashes are irreversible.
Integrity because the recipient can decrypt the message with their *private key*, hash it using the same hashing algorithm and then check it against the decrypted signature.

This is how we can send messages to ensure they are **confidential, authenticated and integrity protected.**

The Public Key Infrastructure (PKI)

The issue with using public key cryptography for every message is that it can’t be used for bulk data. The solution to this is to create a hybrid encryption using public key encryption and symmetric encryption.

By sending the symmetric key using public key cryptography this ensures that the symmetric key is known only to the sender and recipient and has not been intercepted using a man in the middle attack. Now both users can send messages using the symmetric key and symmetrical encryption as they have established a secure ‘channel’ over which they can send messages. Each time new contact is made this secure channel has to be established again using a new symmetric key. (*Practical Networking, 2021b*)

Pseudo code design of PKI

For sending the initial symmetrical encryption key the functions are the same but don't include the encrypt or decrypt data step highlighted in bold but instead just send or return the data itself. This is because the public key encryption of the symmetrical encryption key will be handled elsewhere.

```
Unset
def generate_signature(encrypted_data, private_key, public_key):
	valid = False
	signature_hash = hash(encrypted_data)
	signature = pk_encrypt(signature_hash, private_key)
	full signature {'signature' : 'signature', 'public_key' :
'public_key'}

def validate_signature(encrypted_data, signature):
	signature_hash = hash(encrypted_data)
	decrypted_signature = pk_decrypt(signature['signature'],
signature['public_key']
	if (decrypted_signature = signature_hash):
		valid = True
	return valid

def send_encrypted_data(data, symmetrical_encryption_key, private_key,
public_key):
	encrypted_data = s_encrypt(data, symmetrical_encryption_key)
	signature = generate_signature(encrypted_data, private_key,
public_key)
	send(encrypted_data)
	send(signature)

def recieve_encrypted_data(symmetrical_encrpytion_key):
	encrypted_data = recieve()
	signature = recieve()

	if validate_signature(encrypted_data , signature):
		data = s_decrypt(encrypted_data, symmetrical_encryption_key)
	else:
		handle_signature_error()
	return data
```

Client Server Model

Creating Protocols

Using the design I had previously created for the protocols I created them in python.

```
Python
def send(self, msg):
```

```
        message = msg.encode(FORMAT)
        self.client.send(self.packet_header(message))
        self.client.send(message)

    def packet_header(self, msg):
        msg_length = len(msg)
        send_length = str(msg_length).encode(FORMAT)
        send_length += b' ' * (HEADER - len(send_length))
        return send_length
```

I used these protocols throughout my project and in the final client server design.

Client - Failed Approach

(Appendix Technical Solution GUI OOP Approach Version 1 (35-122))

My initial attempts at a client was ok however its major flaw was how closely tied with the GUI it was. This meant it was not easy to test the client independently of the GUI and adding new features became difficult as the GUI code was so huge and messy. Despite its flaws the process of coding provided me with an understanding of many concepts that I used in the final design such as threading and JSON encoding.

Server - Failed Approach

(Appendix Technical Solution Server Version 1)

Handling Disconnect Errors attempt 1

My initial attempt to handle a client disconnect was naive and unsuccessful. I would put try and except blocks around every bit of code where the server was trying to receive data from the client and then call a handle function. I quickly stopped this approach as I realised it would be incredibly painful to implement as the server expanded.

Handling Disconnect Errors attempt 2

My second attempt was much more successful than the last. I had recently learnt about stacks in class and then discovered that exceptions in python bubble up the stack and realised I could use this to my advantage. (*Victoria Drake, 2023*)

Instead of having a try and except block inside every function I made I could have a try and accept statement around/inside each thread and however many subfunctions in the thread my code was the moment it returned a disconnect error it would bubble up the stack and be caught by my try and except statement.

Here are the two examples of the use of the try and except block to catch Disconnect Errors:

Catching a disconnect before the user has logged in and therefore before the handle_logged_in_client thread is called

```python
try:
    login = handel_inital_contact(new_user)
    if login:
        logged_in_clients.append(new_user)
        thread = threading.Thread(
            target=handel_logged_in_client, args=(new_user, ))
        thread.start()
        print(
            f"[ACTIVE LOGGED IN CONNECTIONS] {threading.active_count() - 1}
\n")
except ClientDisconnectException:
    print('[CLIENT DISCONNECT] Logged in FALSE')
```

```python
def handel_logged_in_client(user):  # runs for each new client
      handel_client_conn = sql.create_connection(database)

    #REMOVED CODE FOR READABILITY

      try:
            while connected:
            data = user.recive_data()

            #REMOVED CODE FOR READABILITY

      except ClientDisconnectException:
            print('[CLIENT DISCONNECT] Logged in TRUE')
            connected = False
      finally:
            return handel_client_conn
```

Client and Server Inheritance Approach

After coding version 2 of the server I decided to try and combine all the client and server main functionality into one class, because they shared so many functions, on which the separate client and server classes would inherit from. This would be the backbone of both the client and the server

Developing the Backbone

Appendix Location	Category A	Objectives

		Met
(Appendix, Technical Solution, NetworkingProtocols.py)	Complex client-server model parsing JSON/XML to service a complex client-server model A complex client-server model Complex user-defined algorithms	9a

Class/Method Name(given in order they appear in code)	**Explanation**	**Category A**
Base Class		
validate_signature	Validates a signature for parameter passed into seralized_data returning true or false depending on if it was valid or not	
generate_signature	Creates a signature for parameters passed into seralized_data. Returns signature along with public key to a dict with keys: signature, public_key	
add_packet_header	Returns bytes of exactly HEADER length with the first few bytes containing the number of bytes argument data is	
send_with_header	Sends a fixed sized packet containing the number of bytes in the next packet	
receive_data_with_header	Receives the header and handels relevant logic for receiving relevant data returning the json data	
send_encrypted_data	Sends data to self.client. Seralizes and encrypts it before sending	Complex user-defined algorithms (77-117)
recieve_encrypted_data	Receives encrypted data from self.client returning data + others depending on arguments	Complex user-defined algorithms (119-152)
forward_data	Sends data without signature or encryption	
send_data	Sends data and signature to self.client serialising it before sending	
receive_data	Receives data from self.client deserializing it before returning	
serialize_dict	Serialise a dictionary and encodes it in a JSON format then returning it encoded to uft-8	parsing JSON/XML to service a complex client-server model (195-215)

deserialize_dict	Turns serialised bytes into a dictionary also deserializing any objects or bytes too returning the completely deserialized dict	parsing JSON/XML to service a complex client-server model (217-226)
serialize_bytes	Returns bytes encoded to base64	
deserialize_bytes	Returns a string encoded in base64 into bytes	
serialize_object	Returns a serialised string of the object using pickle	
deserialize_object	Returns an python object of the serialised_object using pickle	
is_user_defined_class	Checks if the object is an in-built python object or not. Taken from (https://stackoverflow.com/questions/14612865/how-to-check-if-object-is-instance-of-new-style-user-defined-class)	

Data Serialization

When trying to first implement encryption I created a stripped back version of the client and server code without the GUI. This enabled me to get rid of all the clutter and understand any problems with the code I was having.

Many of these problems resulted from the data not being serialised properly and as such I had to create new data serialisation functions that allowed me to send what I needed to send.

All of the following was figured out in the code editor in comments and I am retelling that process here.

Firstly I made a list of the data types I needed to send by using the python type() function.
These were

- Strings
- Bytes
- Dictionaries
- User defined Objects
- Lists

One complicating factor was that I would be sending a dictionary with one or more of any of these data types (excluding dictionaries) but did not know where they would be in the dictionary.

Next I tried different methods of serialising the different data types with the success criteria being if they could be encoded into JSON.

JsonPickle:
Initially I tried using the jsonpickle module for both bytes and objects which worked however to then deserialize the object at the other end caused an issue. Due to the way jsonpickle works It needs an instance of the object it's deserializing to be created so it can effectively ‘overwrite’ it. While this could work for functions I created myself, for sending rsa keys It would not work. My knowledge of the rsa module was not good enough to fully understand the error I was encountering so I tried a different approach. (*jsonpickle Documentation — jsonpickle 1.4.3.dev0+g5b8d3ea.d20201130 documentation*, no date)

Pickle and base64:
This second approach was successful.

For the bytes I would turn them into base64 and then a string. By turning them into base 64 it ensured that no data was lost as supposed to if I turned it into a string. For example if the data unintentionally contained some unicode character that could throw an error if I just converted it into a string It would not work.

```python
def serialize_bytes(self, data: bytes) -> str:
        """Turns bytes into a base64 encoded string"""
        b64 = base64.b64encode(data)
        return b64.decode(FORMAT)
```

For the objects I had to first serialise them using the python pickle module and then convert the resultant bytes into a string using the above method.

```python
def serialize_object(self, obj: object) -> str:
        """Turns an object into a seralized string"""
        pickled_object = pickle.dumps(obj)
        string_of_object = self.serialize_bytes(pickled_object)
        return string_of_object
```

Then for the decoding I just had to do everything in reverse.

Serialising dictionaries was more difficult as I firstly needed to know if each value of a key value pair was a bytes object or an object and then find a way to keep track of the type of changes I made and where I made them

For knowing the data type for a bytes object I could simply use the python function isinstance(object, bytes) however for objects this was trickier.

Since almost everything is an object in python if I used the same isinstance() function to check if it was an object It would return true for things like strings or lists which I didn't want. The solution to this was to check if the object had the attribute '__dict__' or '__slots__' which most user defined python objects do. While this may not work for all user defined classes for my purposes it was good enough. (*How to check if object is instance of new-style user-defined class?*, no date)

```python
def is_user_defined_class(self, obj: object) -> bool:
```

```
        """
        Checks if object is an in-built python object or not
        """
        if hasattr(obj, '__class__'):
                return (hasattr(obj, '__dict__') or hasattr(obj, '__slots__'))
```

For keeping track of the changes I realised that I could append the key of the value I made a change to to a list which would hold all of the changes made in the dictionary for a specific type. For example obj_mapping would hold all of the keys for the objects in the dictionary I had to serialise. Then for deserializing it was a simple matter of checking if the item was in a specific __mapping list and applying the relevant functions to it.

Python

```
def serialize_dict(self, data: dict) -> bytes:
        """Seralizes a dictionary and encodes it in a JSON format"""
        obj_mapping = [] # contains the keys in the dict where the value was an
object
        bytes_mapping = [] # contains the keys in the dict where the value was
bytes
        for key, value in data.items():
                if self.is_user_defined_class(value):
                        data[key] = self.serialize_object(value)
                        obj_mapping.append(key)
                elif isinstance(value, bytes):
                        data[key] = self.serialize_bytes(value)
                        bytes_mapping.append(key)

        if 'type' not in data:
                data['type'] = 'None'

        data['obj_mapping'] = obj_mapping
        data['bytes_mapping'] = bytes_mapping
        data = json.dumps(data)
        data = data.encode(FORMAT)
        return data
```

Implementing PKI Infrastructure

Now that I had fixed the issue of data serialisation I could begin to properly implement sending encrypted messages.

Having already written the AES algorithm and Designed the public key infrastructure for sending those messages, writing the actual code was fairly simple. One key change I made since designing the

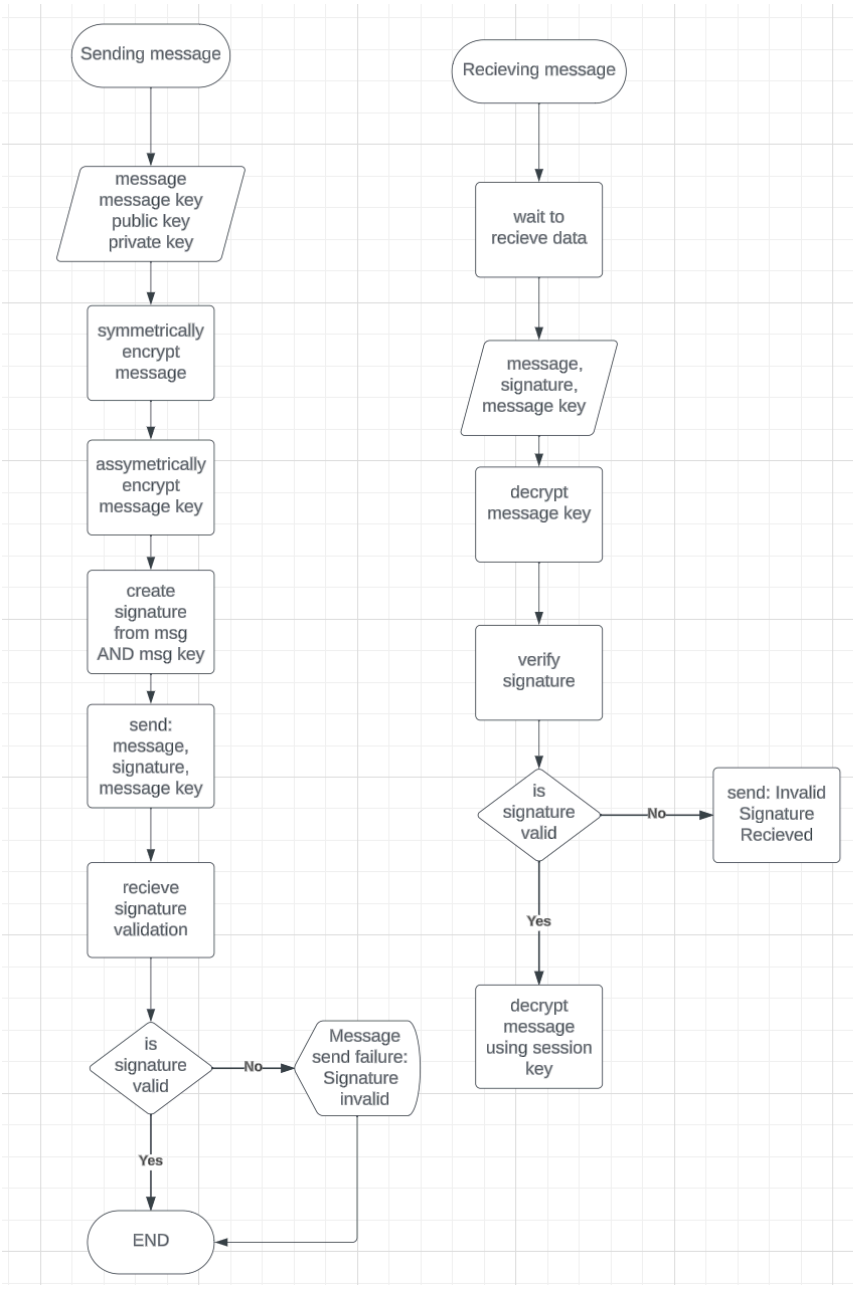
public key infrastructure was that for each message I would use an Ephemeral Key (Epk) to symmetrically encrypt each message rather than a static key. An Epk is a key that is unique to each new sent message. This means that even if an attacker got hold of an Epk they would only be able to decrypt the messages that Epk was relevant to rather than if they got hold of a static key where they would then be able to decrypt all messages. (*Computerphile, 2018)*

Before choosing to use an Epk I considered other ideas such as a rotating key however that would cause the code to become too complex and it would be less secure than if I just used an Epk.

My decision to use an Epk was also largely based on the fact that Signal Instant Messaging Protocol, which WhatsApp uses, also uses it to make sending encrypted data more secure. (*Computerphile, 2018)*

The biggest issue with the Epk is that of sending it securely along with the message. To solve this you can just asymmetrically encrypt the Epk with the recipient's public key, being sure to save the plaintext Epk before for storage purposes.

With all of this in mind I designed a new PKI with data serialisation and Epks included.
Firstly I created two flowcharts to get an understanding of the order of operations then I wrote the pseudo code.



Flowchart of PKI

Unset

```
def send_encrypted_data(data, Epk, private_key, public_key,
recipient_public_key):
      seralized_data = serialize_dict(data)
      encrypted_data = symmetrically_encrypt(seralized_data, Epk)
```

```
       encrypted_Epk = assymmetrically_encrypt(Epk, recipient_public_key)

       lump_data = {'encrypted_data': encrypted_data,
                    'encrypted_Epk': encrypted_Epk}

        #May need to re-seralize new dict depending on datatypes of
encrypted values

        signature = generate_signature(
            seralized_lump_data, private_key, public_key)

        seralized_signature = serialize_dict(signature)

        send(seralized_lump_data)
        send(seralized_signature)

def recieve_encrypted_data(private_key):
       lump_data = recieve()
       signature = recieve()

       #Deseralize data and signature here if seralized in the encryption
process

       if validate_signature(encrypted_data , signature):
              encrypted_Epk = lump_data ['encrypted_Epk']
              Epk = assymmetrically_decrypt(encrypted_Epk, private_key)
              seralized_data = s_decrypt(lump_data ['encrypted_data'],
symmetrical_encryption_key)
              data = deseralize_dict(seralized_data)
       else:
              handle_signature_error()
       return data
```

I then implemented this fully in python in a file called NetworkingProtocols. During implementation and testing all of the errors were either syntax errors or performing operations in the wrong order such as deserializing an encrypted message instead of decrypting it first due to misreading the pseudocode and flowchart. As the entire project developed some methods were minorly altered.

Developing the server

Appendix Location	**Category A**	**Objectives Met**
(Appendix, Technical Solution, Server.py)	Complex user-defined algorithms: message queues Dynamic generation of objects based on complex user-defined use of OOP model Complex user-defined use of object-oriented programming (OOP) model, eg classes, inheritance, composition, polymorphism, interfaces Complex client-server model	1, 2, 3, 6, 7, 8, 9, 12

Class/Method Name(given in order they appear in code)	**Explanation**	**Category A**	**Objective Met**
Class UserHandler		Complex user-defined use of object-oriented programming (OOP) model (34-300)	
get_name	Returns ip, port and client userID		
handel_disconnect	Removes self from list of active clients and closes connection		
Module SENDING AND RECEIVING DATA			
send_data_to_client	Sends data to client autofilling public and private key arguments		
send_encrypted_data_to_client	Sends encrypted data to client autofilling necessary arguments		9a
recieve_encrypted_data_from_client	Returns the received decrypted data from client autofilling Private Key argument		
Module HANDLE INITIAL CONTACT FUNCTIONS			
handel	Runs the initial functions to handle the first contact between a client and a server		
handel_inital_contact	Just call swap_public_keys. Here if I ever had to expand the initial contact.		
swap_public_keys	Swaps public keys with the client setting self.client_public_key in the process		
Module HANDEL LOGGED OUT CLIENT FUNCTIONS			1,2
handle_logged_out_client	Handels logged out client by waiting to	Server-side scripting	

	receive a determiner specifying what the user is trying to do and calling functions appropriately	using request and response objects and server-side extensions for a complex client-server model (100-112)	
handel_login	Handels clients login attempt		2
handel_create_account	Handels clients create account attempt		1
is_user_already_online	Checks if user is already in active client list		
store_user_login_details	Adds users id, screen name, hashed password, password salt and public key to the servers database		
hash_password	Hashes and salts peppers password as per OWASP guidelines 2024		g (178-184)
validate_login_info	Checks received password against stored hash and returns relevant bool		2
get_pepper	Gets stored pepper from Windows Key Store		giii
Module HANDEL LOGGED IN CLIENT FUNCTIONS			
handle_logged_in_client	Swaps public keys, sends screen_name to client and then calls recieve_data_from_logged_in_user		
recieve_data_from_logged_in_user	Received encrypted data from logged in user and handels it accordingly	Server-side scripting using request and response objects Complex user-defined algorithms (214-272)	9, 6, 3, 7, 8, 12
forward_data_to_client	Attempts to forward the data to the intended recipient. If it can't adds to the message queue instead	Queues (274-293)	9
recieved_message_queue	Sends messages waiting in message_queue to relevant client	Queues (295-300)	9
class MessageQueue			9
isEmpty	returns true/false if queue is empty		
enQueue	adds user_id and message to message queue		
deQeueu	removes user_id and relevant message from message queue		

get	returns message queue		
Module DRIVER CODE			
worker	calls handel function for each new user received	Dynamic generation of objects based on complex user-defined use of OOP model (324-330)	
main	starts the server and waits to receive connections calling the worker function in a thread each time it does.		

When creating the server I took key concepts I had learnt from my first version of the server such as threading, and error handling while also improving it by moving the entire server functionality to an OOP approach.

Handling Connections and Disconnections

Functions: main, worker	Purpose: Start the server, handel client connections and disconnections

Explanation:

To handle connections the server creates an instance of the UserHandler class in a separate thread. This fixes the issue of no more than one logged-out user at a time being able to join. It also allowed the server to handle disconnect errors.

By creating an instance of the UserHandler class in a separate thread when an error occurs it will bubble up the stack in that thread and then be handled correctly rather than bubbling up inside or outside the thread depending on whether the user was logged in or not therefore needing two try except statements.

```python
def worker(client, addr):
  # allows disconnect excpetion to be handeled within each new users thread rather than having to leave the thread
  new_user = UserHandler('SERVER', client, addr)
  try:
    new_user.handel()
  except ClientDisconnectException:
    new_user.handel_disconnect()

# ------DRIVER CODE------

def main():
```

```
  # create new socket object
  server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  try:
    # bind socket to port and IP
    server.bind(ADDR)
  except:
    print(f"Unable to bind to server {SERVER} and port {PORT}")
  server.listen()
  print(f"[LISTENING] Server is listening on {SERVER} {PORT}")
  while True:
    client, addr = server.accept() # waits till new connection
    thread = threading.Thread(target=worker, args=(client, addr))
    thread.start()


message_queue = MessageQueue()
main()
```

Message Queue

Module Name: Message Queue **(Appendix Technical Solution (274-300)**	**Purpose:** Handel everything to do with the message queue

Explanation:
The message queue is an important aspect of the app's functionality and so was important to get right. It took a few attempts to create a working message queue.

Initially I used a list as seen in the code below however something wasn't working and it became too difficult to keep track of how the list was being manipulated at each stage.

Python

```
def forward_data_to_client(self, recieved_data):
    print('forwarding data to client')
    seralized_lump_data = recieved_data[1]
    seralized_signature = recieved_data[2]
    recipient_user_id = self.deserialize_dict(seralized_lump_data)[
      'recipient_user_id']
    print(f"{recipient_user_id = }")
    for c in active_clients:
      print(f"{c.client_user_id = }")
      if c.client_user_id == recipient_user_id and c.can_recieve_msg == True:
```

```
        c.forward_data(seralized_lump_data, seralized_signature)
        if len(self.outoging_message_queue) != 0:
          self.outoging_message_queue.pop(0)
      elif c.client_user_id == recipient_user_id and c.can_recieve_msg ==
False:
        print("MESSAGE ADDED TO QUEUE")
        self.outoging_message_queue.append(recieved_data)#add to front of
queue not back

  def recieved_message_queue(self):
    print("RECIEVING MSG QUEUE HOPEFULLY")
    for d in active_clients:
      print(d)
      for message in d.outoging_message_queue:
        print(message)
        seralized_lump_data = message[1]
        seralized_signature = message[2]
        recipient_user_id = self.deserialize_dict(seralized_lump_data)[
          'recipient_user_id']
        if self.client_user_id == recipient_user_id:
          print("FORWARDING DATA")
          d.forward_data_to_client(
            message)
```

Instead I created a messageQueue Class and implemented relevant functions as methods within that class.

Python

```
class MessageQueue():
    def __init__(self):
        self.__items = []

    def isEmpty(self):
        return len(self.__items) == 0

    def enQueue(self, client_recieving_user_id: str, message: list):
        values = (client_recieving_user_id, message)
        self.__items.append(values)

    def deQeueu(self, client_recieving_user_id: str, message: list):
        if not self.isEmpty():
            values = (client_recieving_user_id, message)
            try:
                self.__items.remove(values)
            except Exception as e:
```

```
                print(f"[deQueue ERROR] {e}")

    def get(self):
        return self.__items
```

This worked however there was still a problem which after simplifying the problem as demonstrated in the code below I figured out.

Python

```
def recieved_message_queue():
  for message in message_queue.get():
    print(f"Looking at {message}")
    message_queue.deQeueu(message)


class MessageQueue():
  def __init__(self):
    self.__items = []

  def isEmpty(self):
    return len(self.__items) == 0

  def enQueue(self, item):
    self.__items.append(item)

  def deQeueu(self, item):
    if not self.isEmpty():
      print(f'DEQUEUE {item}')
      self.__items.remove(item)

  def get(self):
    return self.__items


message_queue = MessageQueue()
message_queue.enQueue('a')
message_queue.enQueue('b')
message_queue.enQueue('c')

print(message_queue.get())
recieved_message_queue()

#Returns
>>>
['a', 'b', 'c']
Looking at a
```

```
DEQUEUE a
Looking at c
DEQUEUE c
```

Here I realise the problem was that each time I dequed a message I was using an updated version of the queue since the last dequeue meaning that b gets skipped over.

1) deQueue() -> list [b,c] Item removed from index 0. Looking at index 1
2) deQueue() -> list [b] Item removed from index 1. Looking at index 2

B is skipped as after removing a, b moves to index 0 and the program looks at index 1 of the list seeing c and therefore removing it.

To fix this error I simply took a copy of the message queue when calling the recieve_message_queue function.

Developing the client

Appendix Location	**Category A**	**Objectives Met**
(Appendix, Technical Solution, Client.py)	Complex client-server model Dynamic generation of objects based on complex user-defined use of OOP model Complex user-defined use of object-oriented programming (OOP) model, eg classes, inheritance, composition, polymorphism, interfaces	1, 2, 3, 5, 6, 7, 8, 9, 12

Class/Method Name(given in order they appear in code)	**Explanation**	**Objective Met**
class Client		
connect	Attempts to connect to the server specified in the ADDR variable.	
establish_inital_contact	establishes initial contact with the server by generating public and private keys and swapping them	
send_disconnect_message	sends disconnect message to the server	
close_client	closes client	
connect_to_database	Creates and or connects to database in self.user_path	
close_db_connection	closes database connection	
send_data_to_server	Sends data to server autofilling public and private key parameter	
send_encrypted_data_to_server	Creates Epk and sends encrypted data to server	
recieve_encrypted_data_from_server	returns decrypted received data from server autofilling private key parameter	
send_encrypted_data_to_recipient	Creates Epk, gets recipients private key and sends data to server to forward to recipient	9a
Create/Login/Delete Account methods		
login	Loggs user in by confirming details with the server. If it was successful it connects to the users database, gets the public and private keys and then swaps them with the server	
create_account		1

delete_account	Sends account deletion notification to server and all friends Returns if it was successful or not	6a
delete_directory	Deletes everything in the users account directory and then the directory itself	6
Public and Private Key method		
generate_init_keys	Generates the clients public, private and master keys	
write_keys_to_file	Writes clients public, private and master keys to local file in self.user_path	
get_keys_from_file	Retrieves clients public, private and master keys from local file in self.user_path	
swap_public_keys	sends public key to server and receives servers public key	
Getting friends lists methods		
get_friend_list	Sets self.friend_list to equal user's friend list	
get_friend_request_list	Sets self.friend_request_list = list of incoming friend requests	5
get_pending_friends_list	Sets self.pending_friend_list = list of outgoing friend requests	4
Listen for messages methods		9
handel_logged_in_client	Gets screen_name from server	
listen	Send message to server saying it can listen and starts the listening thread	
stop_listen	Send message to server saying it can't listen and stops the listening thread	
recieving_data_from_client		
handel_send_message	Sends message to server and then stores it	
handel_recieved_message	Handels received message accordingly if it is text or an image	
decrypt_message	Returns tuple (decrypted_message, date, time, from_me, is_image)	
store_sent_message	Stores sent message encrypting the Epk with self.master_key	
store_recieved_message	Stores recieved message encrypting the Epk with self.master_key	
get_message_history		
decrypt_message_history		
Add Friend Page methods		3
block_friend		3a
unblock_friend		3b

accept_friend_request		
reject_friend_request		
send_friend_request		
store_friend_request	Gets friend details from server for an accepted friend request and stores them	3
check_if_user_is_already_friends		
check_if_friend_code_exists		
Sending image methods		9
get_image_data	Returns binary data of image at image_path	
store_sent_image_to_files	Gets sent image sent image path and then stores it Returns stored image path	9b
store_image_to_files	Stores image to self.user_images_path	9b
find_sutable_image_path	Finds suitable image name by adding (i) if there are duplicates Returns full image path	
compress_image	Compresses an image to: - max 100px,100px - LANCZOS resampling - save quality: 65% Returns new image path	
client_get_image_path	Displays popup asking user to select an image file Returns selected image path	
change_screen_name	Notifies the server of screen name change	8
request_all_user_data	Returns all data server has on user	7
get_output_path	Returns chosen path to save user data	7

Threads

Module Name: Handel receiving messages	**Purpose:** To stop and start a thread that
Development and Explanation: This was the biggest problem I had while implementing a functioning client. When the client logs in it needs to be constantly listening for messages but still able to separately send and receive data through other functions. While one solution would be to create more if statements in the recieving_data_from_client function that would make it difficult to read the code and could quickly make it too complex to add new features.	

After experimenting with threads and failing I realised that the key problem hindering me from finding a solution was not actually threads but the self.recieve_encrypted_data function.

What I realised is that each time the function is called it waits at that point in the code indefinitely until it receives data from the server and therefore does not progress to the next line of code. All of my potential solutions relied on checking for some sort of condition at the start of the while loop and so when it wanted to receive data it would never reach the start of the while loop again to check the condition.

The solution turned out to be setting a timeout for receiving data from the server which would then cause an error to occur if the client did not receive any data in time sending allowing for the end of the while loop to be reached and the condition to be checked again.

After I figured this out the rest of the solution was easy and just involved stopping the while loop from executing and then waiting for the thread to complete and close.

Python

```python
def listen(self):
    print('Listen')
    self.stop_event.clear()
    self.listen_thread = threading.Thread(
      target=self.recieving_data_from_client)
    self.listen_thread.start()

  def stop_listen(self):
    try:
      self.stop_event.set()
      self.listen_thread.join()
      print('STOP Listen')
    except:
      pass

  def recieving_data_from_client(self):

    while not self.stop_event.is_set():
      try:
        self.client.settimeout(1)
        recieved_data = self.recieve_encrypted_data(self.private_key)
       #code remove for readability
      except socket.timeout:
        continue
    print("thread_stopped")
```

Message Compression

Analysis

When trying to send encrypted images I found that the UI was freezing for a long time and sometimes crashing if I tried to do anything with it. After tracing through the code I found out the reason was that my encryption algorithm was taking so long to encrypt the messages.
From here I had two options, compress images or optimise the algorithm. Image compression was the simplest and fastest option so I went with that one.

Technical Solution

Function name: compress_image **(Appendix Technical Solution Client.py (566-583))**	**Purpose:** To compress an image to a suitable size such that the encryption algorithm could compress it within a few seconds
Explanation: For my implementation I decided to just load the image, make its dimensions smaller which would in turn reduce the file size. I would then save it at a downgraded quality to further decrease the file size.	

```python
def compress_image(self, image_path: str):
    """
        Compresses an image to:
            - max 100px,100px
            - LANCZOS resampling
            - save quality: 65%

        Returns new image path
           """
    old_path, extension = os.path.splitext(image_path)
    file_name = old_path.split('/')[-1]
    split_path = image_path.split('/')[:-1]
    new_path = 
f"{'/'.join(split_path)}/{file_name}(downsized_for_encryption){extension}"

    image = Image.open(image_path)
    image.thumbnail((100, 100), Image.Resampling.LANCZOS)
    image.save(new_path, optimize=True, quality=65)
    return new_path
```

Testing

I chose a series of large images and compressed them and then ran them through my encryption algorithm to see if it took an acceptable amount of time.

Handling Disconnect Errors Attempt 3

Getting the server to gracefully handle disconnect errors was a combination of two factors.
Firstly, creating an instance of the UserHandler class in a separate thread meant that when an error occurred it would bubble up the stack in that thread only and then be handled gracefully rather than bubbling up inside or outside the thread depending on whether the user was logged in or not therefore needing two try except statements. The code below shows my implemented solution of this.

(Appendix Technical Solution Server.py (324-end)

```python
def worker(client, addr):
  # allows disconnect excpetion to be handeled within each new users thread rather than having to leave the thread
  new_user = UserHandler('SERVER', client, addr)
  try:
    new_user.handel()
  except ClientDisconnectException:
    new_user.handel_disconnect()


# ------DRIVER CODE------

def main():
    #code removed for readability
    while True:
    client, addr = server.accept() # waits till new connection
    thread = threading.Thread(target=worker, args=(client, addr))
    thread.start()


message_queue = MessageQueue()
main()
```

Secondly, having the client stop its own listening thread when trying to disconnect prevented any errors occurring client side as it wasnt able to receive any messages from other users that could cause it to run different functions. Below is the implemented code with lines sections highlighted in bold.
(Appendix Technical Solution GUI OOP Approach Version 2 (75-87))
(Appendix Technical Solution Client.py (88-94))

```python
#GUI CODE
def on_close(self, account_deletion=False):
        """Function called when the user closes the entire app"""
        try:
            self.client.stop_listen()
            self.client.send_disconnect_message()
            # self.client.send_data({'type': 'DISCONNECT'})
            print('[-] CLOSING APP...')
```

```
        except Exception as e:
            print(e)
        self.destroy()
        self.client.close_db_connection()
        if account_deletion:
            self.client.delete_directory()
        self.client.close_client()
#CLIENT CODE
def send_disconnect_message(self):
        print("[SENDING DISCONNECT MESSAGE]")
        if self.logged_in:
            self.send_encrypted_data_to_server({'type': 'DISCONNECT'})
        else:
            self.send_data_to_server({'type': 'DISCONNECT'})
        self.alive = False
        print("[DISCONNECTED FROM SERVER]")
```

GUI

OOP Approach version 1

(Appendix Technical Solution GUI OOP Approach Version 1 (124-end))

Video demonstrating functionality of this initial version:
https://youtu.be/4DVpa3SCzoc

This version worked in conjunction with the second version of the server

Ultimately this approach failed for a few reasons:

- The functionality of the client was not fully fleshed out and so connecting it with the GUI was difficult
- The GUI and Client classes communicated ineffectively with each other as I was not using a strictly inheritance approach.
- Having the client code in the same file as the GUI causes the code to become cluttered and confusing quickly due to the amount of GUI code.

OOP Approach version 2

(Appendix Technical Solution GUI OOP Approach Version 2)

The main reason why this approach worked was that by the time I started building the GUI the majority of the client code was complete and so it was easy to inherit from the class in the client code and use the functions without trying to figure out if the reason they were not working was to do with the GUI or the client code itself.

The GUI implements functionality to meet all objectives.

SQL

Server Side Storage Approach

For full code see **(Appendix, Technical Solution, Server Side SQL Approach Code)**

Ultimately I scrapped this approach for reasons listed in the Client side storage approach however it is worth explaining certain complex SQL functions.

SQL Code

SQL	Explanation
Unset `SELECT` `  c.conversation_id,` `  CASE` `  WHEN c.participant_1 = ? THEN u2.user_id` `  WHEN c.participant_2 = ? THEN u1.user_id` `  END AS other_user_id,` `  CASE` `  WHEN c.participant_1 = ? THEN u2.screen_name` `  WHEN c.participant_2 = ? THEN u1.screen_name` `  END AS other_screen_name` `FROM conversations c` `JOIN users u1 ON c.participant_1 = u1.user_id` `JOIN users u2 ON c.participant_2 = u2.user_id` `WHERE ? IN (c.participant_1, c.participant_2);`	This statement took a few attempts to get right. For one of the failed attempts see **(Appendix, Technical Solution, Server Side SQL Approach Code, Lines(79-89))** I needed to get the conversation_ids, user_ids and screen_names for all conversations the user was a part of. Essentially getting their friend list. Working backwards the WHERE statement is trying to find rows where the user's user_id is either participant_1 or participant_2 of the conversation. The two JOIN statements join the conversations table and users table where the user_id is either participant_1 or participant_2. I needed two join statements as the SQL does not know if the recipient is participant_1 or participant_2. The JOIN statements then allow me to get the screen_name and user_id of the other user in the conversation by using two CASE statements in a similar fashion to how it is used when getting the recipients_id in the below SQL statement.
Unset `SELECT` `  CASE` `  WHEN participant_1 = ? THEN participant_2` `  WHEN participant_2 = ? THEN participant_1` `  END AS other_user_id` `FROM conversations`	This was my introduction to the CASE statement in SQL. I needed to get the recipient's user id when a user was sending a message. However due to how the database is structured there is no definition between the sender and received in a conversation as the conversation goes two ways. After some research I found out about the CASE statement

`` WHERE conversation_id = ?; ```	and put it into use. The CASE statement essentially acts as an if statement in the SQL code. When participant_1 of the conversation was equal to the current user's user id, return participant_2 under the name other_user_id. The other WHEN statement does the opposite.
`` Unset SELECT m.message_id, m.conversation_id, m.message_text, m.date, m.time, m.sender_id, users.screen_name FROM messages m JOIN users ON m.sender_id = users.user_id WHERE m.conversation_id = ? ORDER BY m.message_id ASC; ```	Here I needed to get the message history for a user where it was relevant to a specific conversation id. The join is necessary as I also needed the sending user's screen_name for the display of the message.

Server and Client side storage approach

For full code see **(Appendix, Technical Solution, Server and Client Side SQL Approach Code)**

Analysis

While implementing encryption I realised that if I stored all user data server side it would be difficult to implement and also not as secure as it could be. Taking inspiration from whatsapp and how they implement security I have decided to switch to storing all client relevant data client side.

Pros of switch:

- Puts the server under less pressure as it is not constantly having to retrieve and store data.
- Makes implementing encryption easier as it allows Epks to be stored and encrypted locally meaning there was no need find a workaround of storing Epks securely server side
- Makes the security of the app better as it means Epks don't need to be encrypted and encrypted server side.
- Makes security of the app better as now all the users messages are stored client side so if there is a database leak server side their encrypted messages are not leaked.

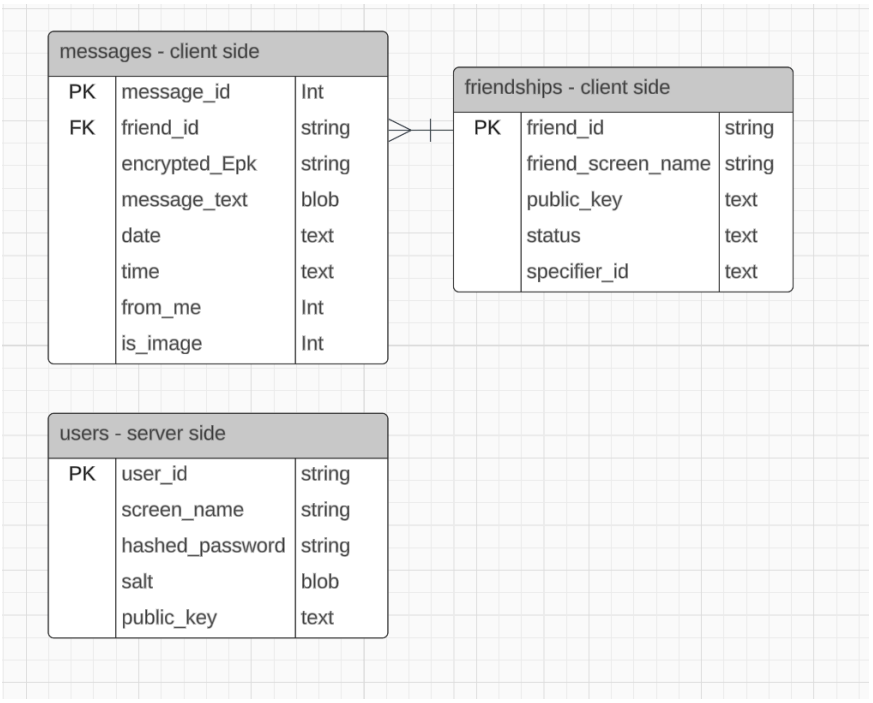
Cons of switch:

- Each time a user creates a new account I will need to create a specific directory on their computer to store their database and anything else such as message images or private keys.

After analysing the pros and cons of switching I informed my Client about the switch and the reasons behind it explaining the pros and cons. He agreed to make the switch to client side storage.

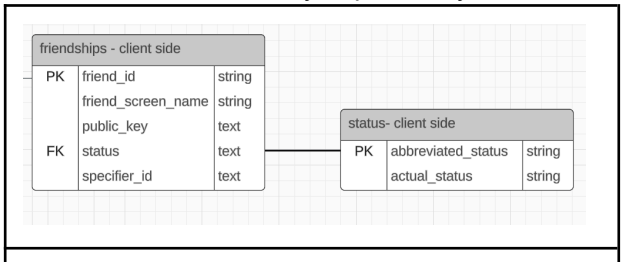
Design

The important thing to understand when designing this database was that there was no need to store any sort of conversation_id as because everything is stored user side and a conversation can only have two participants it would be guaranteed to be between the user and the friend.



Normalised entity relationship diagrams

One key thing to point out with this diagram is that ‘status’ in the friendships table can take one of 3 values: ‘blk’, ‘acc’ or ‘req’. Each of these is an abbreviation for the real value. If I were storing the abbreviation of the real values and then using the real values, ‘blocked’, ‘accepted’ or ‘requested’ I would need another table to remove the non-key dependency and maintain a fully normalised table.



Maintaining a normalised table

However as I am not using the real values and just the abbreviated ones in my code I do not need to do this.

Server Side SQL Functions	
SQL	**Explanation**
Unset `CREATE TABLE IF NOT EXISTS users(` `user_id text NOT NULL PRIMARY KEY,` `screen_name text NOT NULL,` `hashed_password text NOT NULL,` `salt blob NOT NULL,` `public_key text NOT NULL,` `UNIQUE(user_id)` `);`	Creates users table
Unset `INSERT INTO users(` `user_id,` `screen_name,` `hashed_password,` `salt,` `public_key)` `VALUES(?,?,?,?,?)`	Adds a new user to the database
Unset `SELECT COUNT(*)` `FROM users` `WHERE user_id = ?`	Counts number of users with specified user id. Can be used when checking if user_id is already used when creating an account or if a user exists when a user is trying to friend another user
Unset `SELECT user_id, screen_name, public_key` `FROM users` `WHERE user_id=?`	Gets user_id, screen_name and public_key
	Gets the screen_name of the specified user_id

Unset `SELECT screen_name` `FROM users` `WHERE user_id=?`	
Unset `SELECT public_key` `FROM users` `WHERE user_id=?`	Gets the public key of the specified user_id
Unset `SELECT hashed_password, salt` `FROM users` `WHERE user_id=?`	Gets hashed password of the specified user_id
Unset `UPDATE users` `SET screen_name = ?` `WHERE user_id = ?;`	Updates a users screen name
Unset `UPDATE users` `SET user_id = ?, screen_name = ?` `WHERE user_id = ?;`	Used when a user deletes their account and both the screen name and user_id needs to be changed

Client Side SQL Functions	
SQL	**Explanation**
Unset `CREATE TABLE IF NOT EXISTS messages (`	Creates messages table

`message_id INTEGER PRIMARY KEY,` `friend_id text NOT NULL,` `encrypted_Epk blob NOT NULL,` `message_text blob NOT NULL,` `date text NOT NULL,` `time text NOT NULL,` `from_me integer NOT NULL,` `is_image integer NOT NULL,` `FOREIGN KEY (friend_id) REFERENCES friendships(friend_id)` `);`	
Unset `CREATE TABLE IF NOT EXISTS friendships (` `friend_id text NOT NULL UNIQUE,` `friend_screen_name text NOT NULL,` `public_key text NOT NULL,` `status text NOT NULL,` `specifier_id text NOT NULL,` `PRIMARY KEY (friend_id)` `);`	Creates friendships table
Unset `UPDATE friendships` `SET status = 'blk', specifier_id = ?` `WHERE friend_id = ?;`	Blocks a friend where the user has blocked a friend or a friend has blocked the user.
Unset `UPDATE friendships` `SET status = 'acc', specifier_id = ?` `WHERE friend_id = ?;`	Unblocks a friend where the user has unblocked a friend or a friend has unblocked the user.
	Gets the friend list of a user

Unset `SELECT friend_id, public_key, friend_screen_name, status, specifier_id FROM friendships WHERE status = 'acc' or status = 'blk';`	
Unset `SELECT friend_id FROM friendships WHERE status = 'acc';`	Gets all users accepted friends.
Unset `SELECT friend_id, public_key FROM friendships WHERE status = 'req' and specifier_id != ?;`	Gets all users incoming friend requests. To be used when showing incoming friend request list
Unset `SELECT friend_id FROM friendships WHERE status = 'req' and specifier_id = ?;`	Gets all users pending friend requests. To be used when showing pending friend request list
Unset `INSERT INTO friendships ( friend_id, friend_screen_name, public_key, status, specifier_id);`	Used when adding a newly received friend request
Unset `SELECT COUNT(*) FROM friendships`	Checking if a user is already friends with another user. Used when trying to send a friend request.

`` WHERE friend_id = ?; ```	
`` Unset UPDATE friendships SET status = 'acc', specifier_id = ? WHERE friend_id = ?; ```	Accepting a friend request
`` Unset DELETE FROM friendships WHERE friend_id = ?; ```	Rejecting a friend request or receiving a rejected friend request
`` Unset SELECT encrypted_Epk, message_text, date, time, from_me, is_image FROM messages WHERE friend_id = ? ORDER BY message_id ASC; ```	Used when getting all of a user's messages to, and received from, a specific friend.
`` Unset INSERT INTO messages(friend_id, encrypted_Epk, message_text, date, time, from_me, is_image); ```	Stores a sent or received message
`` Unset UPDATE friendships ```	Updating a friends screen_name if they have changed it

`` SET friend_screen_name = ? WHERE friend_id = ?; ```	
Unset ``` UPDATE friendships SET friend_screen_name = ?, friend_id = ? WHERE friend_id = ?; ```	Used only when a friend has deleted their account.

Technical Solution

For full code see **(Appendix, Technical Solution, Server and Client Side SQL Approach Code)**

When implementing this SQL I decided to implement it, as before, as a module but this time using a class based approach enabling me to use the module with ease for both the client side code and the server side code.

The functions within the classes were split into groups, global, server side and client side. The global functions were used by both server side and client side code as well as server and client side SQL functions. The global functions also allowed me to print useful debug and error messages without having to change many functions.

Below is an example of a global function.

Function Name: execute_update	Purpose: Execute any sql update function with values taken from arguments.

Python

```python
def execute_update(self, sql: str, values: tuple):
    c = self.conn.cursor()
    try:
      c.execute(sql, values)
      self.conn.commit()
    except sqlite3.Error as e:
      print(f"UPDATE SQL ERROR: {e}\nfor {sql = }\n{values = }")
      self.conn.rollback()
    finally:
      c.close()
```

Testing

The testing of these functions was ongoing as the rest of the app was developing as each time a new feature was added a new SQL function was used which would tell me if it was working or not.

Scalability

To allow for easy scalability and understandability of my code I have used doc strings and type hints throughout. This allows me to easily see what a function In another module I have coded does without having to investigate and remember that function. It also allows me to keep track of all the datatypes. In addition it allows others to easily understand and build upon my existing code.

Below is an example of what they would look like:

```python
def get_pet_age_in_dog_years(pet_name: str, pet_age: int) -> str:
       return f"{pet_name} is {pet_age * 3} in dog years"
```

I have also subdivided big sections of my project into separate python files allowing for code modularity and reusability which helps with testing and keeping code clutter free.

Defensive Programming

SQL Injection

I was able to prevent SQL injection by following *Python (2021)*'s guidance on how to properly structure SQL statements to prevent SQL injection.

Try, Except

Throughout the code there is multiple use of try and except statements to catch any potential errors and handle them accordingly.

Testing

Key:
N - Normal
B - Boundary
E - Erroneous

Video Link	Description	Type	Pass/Fail	Notes
https://youtu.be/ZJXcGtBJlAA	Testing app start up functionality			
	Testing app starts up correctly when server is online	N	P	
	Testing app starts up correctly and displays appropriate error messages if the server is offline	N	P	
	Testing account creation			
	Testing an account can be created where all inputs are valid	N	P	
	Testing an account can not be created where the user has not entered valid details	E	P	
	Testing an account can not be created where the user_id is already taken	N	P	
	Testing account can not be created when password is not of the required length	B	P	
	Testing account can not be created when password is not of the required ‘strength’	N	P	
	Testing login			
	Testing an account can be logged into where all inputs are valid	N	P	
	Testing an account can not be logged into where user has not entered valid details	E	P	
	Testing an account can not be logged into where that same account is already online	N	P	
	Testing an account can not be logged into where the user does not exist	N	P	
https://youtu.be/pc-Yp6gSOjY	Testing friend request and accept system			
	Testing a friend request can be sent to a user	N	P	Friend request is not shown until page is refreshed
	Testing a friend request can not be sent where user has not entered valid details	E	P	

	Testing a friend request can not be sent where the user is already friends with the person they are requesting	E	P	
	Testing the recipient can accept the friend request and then the two users can communicate	N	P	
	Testing the recipient can reject the friend request and can therefore be ‘re-requested’ as a friend	N	P	
	Testing blocking system			
	Testing a friend can block another friend	N	P	
	Testing a friend can not block a friend that has blocked them	E	P	
	Testing a friend can unblock another friend	N	P	
	Testing you can't send messages to a blocked friend or receive messages from a blocked friend	N	P	
	Testing when a friend is blocked the block friend can see that they have been blocked	N	P	
https://youtu.be/UCs4XXesKlU	Testing send messages system			
	Sending images	N	P	Encryption is incredibly slow when files are large despite data compression.
	Sending emojis	N	P/F	Certain emojis are not correctly linked to buttons and moving between slides does not work as intended. I should have checked that the code for the emoji keyboard I took was completely working before implementing it. So emojis can be sent but some emojis don’t work. This however does not break the system due to suitable error handling.
	Sending normal messages	N	P	
	Testing user can not send a message to no one specific	E	P	
	Testing that when a user receives a message the messages are displayed if they are on the relevant chat page.	N	P	
https://youtu.be/pkoDUmCv1p8	Testing protocols, encryption, and serialisation work using wireshark	N	P	

https://youtu.be/CposvyjCMxg	Testing message queue system			
	Testing that when a messages is sent to an offline user they receive it once they come online	N	P	
	Testing that when a messages is sent to a user that can not currently receive messages they receive it once they can receive it.	N	P	
https://youtu.be/l-nK9xIkXCU	Testing change scree_name system			
	Testing synchronisation of details	N	P	
	Testing screen_name can't be specific names but can include parts of that specific name	B	P	
	Testing account deletion	N	P	
	Testing synchronisation of details	N	P	
Testing requesting data				
	Testing appropriate error handling	N	P	
	Testing data appropriate data is sent and saved	N	P	
Testing Misk UI stuff				
	Testing show hide password	N	P	
	Message entry box appropriately displays placeholder text	N	P	
	Testing all UI error messages are shown appropriately	N	P	
Testing password storage				
	Two passwords are the same but are stored differently	B	P	This passes specifically thanks to the addition of a salt
Testing encryption				
https://youtu.be/d1-b4S9dS7Y	Testing text is encrypted correctly.	N	P	
	Testing encryption of images	N	P	
Testing sql				
	Does the app retain changes made by the user once it is closed.	N	P	

Test To see if the inverse_s_box I calculated decrypts the binary values correctly
Explanation: If the inverse_s_box I calculated the output from this code should show the initial and decrypt values will be equal.

Code:

```
38  def sub_word(column):
39      row_nibble = int(column[:4], 2)
40      column_nibble = int(column[4:], 2)
41      sub_value = str(bin(int(s_box[row_nibble][column_nibble])))
42      sub_value = '{:0>8}'.format(sub_value.replace('0b', ''))
43      return sub_value
44
45  inverse_s_box = [[82.0, 9.0, 106.0, 213.0, 48.0, 54.0, 165.0, 56.0, 191.0, 64.0, 163.0, 158.0, 129.0, 243.0, 215.0, 251.0
46
47  def sub_word_inverse(column):
48      row_nibble = int(column[:4], 2)
49      column_nibble = int(column[4:], 2)
50      sub_value = str(bin(int(inverse_s_box[row_nibble][column_nibble])))
51      sub_value = '{:0>8}'.format(sub_value.replace('0b', ''))
52      return sub_value
53
54  transformed_column = '01010101'
55
56  encrypted_value = sub_word(transformed_column)
57  decrypted_value = sub_word_inverse(encrypted_value)
58
59  print(f'Inital Value:    {transformed_column}\nEncrypted Value: {encrypted_value}\nDecrypted Value: {decrypted_value}')
```

Output:

```
Inital Value:    01010101
Encrypted Value: 11111100
Decrypted Value: 01010101
```

Evaluation

Meeting requirements:

I feel the solution I provided is up to a high standard. I met all but a few of the objectives and have created a usable application. By keeping key ideas in mind while designing the UI I was able to create an easy to use and understandable UI that my client was happy with. While tkinter may not have the nicest graphical display or the most customizability I was able to work well within the constraints I had to create a functional version of the UI. Testing shows that my implementation was solid with suitable error messages given to the user and that I was suitably able to prevent my user doing anything that could internally break something such as quitting the application while requesting to delete their account.

The message sending functionality, the main purpose of the app, was well executed with it being shown to the user where they could send messages and the extent of what they could put in a message through manipulating the UI. Furthermore my implementation of a message queue allows a user to receive a message or any other data when they come online that was sent to them offline. The only potential flaw in my implementation is if the server is closed or crashes as then all data in the queue is lost.

The security and encryption side of the project was fascinating and mostly showed me how hard it was to implement all security measures from scratch. Despite this I feel I was able to make the most secure application to the best of my ability. My AES encryption worked properly and implementation of public key encryption was successful.

If my client wanted to host the server themselves I can simply give them the python server file to run however they would have to organise any changes such as if they wanted the server to be running even when their computer is off or if they wanted to remotely run the server.

Meeting Objectives

I have met all objectives except 9d and 9e

Objective	Evaluation
1a. The new user must complete a form where they enter their Username, Screen-name, Password.	The form is easy to use and looks like a typical create account form they have seen before which means the user will know exactly what to do. Suitable error messages are displayed if the user does not fill out the form correctly.
1b. The Username must be unique to all the usernames on	Implementing this was easy as it just involved checking to see if the chosen user Id already existed within the server's database. There is also other error handling such as not allowing the user to create the account if for

the app.	some reason the server returns the fact that it is unique but locally there is already a user of that name stored. The issue with this error handling is that it is not very robust and does not display this specific scenario's error message to the user; however, it is such an unlikely scenario to occur that I feel that what I have implemented is enough.
1c. The Screen-name does not have to be unique	The user does not necessarily have this information provided to them when choosing a screen name however that is ok as it will not impact their choice. Implementing this involved not checking the database for a unique screen name.
1d. Password must be entered twice for verification	If the user fails to meet the objectives requirement a suitable error message is shown. One key aspect to this implementation is that if this error occurs no data is sent to the server. This reduces the computational load on the server and the user's computer too.
1e. The password must be at minimum 8 characters long, at maximum 64 and contain numbers letters and special characters	The maximum length password is defined by the maximum character limit for a Bcrypt hash which is 72. By making the enforced limit 64 I ensured it would never cause an error however with more time I could push this limit up to the max of 72 with some iterative testing. If the user fails to meet the objectives requirement a suitable error message is shown. One key aspect to this implementation is that if this error occurs no data is sent to the server. This reduces the computational load on the server and the user's computer too.
1f. The user must be able to show/hide their entered password	The show hide password looks and behaves like a typical show hide password button the user will have seen before when creating another account so they immediately understand what it does and how it works. Importantly the state of the show/hide password button does not transfer over when they are swapping between the create account page and the login page. This allows for a smoother user experience.
g. Passwords must be hashed before they are stored	Passwords are hashed according to OWSAP guidance. For a future implementation I would seek out other guidance and check if it matches the OWSAP guidance to ensure I am being given accurate and reliable information. In addition, in a future expansion I would have to always be on the lookout for any password hashing advancements or changes to algorithms such as a flaw being discovered in Bcrypt to be able to change my system accordingly to ensure passwords are always stored safely.
gi. The hashing process must involve the use of salt and pepper to ensure maximum security	This was interesting to learn about and easy to implement using imported modules.
gii. The algorithm to hash the password must be Bcrypt	I feel my choice of Bcrypt as a hashing algorithm was suitable given my analysis of hashing algorithms in the beginning.
giii. The pepper must be stored securely outside of the database holding the	I successfully completed this objective by storing the password in windows credential manager The issue with my implementation of this objective that the pepper is taken directly from my windows secure key storage and therefore if the server

passwords	needed to be hosted on another computer that pepper would have to be securely transferred over to that computer's secure key storage. One way around this would be to use an online hardware security module.
2. To login the user must enter their username and password	In the process of implementing this feature's error messages I learnt about striking the balance between specific and general error messages and why modern systems don't just directly tell you your password does not match the given user ID but rather tell you that it does not match any credentials in their database. This is done because if a hacker didn't know anyones user Id but was then told though an error message that the person's user Id they have entered does exist they could then try and brute force a login for that specific user.
2i. There must be no way to recover or reset a password	There is no forgotten password button to be clicked and passwords are hashed before they are stored meaning they can not be recovered.
2ii. The user must be able to show/hide their entered password	The show hide password looks and behaves like a typical show hide password button the user will have seen before when creating another account so they immediately understand what it does and how it works. Importantly the state of the show/hide password button does not transfer over when they are swapping between the create account page and the login page. This allows for a smoother user experience.
3. The user must be able to add a new friend by typing the username of the person into a specific box	The box is located in the add friend area which the user has access to through a button with a suitable image indicating friend management. The user has easy access to their friend code and can copy it to their clipboard to send it to someone else. The user can also easily enter their friend's friend code with suitable error messages shown if something goes wrong. The responsive UI then clearly shows they have sent a friend request.
3a. The user must be able to block any of their current friends provided they themselves are not already blocked	The user is shown a box with a clear heading of friends which they are shown they can scroll up and down on through the use of a scroll bar to the side of the box. Having the friends being radiobuttons of the same design as their friend list in the messaging area shows the user that these are also their friends and they can click on them and manage them using the buttons provided. The responsive UI then turns the box to red if they have been blocked. This colour was discussed and tested with members of my household to ensure it was clear enough and had an obvious meaning. This colour also then transfers through to the message page so the user can also clearly see who they have blocked there. Unblocking is also an option and returns the button to its original state.
3b. The user must be able to unblock any of their current friends provided they themselves are not already blocked	See above objective
4. The user must be able to see their outgoing friend requests	Once the user has added a friend, a radio button in the same style as the radio buttons in their friend list appears in a clearly marked section.
5. The user must be able to see their incoming friend requests	There is a clearly marked section for incoming friend requests and the user is able to click on and choose to reject or accept any of those incoming friend requests. Once again the design and style of the radio buttons and buttons in this area remain consistent with the rest of the UI meaning the

	user instinctively knows what to do.
6. The user must be able to permanently delete their account	Implementing this was somewhat tricky. I had to change the user's user_id and screen name to deleted account (i) where I is a number. This is because of the unique constraint on the user_id row in the SQL database. I am pleased with my solution as while it does not guarantee two deleted accounts won't have the same name it puts the chance of that occurring at 1/100,000,000.
6a. Other users must not be able to send messages to or see messages between them and a deleted account	When an account is deleted that a user is friends with the screen name shown to the user is changed to deleted user (i) and the radio button is disabled meaning they can not see their message history with the deleted user. The disabled radio button along with the change of text clearly demonstrates to the user that this is intentional and not a bug.
6b. The user must be asked twice before they can delete their account	The user is asked in a popup box which directs their attention to the question being asked. When this box shows they can not click or alter the main application in any way or even close it (when they attempt to close it a suitable error message is shown as shown in testing) showing to them they must choose if they want to delete their account or not before they can do anything else.
7. The user must be able to request all data with the exception of the hashed password from the server	This objective was important to meet as it allowed my program to comply with the GDPR (to the extent that is reasonable for a project of this scope). The user is shown a file manager that they are familiar with and directed to choose an output location for their user data. The program will then write the data to a .txt file. The .txt format is suitable; it doesn't matter if the user doesn't have any other word processors installed such as word they can still open the file.
7a. The user must be able to choose where this data is stored	See above objective
8. The user must be able to change their screen-name to whatever they want provided it does not interfere with the code in any way.	There is a dedicated box where the user can enter their newly desired screen name and a button to commit the change. If the screen name is invalid a suitable error message is shown. The responsive UI then updates the users screen name so they can see the change in action.
9 The user must be able to send text, emoji or images to another user	The user is clearly shown that they can do one or more of these 3 options when sending a message. If they want to send an image a choose file popup is shown that is a file explorer native to their system. This file explorer is restricted to the choice of png and jpeg images only but the code also has error handling if they somehow try and choose something else. The emoji keyboard was taken from someone else's pre-existing code and I altered it to fit my purpose. It is shown as a popup with the same colour theme as the app making it obvious to the user it is part of the same application and allows users to insert desired emojis.
9a. These messages must be encrypted end to end using AES and a key known between the	The message is encrypted using my AES algorithm, and in the case it is an image it is compressed prior to encryption to reduce encryption time. The key is known between both users as it is an Epk which is sent securely along with the message. The responsive UI displays sent and received messages appropriately.

two users	
9b. These messages must be stored on the server in an encrypted form (changed to 'stored locally in an encrypted form' see: Server Side Storage Approach)	This was not too difficult to implement as it just involved storing the encrypted message and then encrypting the Epk before storing that alongside the message. For message retrieval and display operations were done in reverse order and because images are stored as the path to the image rather than the image themselves displaying the image was quick even if it was a huge image file.
9c. These messages must include date and time of sending	The 'timestamp' of the message is displayed above the message in a format common to how other message timestamps are formatted to make it clear to the user that it is not part of the actual message itself.
9d. The user must be able to delete any of their sent messages provided they have not blocked the user the messages has been sent to or that user has not blocked them.	This objective was not met mostly because I ran out of time. However, considering how I would Implement it, I feel I would have asked the client if this feature was really necessary for them. This is because I would need to implement some sort of right click action in tkinter where it selects the specific message the user has right clicked and because all of my messages are in a text box pasted as plaintext this would be incredibly difficult to achieve as there would be no way to distinguish what message was clicked on. I could have had each message as its own radio button however that has the potential to become visually cluttered and clunky.
9e. The deleted message must me replace with 'Message Deleted'	See above objective reasoning.
9f. The user must not be able to send any messages to or receive any messages from a blocked friend or a friend they have blocked	This was just a matter of either disabling the radio button or making it red and disabling the sending of messages for each relevant situation. This allows the UI to meet objectives 10 and 11 and makes it obvious to the user what has happened and the new imposed restrictions.
10. The user must be able to see the message history between them and a friend they have blocked	See 9f
11. The user must not be able to see the message history between them and a friend who has blocked them.	See 9f
12. When the user closes the application	All changes the user makes persist as any changes they can make are subsequently stored into a database and loaded from that database when

all changes they have made must persist	they login again. In a future expansion there could be the ability to customise the UI such as a light or dark mode however within the constrained time of the NEA I feel I did as much as I could.
13. The system must have an ‘easy to use’ UI where ‘easy to use’ is defined as meeting all sub objectives below	While this objective was met were this a real system the ease of use of the UI would be defined by numerous rounds of testing UI and UX experience within the targeted demographic. In addition it would have to comply with accessibility requirements for colour blind people and people with any kind of visual impairment. However I have tried to make my UI easy to use by keeping a consistent theme throughout the UI. For example all buttons look the same and all text entry fields look the same.
13a. The UI must use understandable symbols wherever it is trying to communicate a message through the use of those symbols	When seeing how well someone understands a message and icon you can never guarantee that everyone will understand the message. This problem is faced at a much greater scale by graphic designers in relation to the dumping of radioactive waste; however, here I solved it by deciding to go with the most frequently used icons assuming that therefore the majority of people would understand them.
13b. For all actions taken by the user the user must be able to visually see their action has had an effect	Every UI does this so it was important I did too. Implementing this was simple as tkinter allows suitable customisation of what happens when a button or something else is clicked. Were the server slower I may have had to make a way to notify the user that something internally was waiting to complete its process such as changing the cursor to a spinning circle.
13c. There must be placeholder text where the user needs to enter text that is removed when they go to enter text	All text boxes have clear understandable placeholder text where necessary. This placeholder is removed when the user focuses in to send a message and is added back when the focus is left.

Client Feedback

Previously in the project I had decided to make a change to how data was stored and made contact with the client to explain this change to him. That can be seen at Server and Client side storage approach.

I sent the python file containing the code and asked him to run the app and use it for a few days then return with feedback. His feedback is below.

“*I love the overall design of the program, as well as how easy it is to use. You accomplished virtually all of my requests which is great. Sending messages and files works great and the recipient receives them super quickly. I am very happy with the security aspect of the program: the fact that no messages are stored on the server brings me confidence that my beloved memes will not be stolen by a potential man-in-the-middle. I am slightly concerned about what would happen if the server crashed after my recipient hadn't had a chance to log on. From what I understand these messages would be*

lost, so it would be nice if you could find a way to retain them in a future version. I did also ask for users to be able to delete messages after sending them, and this doesn't appear to be included in the app's functionality? Again, it would be nice to see that in a future version."

Analysis:
Ben was clearly very happy with the project as a whole and was impressed at the speed messages were received. He was pleased with the change to store messages client side and although he is a little confused on what that actually means for improved security he has got the spirit.

It is clear the steps he would like me to take next are in regards to server crashes and therefore a potential loss of data. One potential implementation would be to store everything in the message queue in some sort of database the moment it is received by the server and then delete it when it is sent to the client.

Key learning points:

- Communication with the Client is vital if you have not met their expectations not just when you are making a change
- Giving the client a basic understanding of the full extent of the key features you have added even if they are not UI related, in my case, all the steps involved in making sure the messages and passwords are secure, would help build their confidence in the project.
- Putting a bit of effort into UI pays off as that is the side the user is interacting with

How the outcome could be improved

If this project were to expand or be redone many changes would be made.

In terms of UI and functionality, I would like to add features such as group chats, the ability to send files, videos, audio and even the ability to call other users. One other aspect of UI I would implement would be a received and read function similar to the one user in whatsapp. If I expanded the functionality of the server as described lower down in this section implementing this would be fairly easy as whenever the server echoes the message was sent the client would know the message was received and then when the receiving client echoes the message was received the sending client would know it was seen.

Were the project to be expanded into an application I could distribute online I would need to make a few changes. One change I would need to make would be to package all client-side code neatly into an installer package which instals all the compiled Python code and anything else the user would need to execute the program. This would make it easy to distribute the program without having to share actual Python files that an average user may not know what to do with. Another change I would have to more strictly comply with the GDPR such as creating a user privacy policy, reviewing internal data transfer and adding measures to mitigate data breaches.

In terms of the actual code, there would be some amendments and changes to make.
Many of the networking protocols were designed and coded with future expansion in mind so creating bigger or slightly different types of servers would not be too difficult as they could all have the same solid functioning backbone.

In the actual server and client-side code, I would not have redundant data being sent such as the screen name or recipient name being sent twice in the same data package. In addition, I would make it more obvious what data is automatically filled out when sending data and what data I need to

provide to the function. This would make expanding both the server-side and client-side code much easier. I would also have a further look into asynchronous processing and threads to try and make my server more responsive and allow for easier future expansion.I would also try to optimise my encryption algorithm, or in another scenario use a pre-built one, as I had to compress every image sent or the encryption process would take too long. I could have also implemented an echo protocol where whenever a user or the server receives data they echo it back to confirm they have received it. With this implementation and some changes in the code to allow for suitable handling if no echo is received it would solve the issue of data being lost when the server crashes. For example if one user sent a friend request and then the server crashed their send friend request would be deleted on their end and they could then resend it.

In terms of security, there are a few glaring flaws in my system which I could not find a suitable workaround within the time I had. One issue was that when the client initially connects to the server the code does not have access to the users public and private keys until they are logged in. This is reasonable but means that upon initial contact the server has no guarantee who they are receiving information from or sending information to. A way around this would be storing each user's public key in some sort of global public database and having their private key accessed when they log in to their computer. Having the private key accessed when the user logs into their computer may prove difficult so I may have to look to other avenues of storing public and private keys. Furthermore, the server changes public and private keys each time it runs so the clients can guarantee that whoever they are talking to at the start remains the person they are talking to but the client can not guarantee this is the server. As with the above issue, I could look at storing the server's public key in a global database and having the client code retrieve it as necessary. This would require the server's private key to remain static and therefore need to be stored somewhere securely.

Another security issue is the lack of public-private key rotations. While doing research I discovered that password rotations are actively discouraged as they weaken security however pepper and public-private key rotations are sensible. (Why Periodic Password Changes are Not Recommended by NIST, 2023) If I had a robust system of public-private key storage whereby the server's public key is accessible to all, a rotation of public-private keys would be simple to implement. (Why Periodic Password Changes are Not Recommended by NIST, 2023)

Another security issue is that I have coded my own encryption algorithm. No matter how robust I think my coding is, it will never be as robust and secure as one coded by professionals. Therefore in a future expansion I would switch to using one built by professionals.

I would need to do more research into the correct use of peppers. Through my research, I found different sources saying different and conflicting things about the use of peppers and as such I just went with the guidance given by OWSAP as they were my most trusted source. In addition, I would need to look into how Python stores variables in RAM when executing code. This is because for brief moments in both the client and server code they deal with the user's password in an unhashed format. If an attacker was able to look into the RAM of the computer while the program was running they would be able to see the unhashed password.

Finally, I would need to be transparent with my users about how I secured their data to give them the most confidence in my application.

References

Password Storage - OWASP Cheat Sheet Series (no date).

https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html.

Team, S. (2024) The top 10 password cracking techniques – and how to outmaneuver them.

https://stytch.com/blog/top-10-password-cracking-techniques/.

What is a salt and how it boosts security? (no date).

https://www.loginradius.com/blog/identity/what-is-salt/#:~:text=Ideally%2C%20the%20length%20of%20Salt,to%20passwords%20with%20specialized%20characters.

Isaac Computer Science (no date).

https://isaaccomputerscience.org/questions/dsa_datastruct_31?examBoard=all&stage=all.

Greenberg, A. (2020) 'What is a side channel attack?,' WIRED, 21 June.

https://www.wired.com/story/what-is-side-channel-attack/.

Infospoint (2023) GPU Memory Attacks: the next generation of cybercrime | Infospoint.

https://www.infospoint.com/gpu-memory-attacks/.

Isaac Computer Science (no date b).

https://isaaccomputerscience.org/concepts/data_encrypt_encryption?examBoard=aqa&stage=a_level&topic=encryption.

Types of encryption: symmetric or asymmetric? RSA or AES? | Prey blog (2021).

https://preyproject.com/blog/types-of-encryption-symmetric-or-asymmetric-rsa-or-aes#:~:text=Asymmetric%20and%20symmetric%20encryption%20are,a%20private%20key%20for%20decryption.

ParthJadhav (no date) GitHub - ParthJadhav/Tkinter-Designer: An easy and fast way to create a Python GUI 🐍. https://github.com/ParthJadhav/Tkinter-Designer.

All About Python (2021b) How to create a real time chat app in Python using socket programming | Part 1. https://www.youtube.com/watch?v=hBnOdIg0jAM.

Bek Brace (2020) Python Network Programming #3: TCP chat room (Server and multiple clients). https://www.youtube.com/watch?v=nmzzeAvQHp8.

Francisrstokes (no date) githublog/2022/6/15/rolling-your-own-crypto-aes.md at main · francisrstokes/githublog. https://github.com/francisrstokes/githublog/blob/main/2022/6/15/rolling-your-own-crypto-aes.md.

Chirag Bhalodia (2022) How to solve AES Mix Column Transformation | Mix Column Transformation in AES | Solved Example. https://www.youtube.com/watch?v=WPz4Kzz6vk4.

O3DE: JSON serialization of O3DE data Types (no date). https://docs.o3de.org/docs/user-guide/programming/serialization/json-data-types/#:~:text=The%20primitive%20types%20used%20in.AZstd%3A%3Astring%20and%20OSString%20.

Python socket receive - incoming packets always have a different size (no date). https://stackoverflow.com/questions/1708835/python-socket-receive-incoming-packets-always-have-a-different-size.

Tech With Tim (2020) Python Socket programming Tutorial. https://www.youtube.com/watch?v=3QiPPX-KeSc.

CrypTool Portal (no date). https://www.cryptool.org/en/cto/aes-step-by-step.

Boppreh (no date) aes/aes.py at master · boppreh/aes. https://github.com/boppreh/aes/blob/master/aes.py#L125.

Tasos-Py (no date) AES-Encryption-Classes/aes_encryption.py at master · tasos-py/AES-Encryption-Classes. https://github.com/tasos-py/AES-Encryption-Classes/blob/master/aes_encryption.py#L328.

Computerphile (2019) AES explained (Advanced Encryption Standard) - Computerphile. https://www.youtube.com/watch?v=O4xNJsjtN6E.

Chirag Bhalodia (2022a) AES Inverse Mix Column | How to solve AES Inverse Mix Column | Inverse Mix column Solved Example. https://www.youtube.com/watch?v=SDrzMyqi2Sc.

GfG (2022) Man in the Middle attack in Diffie-Hellman Key Exchange. https://www.geeksforgeeks.org/man-in-the-middle-attack-in-diffie-hellman-key-exchange/.

Simply Explained (2017) Asymmetric Encryption - Simply explained. https://www.youtube.com/watch?v=AQDCe585Lnc.

Practical Networking (2021) Public and private keys - signatures & key exchanges - cryptography - practical TLS. https://www.youtube.com/watch?v=_zyKvPvh808.

Practical Networking (2021a) How SSL & TLS use Cryptographic tools to secure your data - Practical TLS. https://www.youtube.com/watch?v=aCDgFH1i2B0.

Practical Networking (2021b) How SSL & TLS use Cryptographic tools to secure your data - Practical TLS. https://www.youtube.com/watch?v=aCDgFH1i2B0.

Victoria Drake (2023). https://victoria.dev/.

jsonpickle Documentation — jsonpickle 1.4.3.dev0+g5b8d3ea.d20201130 documentation (no date). https://jsonpickle.github.io/.

How to check if object is instance of new-style user-defined class? (no date). https://stackoverflow.com/questions/14612865/how-to-check-if-object-is-instance-of-new-style-user-defined-class.

Computerphile (2018) How Signal Instant Messaging Protocol Works (& WhatsApp etc) - Computerphile. https://www.youtube.com/watch?v=DXv1boaIsDI.

Python, R. (2021) Preventing SQL injection attacks with Python. https://realpython.com/prevent-python-sql-injection/.

Why Periodic Password Changes are Not Recommended by NIST (2023). https://www.packetlabs.net/posts/periodic-password-changes/#:~:text=To%20protect%20enterprise%20assets%20and,Standards%20and%20Technology%20(NIST).