# ­AQA A-Level Computer Science (7517) NEA

The Hourglass Protocol

## Encrypted Peer-to-Peer Data Communications System / User-Distributed Anonymous Network

Shah Anwar

# Ilford County High School

# Centre Number: 13327

# Candidate Number: 5125

Contents

[AQA A-Level Computer Science (7517) NEA 1](#_Toc92040099)

[Encrypted Peer-to-Peer Data Communications System / User-Distributed Anonymous Network 1](#_Toc92040100)

[Ilford County High School 1](#_Toc92040101)

[Centre Number: 13327 1](#_Toc92040102)

[Candidate Number: 5125 1](#_Toc92040103)

[1.1 Introduction to Problem 3](#_Toc92040104)

[1.1.1 Main Background 3](#_Toc92040105)

[1.1.2 Solvability of the Problem through Computational Methods 4](#_Toc92040106)

[1.2 Research and Third Parties 5](#_Toc92040107)

[1.2.1 Research Outline 5](#_Toc92040108)

[1.2.2 Identification of Third Parties 6](#_Toc92040109)

[1.2.3 Primary Research 6](#_Toc92040110)

[1.2.4 Secondary Research 6](#_Toc92040111)

[1.2.5 Summary of Research 6](#_Toc92040112)

[1.3 Prototyping 7](#_Toc92040113)

[1.3.1 Objectives 7](#_Toc92040114)

[1] Analysis

# 1.1 Introduction to Problem

## 1.1.1 Main Background

The purpose of this NEA is to create a suitable alternative to existing ‘private’ communication applications and protocols. This project was selected due to certain drawbacks concurrent applications face when distributing data over the internet, which can be condensed into three main problems:

1. **Reliance on centralised servers.**

Each user of the specified protocol must direct data through a central server, as opposed to directly connecting to recipients. This is problematic as it allows the owners of the server to collect metadata on users’ communications, thereby permitting the violation of privacy.

1. **Not truly anonymous.**

Each instance of data transmitted over the specified protocol can be traced back to a specific machine, and by extension a real-world person. This can be a good thing as it allows law enforcement and moderators to trace users using the network for malicious purposes (e.g., harassment, death threats, explicit material, etc.).

However, this is also a problem as it can allow the adversaries of certain users (i.e., whistleblowers hiding from rogue states) to figure their identity and potentially harm them. In countries such as China or Turkey, where information is heavily censored, my project hopes to allow citizens of said countries to exercise their freedom of speech as it is a fundamental human right.

1. **Use of insecure encryption/not encrypted**

Unencrypted data sent over the internet allows anyone with the right tools and methods to view what is being sent between users – a total invasion of privacy.

My goal is to design a communication protocol, that allows the use of external applications through it, that is as free as it can be from the aforementioned problems.

# 1.2 Research and Third Parties

## 1.2.1 Research Outline

The research that is to be conducted for this project can be summarised into the following steps:

1. Identification of Third Parties

Before analysis begins, it is necessary that potential users and stakeholders are identified so that they may be consulted during the development of the project. Third parties can offer important external opinions to ensure that all sides are considered before a decision is made.

1. Primary Research
   1. Consultation of Third Parties
      1. Interviews

Interviews are to be conducted with a select number of individuals due to time restrictions placed upon this project. Even still, having a conversation with potential stakeholders can allow me to identify and analyse what users truly want, as my own opinions are subject to bias.

* + 1. Questionnaires

Unlike interviews, questionnaires are not subject to time restrictions as it doesn’t require me to actively spend time in extracting information from third parties; I only have to create the questionnaire and send it to them.

1. Secondary Research
   1. Analysis of Existing Systems
      1. Messaging Applications – The methods utilised by so called ‘private’ messaging apps (such as Telegram and Signal) can be analysed to see how they function within a given system and modified to be used in this project.
      2. Transmission Protocols – Similarly, the methodology utilised in certain protocols and networks (such as BitTorrent, TOR, and I2P) can be adapted and utilised in this project.
      3. Existing Algorithms – Due to time constraints, I am unable to design my own advanced encryption and hashing algorithms. Instead, I can analyse existing algorithms and hard code it into the project itself rather than relying on external libraries and code.

## 1.2.2 Identification of Third Parties

Since WhatsApp updated its privacy policy in early 2021, there has been a massive influx of new users to more ‘private’ messaging services such as Signal or Telegram. It is clear that many individuals care about their privacy when using services on the Internet, which is why I have decided that every individual with an internet connection is a potential user.

Rather than creating this project for specific ‘privacy-conscious’ users to use, this project should aim to be usable by everyone and should serve as a replacement for mainstream services.

## 1.2.3 Primary Research

*I have created and sent a questionnaire to various individuals, the results of which can be found in the appendix.*

### Interview with Potential User #1 – A. Haque

**Question**: How safe do you think your data is when using the Internet? Do you trust apps like WhatsApp and Facebook Messenger to keep your messages safe and secure?

**Response**: I have been much more vigilant with the apps that I use for messaging since the controversy with WhatsApp regarding their change in Privacy Policy began. Nowadays, I tend to read the policies more in depth before accepting them – before I used to just accept them without even looking.

I, and those whom I regularly communicate with, have switched to more privacy-oriented apps like Signal and Telegram; but even then, there is no way to fully trust them.

How do we know where our data is being sent to? How do we know if contemporary cryptography algorithms have been cracked or not? Nothing is truly secure nowadays.

**Question:** How important is a user interface to you when using certain applications?

**Response:** Not at all important. Coming from using IRC back in the 90s to currently working mostly with Linux, I am not a user that absolutely requires a graphical interface to run a program, although having one helps sometimes.

**Question:** What knowledge do you have of peer-to-peer networking? Think Tor Browser, BitTorrent, and I2P.

**Response:** Other than BitTorrent, I have no experience of any of the programs you mentioned, but I have heard of Tor Browser before. I understand that data is sent through other nodes on the same network rather than from a server in peer-to-peer networking, I myself hosting many torrent seeds on my home server.

**Question:**

**Response:** BBB

### Interview with Potential User #2 – T. Robertson

**Question:** AAA

**Response:** BBB

## 1.2.4 Secondary Research

### Analysis of Existing Systems

#### Telegram and Signal

Signal and Telegram are both examples of centralised, encrypted messaging services marketed towards privacy-conscious consumers.

Diagram

Description automatically generated[Telegram Message Encryption Layout] {Source: Telegram Messenger Inc.}

Inspiration and ideas taken:

As shown in the diagram above, messages sent using Telegram are encrypted end-to-end by using a proprietary system called ‘*MTProto’*. We can see that the packages are sent as a combination of encrypted data and identifiable message keys. Using this as inspiration, I have designed the packages that will be sent across the Hourglass network via the User Datagram Protocol (like how Telegram uses TCP or HTTP).

#### BitTorrent

The BitTorrent protocol is a decentralised, peer-to-peer file-sharing protocol which allows users to distribute data over the internet. Instead of downloading files from a single server, the protocol allows users to upload and download from each other.

#### The Onion Router (TOR)

The Onion Router, or TOR for short, is an open-source anonymity network and software suite which directs user internet traffic through an anonymised peer-to-peer network known as the TOR network, consisting of voluntary servers which act as nodes to transmit data across.

A picture containing diagram

Description automatically generated

[Diagram of the Onion Routing Principle] {Source: *Harrison Neal, GNU Free License*}

In certain circumstances, TOR allows the transmission of data between two systems without either of them knowing each other’s IP Addresses, using ‘onion services’. Servers are assigned a unique URL known as an ‘onion link’ which can be distributed across the internet to allow users to connect to it – all without knowing the server’s IP Address (unlike how DNS is used in normal HTTPS communications, where the URL links to a specified IP Address).

#### Internet Relay Chat (IRC)

Internet Relay Chat, often referred to as IRC, is an application layer protocol that allows users to communicate with others using text using a specific IRC Client. Created in 1988 by Finnish computer scientist Jarkko Oikarinen, IRC allowed users to participate in conversation channels on voluntarily run servers. The idea of the IRC protocol being a framework for others to utilise and run their own communication “networks” is something I intend to replicate in this project.

Text

Description automatically generated

[Example IRC Interface: Weechat] {Source: ‘*hullap’ (pseudonym) – Public Domain*}

#### Cryptographic Hash Functions (SHA and DSA)

The Secure Hash Algorithm (SHA) and Digital Signature Algorithm (DSA) are both families of contemporary cryptographic hash functions.

#### Asymmetric Encryption Algorithms (RSA)

Asymmetric encryption algorithms such as RSA allow users to exchange data using public and private key pairs, rather than relying on a single shared key.

#### Diffie–Hellman key exchange

The Diffie-Hellman key exchange method allows users to securely exchange cryptographic keys over an unsecured channel (i.e., The Internet).

A picture containing icon

Description automatically generated

[Explanation of Diffie-Hellman Key Exchange] {Source: A.J. Han Vinck, University of Duisburg-Essen – Public Domain}

## 1.2.5 Summary of Research

# 1.3 Prototyping

## 1.3.1 Objectives

1. Without leaving the scope of the A-level Computer Science syllabus, the protocol should strive for maximum possible anonymity of users.
   1. Transmission of data across the network should be in such a way that reduces the ways in which the identity of users can be uncovered (e.g., using Onion Routing or Proxies)
   2. Metadata used within the protocol should be kept to a bare minimum; it must be implemented in such a way that it cannot be used to fingerprint a certain user (e.g., it should only contain things that require the protocol to function such as timestamps and destination IP addresses)
2. Data sent across the network using the protocol must be accessible to the recipient and the recipient alone.
   1. This can be achieved using a combination of asymmetric and symmetric encryption, as well as limiting the number of nodes that can access a particular packet in the network
3. Reliance on a centralised server should be kept to a bare minimum also; data should never have to pass through a central server to reach its destination.
   1. A central server should only serve the purpose of indicating whether or not a specific user is online, and what proxy they are accessible from
4. Algorithms such as those for cryptography or data manipulation should be self-coded to the restraint that python allows; if required, certain functions can be programmed in languages such as C and then called by Python.
   1. For data manipulation, algorithms such as linear searching and number conversion should be self-coded rather than using the in-built python functions bin() and hex()
   2. For cryptography, algorithms such as asymmetric encryption, decryption, and key generation should be self-coded instead of using external modules
5. Instead of implementing a regular database, a hash table should be used server-side to maximise efficiency (considering that in a real-world scenario, a very large amount of people will be connecting to a server at once).
   1. The NodeID (the username) should be hashed to retrieve information from the database, rather than using the NodeID itself in an SQL query in order to maximise efficiency
6. As a demonstration, a simple application of the protocol, such as a messenger function, should be implemented into the program to showcase an aspect of the various uses of the protocol.

## 1.3.2 Acceptable Limitations

My prospective users and I have understood that it would be a very time-consuming process for me to implement any complex encryption algorithms which are beyond the scope of this NEA. Therefore, I have limited myself to implementing more common and simpler algorithms such as older versions of RSA, AES, and SHA.

In addition to this, all consulted prospective users have unanimously agreed that as a first iteration of the project, the framework should have a simple showcase programmed alongside it, such a private messenger application, before moving onto more advanced use cases such as video streaming or file sharing.

[2] Documented Design

# 2.1 Overview

## 2.1.1 Conceptual System Flowchart

Diagram

Description automatically generated[Client System Flowchart] {Source: Self-made}

## 2.1.2 IPSO Chart

|  |  |
| --- | --- |
| **Input**   * Data for Transfer * NodeID of intended recipient * New NodeID of friend * Config File (Both input and storage) | **Process**   * Any type of data can be converted to bytes which are sent as packets over the network via UDP * Config file works for any client if copied, similar to a blockchain wallet – all personal data belonging to the user is stored here securely after being encrypted with a password |
| **Storage**   * Client Config File, file for each profile which can then be copied onto other clients for access * Server Database File, SQL hash table containing every user registered to the network and their corresponding information (i.e., NodeID, Public Key, Exit IP address, Online Status, etc.) | **Output**   * Either all assembled data sent to specified program through loopback SOCKS proxy – akin to the Tor Network * Or data is used by the program itself to output important data such as server notices or errors direct to the user |

# 2.2 Algorithm and Function Design

## Prime Number Generation - Sieve of Eratosthenes

### Description

Attributed to the 3rd century BCE Greek Polymath *Eratosthenes of Cyrene*, the Sieve of Eratosthenes is one of the most efficient ways to calculate a list of prime numbers up to a specified limit.

### Pseudocode and Explanation

The algorithm works by starting with a sequence (in our case an array) of numbers from 2 to the limit, *n*. Starting with the number 2, the algorithm removes all numbers from the list which are a multiple of 2. The algorithm repeats this for all numbers 2 to (n/2) which are still in the list. The result is a list of prime numbers from 2 to n.

|  |
| --- |
| Primes = [True] \* N  Loop from X to N/2:  X = X + 2  If Primes[X] equals True:  Y = 0  Z = 0  Loop indefinitely:  Z = (X + Y) \* X  If Z < N:  Primes[Z] = False  Y = Y + 1  Else:  Break Loop  Loop from A to Length (Primes):  If Primes[A] Equals True:  Append A to array Result  Return Result |

### Uses

This algorithm is required to generate prime numbers for the RSA (see following sections) asymmetric cryptosystem.

## Hash Function – FNV (Fowler–Noll–Vo) Hash

### Description

The FNV hash algorithm is a non-cryptographic hashing algorithm designed for hash table and checksum use (but not cryptography).

### Pseudocode and Explanation

Note: All bytes are unsigned integers.

2166136261 and 16777619 are non-random 64-bit integers which are specific to the algorithm.

|  |
| --- |
| Hash = 2166136261  Loop for each byte of data in Data:  Hash = Hash \* 16777619  Hash = Lower 64 bits of Hash  Hash = Hash XOR (Byte of Data)  Return Hash |

### Uses

The FNV hash can be potentially used server-side for the hash table that will contain the data of all users on the Hourglass network.

## Custom Hash Function

### Description

In the event that Python does not allow me to program the FNV Hash myself, I have designed a back-up hash function that should suffice for use in a hash table.

### Pseudocode and Explanation

|  |
| --- |
| Data = Convert Data to Unicode (UTF-16) decoded byte array  X = 0  Total = 0  For each character in Data:  Attempt:  Total = Total + (Ordinal (CHAR) \* X)  X = X + 1  Exception:  Total = Total + X  Return Total |

### Uses

Similar to the FNV hash above, I can implement this function for server-side hash table use if Python doesn’t support bitwise operations without the use of external libraries.

## Encryption – RSA (Rivest–Shamir–Adleman) Cryptosystem

### Description and Background

The RSA cryptosystem is an asymmetric, public-key cryptosystem that utilises the difficulty of factoring the product of two prime numbers to encrypt data.

The basic principle of the RSA cryptosystem revolves around a single mathematical identity:

**(me)d ≡ m mod n (**where e and d also fulfil **(md)e ≡ m mod n)**

The cryptosystem relies on the “factoring problem” to remain secure, which is the difficulty for a conventional computer to factorise the product of two prime numbers. However, with the slow advent of quantum computing, we may see it being broken in the near future as these types of computers can solve the problem very fast.

Due its time-consuming nature, data is often first encrypted with a symmetric encryption algorithm. The symmetric key is then encrypted using the RSA public key of the recipient and is sent with the data.

### Pseudocode and Explanation

Key Generation

1. Generate two random prime numbers, *p* and *q*.
2. Calculate the value of *n*, where *n* = *p* \* *q*.
3. Calculate *phi* (Euler’s Totient Function), where *phi* = (*p*-1)\* (*q*-1).
4. Calculate *lambda* (Carmichael’s Totient Function), where *lambda* = *phi*/gcd(p-1, q-1)**a**
5. Select an integer *e*, such that gcd(e, *lambda*) = 1 and 1 < *e* < *lambda*.
6. Calculate *d*, where *d* = modinv(*e*, *lambda*)**b**
7. Return n, e, and d in tuples as (n, e) and (n, d).

**NOTE:**

**a – gcd(x, y) returns the greatest common divisor between x and y**

**b – modinv(x, y) return the modular multiplicative inverse of x modulo y**

**The public key is (*n, e*), whereas the private key is (*n,* *d*).**

Encryption

## Circuit Generation Algorithm

### Description and Background

### Step-by-Step Process and Explanation

Server-side Node List Generation

1. The client sends a

Probabilistic End-Node Selection

Chart, line chart

Description automatically generated

[Probability Mass Function of Geometric Distribution]

{Source: Wikimedia Commons, Author: Skbkekas, Creative Commons License}

The *Geometric* distribution models the number of trials required to achieve a single success where the probability of success is constant, and trials are independent of each other.

Any node in the circuit has a 1 in 3 chance of becoming the end node during circuit generation; the end node is chosen probabilistically. The program generates a random integer between 1 and 3, with the node becoming an end node if the integer equals 1.

Since the random number generation algorithm is independent each time it is run, the circuit generation process fulfils each criterion of the geometric distribution, allowing us to model the size of a given circuit using it.

**X ~ Geo (1/3)**

X models the number of nodes in a given circuit, P(X = k) giving us the probability of the number of nodes equalling k. P(X = k) can be calculated using the formula:

**P(X = k) = p(1 - p)k – 1**

|  |  |
| --- | --- |
| k | P(X = k) |
| 1 | (1/3) x (2/3)0 = 1/3 = 0.3333 |
| 2 | (1/3) x (2/3)1 = 2/9 = 0.2222 |
| 3 | (1/3) x (2/3)2 = 4/27 = 0.148148… |

Therefore, we can see that a circuit with only one node has the highest probability of occurring. However, this doesn’t mean that most circuits have a length of one, it is only probabilistic. Rather, the greatest use for the geometric distribution in this circumstance is calculating the average/mean value of the number of nodes in a circuit, also known as the expected value, or E(X).

**E(X) = μ = (1** **÷ p) = 3**

Therefore, the average number of nodes in a circuit on the Hourglass network is equal to 3, as intended.

Recursion Completion

Finally, once the end node has been determined, it sends a packet up the circuit chain to the start node to indicate that the circuit is complete, ending the recursion.

Join Circuit

Join Circuit

End Circuit

Create Circuit

Node End

Node 2

Node 1

Owner

## Data Manipulation

### Linear Search

Pseudocode

### Decimal to Binary Conversion

Pseudocode

### Binary to Decimal Conversion

Pseudocode

### Base64 to Decimal Conversion

Pseudocode

### Decimal to Hexadecimal Conversion

Pseudocode

## Data Message Creation

## Data Message Digestion

## Console/Command-Line Handler

## Config Handler

# 2.3 Data Structure Design

## Matrix

### Description

A matrix is a certain type of two-dimensional array in which all elements are of the same data type.

### Pseudocode, Diagram, and Explanation

### Uses

## Queue

### Description

A queue is a form of abstract data structure in which elements are added at one end and removed from the other. It is an example of a first-in-first-out, or FIFO, data structure.

### Pseudocode, Diagram, and Explanation

### Uses

Queues can be used to organise incoming packages for processing, rather than trying to work on all of them simultaneously (something which is hard to do on systems with ‘weaker’ hardware).

## Graph

### Description

### Pseudocode, Diagram, and Explanation

### Uses

### Description

A stack is a form of abstract data structure where elements are added at one end and removed from the same end. It is an example of the last-in-first-out, or LIFO, data principle.

### Pseudocode and Explanation

### Uses

Stacks are required for the depth-first traversal algorithm to search graph data structures – such as when navigating a circuit of interconnected nodes.

## Hash Table

### Description

A hash table is a data structure that implements an associative array by storing data as key-value pairs. The data in the array is indexed by the value of its hash. Hash tables allow efficient searching, having a time complexity of O(k) in the best circumstance.

### Pseudocode and Explanation

### Uses

# 2.4 Database Design

Diagram

Description automatically generated[Level Zero DFD] {Source: Self-made}

### Data Dictionary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Type** | **Description** | **Valid Data** | **Invalid Data** |
| Header Type | String (from ENUM) | Type assigned to each transmission between connected nodes; used by the recipient node to handle accordingly | **ASK** – Ask individual connected node if they have the IP address of a specified NodeID  **ASKALL** – After asking all connected nodes for data and failing, ask all individual nodes to then ask any nodes they are connected to  **REPLY** – Reply to ASK/ALL request sent by connected node with credentials of the NodeID specified  **REPLYNULL** – Reply to ASK/ALL request that no credentials assigned to the NodeID specified were found  **CREDGET** – Ask unknown node for its credentials, (i.e., ask the user with a certain IP address if they are *NodeID,* if so, check using cryptography and if successful exchange keys and establish connection)  **END** – Send to selected node to terminate connection, keep sending message repeatedly at a certain interval until REPLYEND received  **REPLYEND** – Reply to END message, thus terminating the connection  **DATA** – Data sent to nodes after connection established, either encrypted or signed. Sent alongside “Data Type” assignment (see below). | Any strings not specified in the valid data section |
| NodeID | Hex Value | A hexadecimal value assigned to each node on the network, potentially has some sort of cryptographic importance when implemented | Hexadecimal value in the form of a string with only characters from the following list: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F] | Any data types not a string, or a string with any characters other than ones specified |
| Exit IP Address | Numerical value combination in String Form | Assigned to every connected NodeID saved, the IP address denotes the location of a specific device (in this case the node) on the internet | String in format X.X.X.X where X denotes any number within 0-255 (for IPv4), or in format x:x:x:x:x:x:x:x where x denotes a 16-bit hexadecimal value (for IPv6). | Anything other than that specified in the previous section |
| Data Type | String (from ENUM) |  | TEXT  FILE  CRED – Credentials of sender node (i.e., Encryption Keys, IP Addresses, NodeIDs – either of itself or as a reply to an ASK request) | Anything other than that specified |
| Online Status | Boolean | True or False that denotes the availability of a specified user on a network | True  False | Anything other than a Boolean data type |
| CircuitID | Byte string (which contains data and a signature) | A byte string that contains the ownership and the name of a specified circuit on the network. Used to verify that the exit node provided is currently in control of the desired user (through the signature) | Byte string – originally a string and signature pair | Anything other than that specified |

# 2.5 Class Design

Diagram

Description automatically generated

[Conceptual Client program class diagram] {Source: Self-made}

As visible in the diagram above, most classes in both the client and server programs will not be run as objects – most of them acting as a collection of similar subroutines (such as transmission handlers or configuration functions). However, some classes such as those for connections and circuits are to be stored as individual objects, with their own attributes.

# 2.6 UI Design

## 2.6.1 Overview

**Login Menu Input**

|  |  |  |
| --- | --- | --- |
| Data Item | Data Type | Validation/Restrictions |
| Server IP Address | String | Four integers within range 0-255 (inclusive) separated by full stops in the format X.X.X.X |
| Server Port Number | Integer | Integer within range 1 to 65535 (can be shortened to avoid collision with other programs) |
| Login Menu Selection | Char | Must be a character within a list in the program, potentially called “Valid\_Chars” |
| Username Selection (Registration) | String | Any string, the server itself decides whether or not it is a valid username based on its availability |

**Console Input**

|  |  |  |
| --- | --- | --- |
| Data Item | Data Type | Validation/Restrictions |
| Command (Word 1) | String | The first word in the user input must be present within a list of valid command, potentially called “Valid\_Commands” |
| Provided Argument (Words 2+) | String | All arguments provided after the first one must be valid according to the command provided, i.e., a valid NodeID should be provided after using the “getinfo” command – a command that returns the credentials of a given user from the server |

**Messenger Application**

|  |  |  |
| --- | --- | --- |
| Data Item | Data Type | Validation/Restrictions |
|  |  |  |

## 2.6.2 Page Navigation

Chart, box and whisker chart

Description automatically generated

[Proposed System Page Navigation Diagram] {Source: Self-made}

# 2.7 Security and Integrity

## 2.7.1 Addressing Issues Associated with The Current System

### The Central Server Problem

Sub-Problem 1: IP Address is public

Diagram

Description automatically generated

[Current System Data Transmission DFD: Level 0] {Source: Self-made}

For data transmissions to be considered secure in the above data-flow diagram, users must have trust in the central server. In the event the central server is compromised, all data belonging to users connected to the specified network is also compromised. For this we have the following solution – see below.

Diagram

Description automatically generated

[Proposed System Data Transmission DFD: Level 0] {Source: Self-made}

Rather than routing all traffic through a central server, the protocol can transmit data directly between systems; only relying on a central server to find the IP address. Therefore, if the central server is compromised, only the IP addresses of users on the network is compromised.

**The Anonymity Problem**

Sub-Problem 2: IP Address is known to sender

As the goal of this project is to create a standalone protocol that isn’t reliant on existing anonymity networks (i.e., Tor or VPN providers), the proposed system above where the sender is given the IP address of the recipient is invalid. However, routing all traffic through a central server is also problematic as the path the data takes can be traced by malicious adversaries in the event the server is compromised.

Instead, the proposed solution must be a combination of both alongside a key player – proxy servers.

Diagram

Description automatically generated

[Proposed System Data Transmission DFD: Level 0] {Source: Self-made}

In the above solution, rather than sending the data directly to the recipient, the data is sent to a proxy server acting on behalf of it. Through this, no one other than the server and the proxy knows the identity of the recipient. However, this can cause problems if the transmission is traced by an adversary with sufficient resources or if the proxy is compromised.

Sub-Problem 3: IP Address is known to proxy

The new issue is that now the central point of failure has been transferred to the proxy. In the event the transmission is traced to the proxy, the proxy can then be attacked – compromising the recipient’s identity.

As this project aims to remain as anonymous as possible to the user, it is not possible to allow the proxy to act as a point of failure. This is where a new technique comes in – onion routing.

Diagram, box and whisker chart

Description automatically generated

[Proposed System Data Transmission DFD: Level 0] {Source: Self-made}

Now, by routing traffic through a series of nodes in a ‘circuit’, systems only know the IP address of the previous and next nodes on the circuit (no one knows the true origin or destination of the message). In the event the message is traced, the recipient can send the message to a ‘dummy node’ so that it seems as though it isn’t the true recipient.

**The Security Problem**

Sub-Problem 4: Contents of Transmission known to Nodes

As Nodes along the circuit between the sender and recipient have access to the transmitted data, the final issue is the fact that they can read it.

The solution is to encrypt all private data using a combination of asymmetric and symmetric encryption so that only the recipient may have access to the contents of the data. To accomplish this, data unimportant to the transmission of the packet must be first encrypted symmetrically using an arbitrary, random key which is then encrypted with the recipient’s public key – the encrypted key is sent alongside the data. Upon receipt, the recipient decrypts the symmetric key using their own private key and uses the symmetric key to decrypt the private data.

**Additional Issue: Instant Messaging Latency Problem**

Sub-Problem 5: Routing packets through both circuits takes a large amount of time

Once again there is a new issue, messages take too long to travel between users – taking the instant out of instant messaging.

Diagram

Description automatically generated

[Proposed System Data Transmission DFD: Level 0] {Source: Self-made}

[3] Technical Solution

# 4.1 Pre-made Libraries Used

**socket**

Low-level network interface for python, allows the program to send and receive UDP data packets to communicate between systems.

UDP was used as opposed to TCP as I wanted to be able to program my own subroutines for actions such as replying to packets, as well as due to the fact that a single socket must be able to send and receive data from multiple systems.

**threading**

High-level threading interface that allows the concurrent and simultaneous running of multiple subroutines.

Used to handle and parse data packets simultaneously, as well as continuously check for incoming packets.

**time and datetime**

Grouping of various time-related functions, ranging from simply getting the UNIX timestamp from the computer’s time chip to converting 64-bit timestamps into user-readable dates.

Used to set the timestamp of every packet sent across the network as seconds since UNIX epoch (maximum efficiency), and to convert message timestamps into human readable times and dates.

**os**

“Miscellaneous operating system interfaces” (from Python documentation), allows the program to access various operating system functions such as listing directories.

Used to list the contents of the configuration and database folder in the client program.

**pickle**

Python object serialisation module. Converts python objects (i.e., arrays, strings, integers, tuples) into byte streams to allow sending over UDP and back.

Used to convert the packet arrays (layout [timestamp, header, destinationID, [msgID, circuitID, publicKey], payload, signature]) into a byte stream before sending using the sockets module over UDP.

**random**

Pseudo-random number generation module. Specifically referring to random.randint, the function generates a random integer between two numbers, x and y, inclusive.

Used to probabilistically select whether or not a given node becomes an end-node, as well as to generate MsgIDs and CircuitIDs.

Security Flaw: As in the description above, the numbers generated are only pseudo-random (they aren’t truly random). Since the value is determined upon its initial *seed* value, adversaries can mathematically figure out the number of nodes in a given circuit if they have access to the seed.

As mentioned in the documented design section for circuit generation, the expected length of any given circuit remains constant (via the Geometric Distribution), therefore should a given circuit have a length different from the expected length, E(X), it can become easily identifiable should the seed value be leaked.

**sqlite3**

From Python documentation: “provides a lightweight disk-based database that doesn’t require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language”.

SQLite3 allows the program to interface with external SQL database files and is used in the client files to store: online node information, “known” node information (similar to a friend list in a social media context), and instant messenger message archiving.

It is also used server-side to store the exit IP addresses of online nodes, register new users, login existing users, and save user credentials (such as Exit IP address, Public Keys, and Last Online timestamp).

**requests**.**get**

The Python requests library provides the ability to make HTTP requests to webservers. Referring specifically to requests.get, the module invokes the HTTP method ‘GET’ and allows the program to load external data from various webservers.

Used in the client program to retrieve the external public IP of the user by sending a GET request to “https://api.ipify.org”. If nothing is returned, the program assumes that there is no internet connection and will refuse to run, telling the user to connect before running the program.

# 4.2 Shared Core Files

**HGAlgorithms.py – Collection of various mathematical algorithms**

from random import randint

def primeNumberList(limit): #Sieve of Eratosthenes, prime number generation algorithm

    n = limit

    numList = [True] \* int(n + 1) #creates an array with all values set to true, length of array is specified as variable 'limit'

    for x in range(int(n\*\*0.5) - 2): #iterates through half the indexes in the list

        x = x + 2

        if numList[x] == True:

            y = 0

            z = 0

            while True: #using indefinite iteration, the algorithm calculates whether or not a number is prime by cycling through each multiplication combination

                z = (x + y) \* x

                if z < n:

                    numList[z] = False #if said number, z, has a multiple (and is therefore a non-prime) it is set to False

                    y += 1

                else:

                    #once the value of z exceeds that of n, exit the loop

                    break

    primeList = []

    for a in range(len(numList)): #all indexes in the list that have the value 'True' are appended to a list of prime numbers

        if numList[a] == True:

            primeList.append(a)

    primeList.remove(n)

    n = len(primeList)

    return (primeList)

primeList = primeNumberList(5000)

def primeNumberGen(limit): #selects a random prime number from the list

    global primeList

    if primeList == None:

        #if the global primeList has not been generated, generate a new one with limit 5000

        primeList = primeNumberList(5000)

    index = randint(3, limit - 1) #selects a random index

    return (primeList[index])

def greatestCommonDivisor(x, y): #function to return the greatest common divisor between two numbers, x and y

    if x > y: #as the function requires y to be larger than x, swap the two in case the order is opposite

        temp = y

        y = x

        x = temp

    if x == 0: #uses recursion to calculate the modulo of y against x until a result is found (when x equals zero)

        return y, 0, 1

    else:

        result, xNew, yNew = greatestCommonDivisor(y % x, x)

        return result, yNew - (y // x) \* xNew, xNew

def modularInverse(x, y): #modular inverse calculation

    try:

        result = pow(x, -1, y)

        #python pow function returns the power of a number

    except:

        result = ord("#")

        #in some cases, certain numbers do not have a modular inverse

        #in this circumstance, simply return the ordinal value of character '#'

        #failed characters will show up like the following: "He##o, Wor#d!"

    return result

if \_\_name\_\_ == '\_\_main\_\_':

    print("This module must be run from within the file 'client.py'.")

**HGCrypto.py – Collection of cryptographic functions and algorithms**

### SELF-MADE MODULES ###

import HGTesting as Testing

import HGAlgorithms as Algorithms

###END###

primeList = Algorithms.primeNumberList(1000)

#used for testing purposes, generates global prime number list when opened

def hash(data): #simple hash algorithm

    try:

        data = data.decode("utf-16")

        #unless data is given directly as a numerical value:

        #decode the data string into 16-bit Unicode Transmission Format

    except:

        pass

    total = 0

    for x in range(len(data)):

        try:

            total = total + (ord(str(data[x])) \* x)

            #for each character in the data string, convert the character to its ordinal value

            #and multiply it by the value of x

            #add this value to the total

        except:

            total = total + x

            #if the character cannot be multiplied, or it doesn't have an ordinal value

            #simply add the value of x to the total

    return total

def sign(data, privateKey):

    #similar to asymmetric encryption, hashing uses the user's own private key

    #rather than a particular recipient's public key

    n, d = privateKey

    hashVal = hash(data) #convert the data to be signed into a hash

    sig = pow(hashVal, d, n) #the signature is (hashval to the power of d) modulus n

    return sig

def verify(data, sig, publicKey):

    #process is the same as above except the corresponding public key is used to change the hash back to its original form

    n, e = publicKey

    hashVal = hash(data)

    checkSig = pow(sig, e, n)

    if hashVal == checkSig:

        #calculated signature value matches the given hash, therefore the data is verified

        return True

    else:

        #verification failed, returns false

        return False

def asymKeyGen(length):

    while True:

        while True: #prime number selection loop

            prime1 = Algorithms.primeNumberGen(length)

            prime2 = Algorithms.primeNumberGen(length)

            if (prime1 != prime2) and (prime1 != 1) and (prime2 != 1):

                break

        n = prime1 \* prime2

        phiN = (prime1 - 1) \* (prime2 - 1) #Euler's Totient Function

        lambdaN = int(phiN/Algorithms.greatestCommonDivisor(prime1 - 1, prime2 - 2)[0]) #Carmichael's Lambda Function

        first = True

        index = 2

        e = primeList[index]

        while (first == True) or (Algorithms.greatestCommonDivisor(n, e)[0] != 1) or (Algorithms.greatestCommonDivisor(lambdaN, e)[0] != 1):

            #exponent picker

            first = False

            e = primeList[index]

            index += 1

        publicKey = (n, e)

        d = Algorithms.modularInverse(e, lambdaN)

        privateKey = (n, d)

        if d < 1:

            pass

        elif e != d:

            if Testing.asymmetricEncryption(publicKey, privateKey) == False:

                break

            else:

                pass

    return publicKey, privateKey

def asymEncrypt(data, PublicKey):

    n, e = PublicKey

    data = str(data)

    cipherData = ""

    for x in range(len(data)):

        #iterates through every character in the given string

        #changes the character to its decimal (ordinal) value in utf or ascii

        #encryption operation is then applied on the value

        #value is then re-converted into its character form and appended to the end of the result

        cipherData = cipherData + chr(Algorithms.modularInverse(ord(data[x]) \*\* e, n))

    return cipherData

def asymDecrypt(cipherData, PublicKey, PrivateKey):

    n1, e = PublicKey

    n2, d = PrivateKey

    if n1 == n2:

        n = n1

    else:

        print("Error: Keypair does not match!")

        return -1

    data = ""

    for x in range(len(cipherData)):

        #similar to asymEncrypt, except the decryption operation is applied on the character value

        data = data + str(chr(Algorithms.modularInverse(ord(cipherData[x]) \*\* d, n)))

    return data

if \_\_name\_\_ == '\_\_main\_\_':

    print("This module must be run from within the file 'client.py'.")

**HGHelp.py – Command-line outputs for user help queries**

def main(): #general help menu

    print("""

    Help Menu

    -=-=-=-=-

        Use any command, followed by "-h" or "--help" to bring up usage for the specified command

    Command             Purpose

    =======             =======

    chat/messenger      Start Messenger Application

    exit/quit           Exit the application

    getinfo             Add a user to your list of friends

    help                View the Help menu

    show                View stored data (i.e., Connected Users, Known Users), use 'show -h' for more details

    whoami              Return the name of the current user

    """)

def show(): #help menu for command: 'show'

    print("""

    Usage: show [option] {optional: target}

    Option              Purpose

    ======              =======

    online              Show list of online users in known list

    known               Show list of known users

    """)

def messenger(): #help menu for command: 'messenger'

    print("""

    Usage: messenger/chat [NodeID]

    Option              Purpose

    ======              =======

    NodeID              Enter the NodeID of the user you wish to chat with

    """)

def getinfo(): #help menu for command: 'getinfo'

    print("""

    Usage: getinfo [NodeID]

    Option              Purpose

    ======              =======

    NodeID              Enter the NodeID of the user you wish to add

    """)

def messenger\_main(): #help menu for messenger command: 'help'

    print("""

    Help Menu

    -=-=-=-=-

        To use any command specified below, precede input with the '/' character

    Command             Purpose

    =======             =======

    exit/quit           Exit the messenger

    help                View the Help menu

    ping                Check whether or not the recipient is online (Not Implemented in Final Version)

    whoami              Return the name of the current user

    """)

if \_\_name\_\_ == '\_\_main\_\_':

    print("This module must be run from within the file 'client.py'.")

**HGMessenger.py – Hourglass Messenger Application**

import time, datetime, HGHelp, threading, pickle, random

import HGCrypto as Cryptography

#HGHelp and HGCrypto are self-made modules

base64Chars = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z",

                "a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z",

                "0","1","2","3","4","5","6","7","8","9","+","/","="] #all valid base64 characters, '=' is padding

class Messenger():

    def \_\_init\_\_(self, username, publicKey, destination, messageList):

        self.\_\_lastDate = '' #last date message was sent

        self.\_\_username = username #user's own name

        self.\_\_publicKey = publicKey #user's own public key

        self.\_\_destination = destination #destination NodeID

        self.\_\_messageList = messageList #array with previous messages, retrieved from SQL table in client program

        self.\_\_restoreMessages() #run message restoration subroutine (see below)

        self.\_\_nextPacket = None

        self.quitStatus = False

        print("""

 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|       WELCOME TO THE HOURGLASS MESSENGER        |

|-=-=-=-=-=-=-=-=-=-=-=-=-=-=--=-=-=-=-=-=-=-=-=-=|

|Precede input with a '/' to use commands.        |

|Use '/help' to access a list of useable commands.|

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

        """)

        self.inputThread = threading.Thread(target = self.\_\_getInput)

        #starts the input getter as a separate thread so that the messenger can print incoming messages without waiting for user input first

        self.inputThread.start()

    def \_\_getInput(self):

        while self.quitStatus == False:

            #userInput = input('\033[1A' + f"|To {self.\_\_destination}| > " + '\033[K')

            userInput = input("")

            if userInput == "":

                pass

            elif userInput[0] == "/":

                userInput = userInput[1 : : ]

                #python string editing syntax above removes the initial '/' character from the input string

                self.\_\_command(userInput)

            else:

                self.\_\_sendMessage(userInput)

    def \_\_sendMessage(self, message):

        timestamp = time.time()

        self.printMessage(timestamp, message, 1)

        msg = [timestamp, "MSG", Cryptography.asymEncrypt(self.\_\_username, self.\_\_publicKey), Cryptography.asymEncrypt(message, self.\_\_publicKey)]

        msg = pickle.dumps(msg)

        self.\_\_nextPacket = msg

        #since the messenger class has no access to the functions of the client program's websockets

        #store the next message to be sent in self.\_\_nextPacket

        #the client program iteratively checks the contents of the aforementioned variable to check what to send

        #uses function below

    def getNextPacket(self): #returns the contents of self.\_\_nextPacket

        temp = self.\_\_nextPacket

        self.\_\_nextPacket = None

        #once the packet has been accessed by the client program

        #the value is changed to none so that the same message isn't sent twice

        return temp

    def \_\_command(self, command):

        print('\033[1A' + f"Running command: {command}" + '\033[K')

        #print(f"Running command: {command}")

        if (command == "quit") or (command == "exit"):

            self.\_\_quit()

        elif command == "help":

            HGHelp.messenger\_main()

        elif command == "whoami":

            print(self.\_\_username)

    def getMessageList(self):

        return self.\_\_messageList

    def printMessage(self, timestamp, msg, own, external=False):

        self.\_\_messageList.append([timestamp, own, msg])

        msgTime = datetime.datetime.utcfromtimestamp(int(timestamp)).strftime('%H:%M:%S')

        #check if the message was sent by the user

        if own == 1:

            sender = self.\_\_username

        else:

            sender = self.\_\_destination

        if external == True:

            #print('\033[1A' + f"[{msgTime}] {sender}| {msg}"+ '\033[K', end='\n')

            print(f"[{msgTime}] {sender}| {msg}", end='\n')

            #print(f"|To {self.\_\_destination}| > ")

            print("")

        else:

            print(f"[{msgTime}] {sender}| {msg}", end='\n')

            print("")

            #print('\033[1A' + f"[{msgTime}] {sender}| {msg}"+ '\033[K',  end='\n')

    def \_\_restoreMessages(self): #restore all previous messages with this user

        currentDate = ''

        if self.\_\_messageList != []:

            for x in range(len(self.\_\_messageList)):

                timestamp = self.\_\_messageList[x][0]

                currentTime = datetime.datetime.utcfromtimestamp(int(timestamp)).strftime('%H:%M:%S')

                #current time stores the timestamp of the current message being parsed in a human-readable format (HH:MM:SS)

                ownCheck = self.\_\_messageList[x][1]

                #check if the message was sent by the user

                if ownCheck == 1: #integer 1 is used since SQL cannot store direct boolean values

                    sender = self.\_\_username #set the sender to our username

                else:

                    sender = self.\_\_destination #set the sender to the destination username

                message = self.\_\_messageList[x][2]

                if currentDate != datetime.datetime.utcfromtimestamp(int(timestamp)).strftime('%A %d %B %Y'):

                    #if the data of the message is different to the previous one:

                    #print a new date header

                    print("")

                    currentDate = datetime.datetime.utcfromtimestamp(int(timestamp)).strftime('%A %d %B %Y')

                    print(f"|<==============>|{currentDate}|<==============>|")

                print(f"[{currentTime}] {sender}| {message}")

        self.\_\_lastDate = currentDate

        if self.\_\_lastDate != datetime.datetime.utcfromtimestamp(time.time()).strftime('%A %d %B %Y'):

            #if the last date processed from the message list doesn't match today's date:

            #print todays date

            #doesn't run if the date matches, in order to avoid printing the same date twice

            print("")

            self.\_\_lastDate = datetime.datetime.utcfromtimestamp(time.time()).strftime('%A %d %B %Y')

            print(f"|<==============>|{self.\_\_lastDate}|<==============>|")

        self.\_\_clearMessages()

    def \_\_clearMessages(self): #clear all messages from message list

        self.\_\_messageList = []

    def \_\_quit(self): #raises quit flag

        self.quitStatus = True

if \_\_name\_\_ == '\_\_main\_\_':

    print("This module must be run from within the file 'client.py'.")

**HGTesting.py – Various testing and input validation scripts**

import HGCrypto as Cryptography #self-made module

def asymmetricEncryption(publicKey, privateKey):

    fail = False

    TEST1 = "abcdefghijklmnopqrstuvwxyz" #alphabetical character test (uppercase not required as their character code is lower than lowercase in UTF and ASCII)

    TEST2 = "1234567890" #numerical character test

    TEST3 = "!£$%^&\*()-\_=+}{[];:''@#~,<>./?" #symbol test

    if TEST1 != Cryptography.asymDecrypt(Cryptography.asymEncrypt(TEST1, publicKey), publicKey, privateKey):

        #encrypts and decrypts test string in the same line

        fail = True

    if TEST2 != Cryptography.asymDecrypt(Cryptography.asymEncrypt(TEST2, publicKey), publicKey, privateKey):

        fail = True

    if TEST3 != Cryptography.asymDecrypt(Cryptography.asymEncrypt(TEST3, publicKey), publicKey, privateKey):

        fail = True

    #if any of the tests above fail:

    #the function returns True, and the key generation is run again

    return fail

def IPValidation(IP, PORT):

    try: #try to change port (potentially in string form) to integer

        PORT = int(PORT)

    except: #invalid port value error, the port provided has an non-numerical character within it (and cannot be converted into an integer)

        return False

    section = 0

    separatedIP = [""] \* 4

    for x in range(len(IP)):

        if IP[x] == ".": #every time a dot is encountered, the algorithm begins a new section

            section += 1

        else:

            separatedIP[section] = separatedIP[section] + IP[x]

    try:

        for x in range(len(separatedIP)):

            separatedIP[x] = int(separatedIP[x]) #convert all sections of the separated IP address from string to integer form

            if (separatedIP[x] > 255) or (separatedIP[x] < 0): #checks if value within IP exceeds limit

                return False

        return True #if all checks are passed, return True (address and port are valid)

    except: #invalid character in IP address

        return False

if \_\_name\_\_ == '\_\_main\_\_':

    print("This module must be run from within the file 'client.py'.")

**structures.py – Collection of dynamic data structures**

class Queue(): #altered queue data structure, 'hybrid' - can be set as either a priority or non-priority queue on intialisation using priorityState flag

    def \_\_init\_\_(self, size, priorityState = False):

        self.main = [None] \* int(size) #sets up empty array with size specified

        self.front = 0 #front pointer

        self.back = 0 #back pointer

        self.limit = size - 1 #limit of array index

        self.priorityState = priorityState #whether or not the queue is a priority queue

        if self.priorityState == True:

            self.priority = [None] \* int(size) #sets up parallel empty array

        #if priority flag is set to true upon instantiation:

        #the queue runs as a priority queue, thus being a 'hybrid' queue

    def peek(self): #returns front value without removal

        return self.main[self.front]

    def dequeue(self): #returns front value, removing it from the queue

        if self.isEmpty() == False:

            result = self.main[self.front]

            self.main[self.front] = None

            if self.priorityState == True: #removes from priority array if it is a priority queue

                self.priority[self.front] = None

            if self.front == self.back: #position of the pointers do not change if they are the same

                pass

            else:

                self.front += 1 #increments the pointer by one

            return result

        else:

            print("ERROR: Queue is empty")

            return None

    def enqueue(self, item, priorityVal = None):

        if self.isFull() == True: #shows error if the queue is already full

            print("ERROR: Queue is full")

        else:

            if self.priorityState == False: #code executed if it is a non-priority queue

                self.main[self.back] = item

                self.back += 1

            else:

                if priorityVal == None:

                    print("ERROR: Priority unspecified")

                else:

                    if self.isEmpty() == True:

                        self.main[self.back] = item

                        self.priority[self.back] = priorityVal

                        self.back += 1

                    else:

                        last = False

                        for x in range(len(self.priority)): #iterates until the item reaches the end of the queue, or an item which has a priority less than it

                            if x == self.limit:

                                index = x

                                last = True

                                break

                            elif self.priority[x] == None:

                                pass

                            elif self.priority[x] > priorityVal:

                                index = x

                                break

                        if last == True: #as it is the last one, it is not required to change the positions of the values behind it

                            self.main[self.back] = item

                            self.priority[self.back] = priorityVal

                            self.back += 1

                        else:

                            currentItem = item

                            currentPriority = priorityVal

                            for y in range(index, self.limit + 1): #shifts all items of lesser priority down the queue

                                if currentItem == None:

                                    break

                                tempItem = self.main[y]

                                tempPriority = self.priority[y]

                                self.main[y] = currentItem

                                self.priority[y] = currentPriority

                                currentItem = tempItem

                                currentPriority = tempPriority

                            self.back += 1

    def getSize(self): #returns size of queue

        size = self.limit + 1

        for x in range(len(self.main)):

            if self.main[x] == None:

                size = size - 1

        return size

    def isEmpty(self): #returns whether or not a queue is empty

        if self.getSize() == 0:

            return True

        else:

            return False

    def isFull(self): #returns whether or not a queue is full

        if self.back == (self.limit + 1):

            return True

        else:

            return False

class LimitlessQueue(): #altered version of the classic queue, it doesn't have a limit and all items in the queue move foreward when dequeued (therefore not relying on a back or front pointer)

    def \_\_init\_\_(self, priorityState = None):

        self.main = []

        self.priorityState = None

        if self.priorityState == True:

            self.priority = []

    def peek(self): #returns front value without removal

        return self.main[0]

    def enqueue(self, item):

        self.main.append(item)

    def dequeue(self): #returns front value, removing it from the queue

        if self.isEmpty() == False: #nothing to pop is queue is empty

            print("ERROR: Queue is empty")

            return None

        else:

            result = self.main[0]

            self.main.pop(0)

            return result

    def getSize(self): #returns size of the queue

        return len(self.main)

    def isEmpty(self): #checks if the queue is empty or not (no need to check if full as it is limitless) {queue is limited to the python array limit; so not 'truly' limitless, but doesn't have a 'set' limit}

        if len(self.getsize()) == 0:

            return True

        else:

            return False

if \_\_name\_\_ == '\_\_main\_\_':

    print("This module must be run from within the file 'client.py'.")

# 4.3 Client File

import socket, threading, time, os, pickle, random, sqlite3

from threading import Thread

from logging import exception

from requests import get

###OWN MODULES###

import HGHelp as Help

from structures import Queue

from structures import LimitlessQueue

import HGCrypto as Cryptography

import HGTesting as Testing

from HGMessenger import Messenger

###END###

###SERVER DEFAULTS###

serverIP = ("192.168.0.1", 107) #Default address of the specified data server until the global config is setup

serverName = "EMPTY" #Default name of the specified data server until the global config is setup

serverOnline = False #indicates whether or not the specified server is online, found by sending a test packet and waiting for a reply with the server's name

#serverName is chosen by the server itself rather than the user

serverAnnouncements = ''

serverKey = (0, 0)

###END###

#DEFAULT PROGRAM FLAGS

loginStatus = False #Whether or not the user is logged into the network

registerStatus = False #Whether or not a registration request was successfull

packetFailure = False #Whether or not a packet has failed to send

myPublicKey = () #sets own public/private key as empty until user is configured

myPrivateKey = ()

ProgramQuit = False

hexChars = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F"] #all valid hexadecimal characters

base64Chars = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z",

                "a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z",

                "0","1","2","3","4","5","6","7","8","9","+","/","="] #all valid base64 characters, '=' is padding

users = []

configFiles = []

username = ""

primeList = None

nonOwnCircuits = [] #circuits that this instance has no control of

ownCircuit = None #same as above but with control

ongoingPackets = [] #packets that are awaiting replies

joinableNodes = []

TIMEOUT = 120 #two minute timeout

RANDOM\_CONST = 3

choicesIP = []

try:

    tempSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

    tempSock.connect(("10.255.255.255", 80)) #arbitrary IP address used to get local IP address of network rather than loopback

    choicesIP.append(tempSock.getsockname()[0])

    tempSock.close()

except:

    print("ERROR: Not connected to any networks!")

    ProgramQuit = True

choicesIP.append(socket.gethostbyname(socket.gethostname()))

messenger = None

ownCircuit = None

temp\_pubKey, temp\_CircuitID, temp\_ExitIP = None, None, None

"""

Packet Layouts:

##General Layout

    [timestamp, header, destinationID, [msgID, circuitID, publicKey], payload, signature]

    payload = encrypted byte(pickle) dump

    payload(unencrypted, non-byte form) = [sender, data] #where sender is the nodeID of the sender

    data = ["APP\_TYPE", APP\_DATA] #example: ["MSG", message]

##To Server

    via Circuit: [timestamp, header, "SERVER", [msgID, circuitID, publicKey], payload, signature] #login, register, nodeInfo

    direct: [timestamp, header] #getNodes & test

    login:

        [timestamp, "LOGIN", "SERVER", [msgID, circuitID, publicKey], login\_payload, signature]

        login\_payload(unencrypted) = NodeID

            Enter timestamp, circuitID and endIP to online database if signature is verified

    register:

        [timestamp, "REGISTER", "SERVER", [msgID, circuitID, publicKey], register\_payload, signature]

        register\_payload(unencrypted) = NodeID

            Check if NodeID already exists in Database

            If it exists: reply with existence error

            Else:

                Verify signature

                Add timestamp, circuitID, publicKey, endIP, and NodeID to database (both online and known)

    nodeInfo:

        [timestamp, "NODEINFO", "SERVER", [msgID, circuitID, publicKey], info\_payload, signature]

        info\_payload(unencrypted) = NodeID(to be searched)

        Verify Signature and return data

            REPLY:

                [timestamp, "DATA", circuitID, [msgID, "SERVER", publicKey], return\_payload, signature]

##To Another User

    data:

        [timestamp, "DATA", destinationID, [msgID, circuitID, publicKey], payload, signature]

        payload = encrypted byte(pickle) dump

        payload(unencrypted, non-byte form) = [sender, data] #where sender is the nodeID of the sender

        data = ["APP\_TYPE", APP\_DATA] #example: ["MSG", message]

    reply:

        [timestamp, "REPLY:msgID", recipientIP]

        if recipientIP matches own IP: Pop MsgID from ongoingPackets

    request:

        [timestamp, "REQUEST:type" ---] #different depending on case

        [timestamp, "REQUEST:JOINCIRCUIT", circuitID, publicKey, signature]

        Verify signature using publicKey

            Reply:

            [timestamp, "REPLY:circuitID", recipientIP]

"""

try:

    publicIP = get("https://api.ipify.org").text

    inetStatus = True

except:

    print("ERROR: Not connected to Internet")

    publicIP = "Not connected to Internet"

    inetStatus = False

valid = ["c", "C", "n", "N", "q", "Q", "t", "T"]

class Interface():

    def bootstrap():

        global sock, currentIP

        #prints introductory ASCII art, may look wrong in document

        print("""

 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@             \_\_\_\_\_ \_

  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@             |\_   \_| |

     @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                  | | | |\_\_   \_\_\_

     @                             @                  | | | '\_ \ / \_ \

     @                             @                  | | | | | |  \_\_/

     @                             @                  \\_/ |\_| |\_|\\_\_\_|

     @@                           @@      \_\_    \_\_                                          \_\_

      @@  @@@@@@@@@@@@@@@@@@@@@  @@     /  |  /  |                                        /  |

        @   @@@@@@@@@@@@@@@@@   @       ██ |  ██ |  \_\_\_\_\_\_   \_\_    \_\_   \_\_\_\_\_\_    \_\_\_\_\_\_  ██ |  \_\_\_\_\_\_    \_\_\_\_\_\_\_  \_\_\_\_\_\_\_

          @@   @@@@@@@@@@@   @@         ██ |\_\_██ | /      \ /  |  /  | /      \  /      \ ██ | /      \  /       |/       |

             @@   @@@@@   @@            ██    ██ |/██████  |██ |  ██ |/██████  |/██████  |██ | ██████  |/███████//███████/

                 @  @  @                ████████ |██ |  ██ |██ |  ██ |██ |  ██/ ██ |  ██ |██ | /    ██ |██      \██      \

                 @@   @@                ██ |  ██ |██ \\_\_██ |██ \\_\_██ |██ |      ██ \\_\_██ |██ |/███████ | ██████  |██████  |

                @@     @@               ██ |  ██ |██    ██/ ██    ██/ ██ |      ██    ██ |██ |██    ██ |/     ██//     ██/

             @@           @@            ██/   ██/  ██████/   ██████/  ██/        ███████ |██/  ███████/ ███████/ ███████/

          @@                 @@                                                 /  \\_\_██ |

      @@                         @@                                             ██    ██/

     @@             @             @@                                             ██████/

     @        @@@@@@@@@@@@@        @                    \_\_\_\_\_\_          \_                  \_

     @   @@@@@@@@@@@@@@@@@@@@@@@   @                    | \_\_\_ \        | |                | |

     @  @@@@@@@@@@@@@@@@@@@@@@@@@  @                    | |\_/ / \_\_ \_\_\_ | |\_ \_\_\_   \_\_\_ \_\_\_ | |

     @                             @                    |  \_\_/ '\_\_/ \_ \| \_\_/ \_ \ / \_\_/ \_ \| |

     @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                    | |  | | | (\_) | || (\_) | (\_| (\_) | |

  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                 \\_|  |\_|  \\_\_\_/ \\_\_\\_\_\_/ \\_\_\_\\_\_\_/|\_|

 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

    """)

        validChoices = []

        sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) #configures socket as UDP socket

        print("Network Interface Selection")

        print("-=-=-=-=-=-=-=-=-=-=-=-=-=-")

        print("Your Public IP Address is:", publicIP)

        print("-=-=-=-=-=-=-=-=-=-=-=-=-=-")

        for x in range(len(choicesIP)):

            validChoices.append(str(x + 1))

            print(str(x + 1) + ". " + choicesIP[x])

        while True:

            choice = input("Please select the network interface you wish to use: ")

            if choice in validChoices:

                while True:

                    portChoice = input("Please enter the port you wish to use [Default: 100]: ")

                    try:

                        if portChoice == '':

                            currentIP = (choicesIP[int(choice) - 1], 100)

                            sock.bind(currentIP) #binds the specified IP address as the IP of the packet

                            break

                        else:

                            portChoice = int(portChoice)

                            currentIP = (choicesIP[int(choice) - 1], portChoice)

                            sock.bind(currentIP) #binds the specified IP address as the IP of the packet

                            break

                    except:

                        print("ERROR: Invalid Port Number")

                threading.Thread(target = Transmission.receiveAgent).start() #starts independent thread to check for incoming packets

                break

            else:

                print("ERROR: Invalid Choice")

        print("-=-=-=-=-=-=-=-=-=-=-=-=-=-")

        print("")

        initAnim() #initialises loading animation

        loadAnim(20) #value '20' indicates that the loading bar is to go through at once

        print("") #print newline

        Interface.login() #login subroutine

    def login():

        global serverName, serverIP, serverOnline, ProgramQuit, users, configFiles

        Config.globalConfig() #run global configuration subroutine

        users = os.listdir('./config/') #the users that can access the protocol are the filenames in the ./config directory

        configFiles = users #saves list of files to global variable

        for x in range(len(users)): #removes filename extension from names of users '.hg', all config files end in the same extensions (inclusing global config) to avoid inter-program conflict

            users[x] = (users[x])[:-3]

        print("""

USER SELECT:

\_\_\_\_\_\_\_\_\_\_\_\_

        """)

        if users == []: #if no users are in the config directory

            print("No Users Registered")

        else:

            for x in range(len(users)): #prints selection menu for all users in the config directory

                lognum = str(x + 1) + "."

                print(lognum, users[x])

                valid.append(str(x + 1)) # adds given selections to the valid options array

        print("---------------------------")

        if serverIP[0] == "EMPTY": #default value for server IP

            print("Server: Not Configured")

        else: #prints the value saved in the global config file

            print(f"Server: {serverName} @ {serverIP[0]}/{serverIP[1]}")

        if serverAnnouncements != '':

            print("Announcements:", serverAnnouncements)

        if serverOnline == True: #the server is set as online if the protocol can successfully send a test packet to the server (and it replies)

            status = "Online"

        else:

            status = "Offline"

        print("Status:", status)

        print(f"Current IP: {currentIP[0]}/{currentIP[1]}")

        print("---------------------------")

        print("C. Configure Central Server")

        print("T. Test Connection")

        print("N. New User")

        print("Q. Quit\n")

        while True:

            choice = input("Please select an option: ")

            if choice in valid: #unless the user option is in the valid options array, the program will display an error

                break

            else:

                inputUnrec()

        if (choice == "q") or (choice == "Q"): #quits program

            ProgramQuit = True #quits program (pass)

        elif (choice == "t") or (choice == "T"): #tests connection to server manually, occurs at program bootstrap

            Transmission.testServer()

            Interface.login()

        elif (choice == "c") or (choice == "C"): #configures the IP Address and port of the central data server

            IP = input("Enter IP Address of Server: ")

            PORT = input("Enter Port of Server: ")

            status = Testing.IPValidation(IP, PORT)

            if status == True:

                serverIP = (IP, int(PORT))

            else:

                print("ERROR: Incorrect format for IP Address")

            Config.updateGlobalConfig()

            Interface.login()

        elif (choice == "n") or (choice == "N"): #configures a new user

            if serverOnline == False:

                Transmission.testServer() #tests connection to server, receives server name and announcements

            if serverOnline == True:

                Config.newConfig() #if the server is online after test, go ahead with registration

            else: #else quit program

                print("Server is offline, quitting...")

                ProgramQuit = True

        else:

            if serverOnline == False:

                #if the server isn't shown to be online, send a test packet to the server

                Transmission.testServer()

            if serverOnline == True:

                Config.login(choice) #logs into the specified user

            else:

                print("Server is offline, quitting...")

                ProgramQuit = True

    def console(): #console code

        global ProgramQuit

        while (ProgramQuit == False): #iterates until the quit flag is put up

            consoleInput = input(f"|{username}| > ") #outputs the console and takes in user input

            Interface.parser(consoleInput) #sends input to parser

    def parser(input):

        #input parsing subroutine

        parsedIn = []

        curWord = ""

        index = 0

        spaces = 0

        for x in range(len(input)): #separates commands and arguments

            if input[x] == " ":

                spaces += 1 #computes the number of spaces in the given input string

        words = spaces + 1 #using the laws of (most) languages that use the latin alphabet, the number of words in a sentence will always be one more than the spaces

        parsedIn = [None] \* words #assign Nonetype to all indexes of parsed input array

        for x in range(len(input)): #separates words and adds them to the aforementioned array

            if x == (len(input) - 1):

                curWord = curWord + str(input[x])

                parsedIn[index] = curWord

                index += 1

                curWord = ""

            elif input[x] != " ":

                curWord = curWord + str(input[x])

            else:

                parsedIn[index] = curWord

                index += 1

                curWord = ""

        Interface.choicer(parsedIn) #sends parsed input to command choice selection subroutine

    def choicer(input):

        #parsed input argument execution subroutine

        global ProgramQuit, messenger

        main = input[0]

        if (main == "exit") or (main == "quit"): #executes specified subroutine for chosen command

            ProgramQuit = True #program quit flag

        elif main == None:

            pass

        elif main == "show":

            try: #argument selection

                if input[1] == "known":

                    try:

                        print(database.showKnown(input[2]))

                    except:

                        print(database.showKnown())

                elif input[1] == "-h" or input[1] == "--help":

                    raise

                elif input[1] == "online":

                    try:

                        print(database.showOnline(input[2]))

                    except:

                        print(database.showOnline())

                else:

                    inputUnrec()

            except: #activates help menu for specified command

                Help.show()

        elif main == "help":

            Help.main()

        elif (main == "messenger") or (main == "chat"):

            try:

                NodeID = input[1]

                if NodeID != username: #validation for messenger app before initialisation

                    info = database.showOnline(NodeID) #assign information array from result of function "database.showOnline" to local variable "info"

                    if info != []: #if the array is not empty, it means that the selected user has information about them stored locally, and we can connect to them

                        CircuitID = info[0][0]

                        ipAddr = (info[0][1], info[0][2])

                        info = database.showKnown(NodeID)

                        key = (info[0][1], info[0][2])

                        messageList = database.getMessages(NodeID) #retrieve messages from SQL table 'messages' and assign to python array 'messageList'

                        messenger = Messenger(username, key, NodeID, messageList)

                        ###SPECIAL MESSENGER CODE###

                        #Since the messenger class has no access to the websockets belonging to the parent program, the parent program uses iteration to check whether there are new packets to be sent using "messenger.getNextPacket"

                        while messenger.quitStatus == False:

                            time.sleep(0.5)

                            nextPacket = messenger.getNextPacket()

                            if nextPacket != None: #default value of nextPacket is None. If it isn't, it means that there is a new packet to be sent

                                Transmission.send(nextPacket, ipAddr) #send the packet to the destination NodeID

                        ###END###

                        #Code reaches here when the messenger program uses it's "quit" function, and messenger.quitStatus equals True - allowing the above loop to end

                        database.writeMessages(NodeID, messenger.getMessageList()) #write all messages to SQL database 'messages' using the destination NodeID as a Foreign Key

                    else:

                        print("ERROR: Either user isn't online or you have not used 'getinfo' first!")

                else:

                    print("ERROR: You cannot chat with yourself!")

            except:

                Help.messenger()

        elif main == "whoami":

            print(username)

        elif main == "getinfo":

            try:

                selectID = input[1]

                ownCircuit.nodeInfo(selectID)

            except:

                Help.getinfo()

        else: #unrecognised input

            inputUnrec()

class Config():

    def userConfig(): #configures settings for specified user, specifically their database (which contains the friends list) and config file (which contains their public key, locked using AES)

        global database, serverIP, serverName, myPublicKey, myPrivateKey

        database = Database()

        lines = []

        userFile = open("./config/" + str(username) + ".hg", "r")

        for x in userFile.readlines():

            lines.append(x.strip()) #remove new line at end of each string

        #elements of the keypair stored as lines in a text file

        #text file was chosen due to its human-readable nature

        n = int(lines[1])

        e = int(lines[3])

        d = int(lines[5])

        myPublicKey = (n, e)

        myPrivateKey = (n, d)

    def newConfig():

        global ownCircuit, username, myPublicKey, myPrivateKey, database

        print("Generating Keypair...")

        myPublicKey, myPrivateKey = Cryptography.asymKeyGen(100) #generate new keypair using HGCrypto file

        ownCircuit = OwnCircuit() #instantiate OwnCircuit object as "ownCircuit"

        ownCircuit.buildCircuit()

        if ProgramQuit != True:

            while True:

                username = input("Enter the name of the new user: ")

                ownCircuit.register()

                if registerStatus == True:

                    break

                elif ProgramQuit == True:

                    print("Exit flag raised: Quitting.")

                    return

                else:

                    print("Registration Failed: Try New Username")

            try: #Try creating a global file, redirects to read if exists

                userFile = open("./config/" + str(username) + ".hg", "x") #creates the specified file

                userFile.close() #closes the file to enable access to writing/appending mode

                userFile = open("./config/" + str(username) + ".hg", "w")

                lines = ["[N]", str(myPublicKey[0]), "[PUBLIC KEY E]", str(myPublicKey[1]), "[PRIVATE KEY D]", str(myPrivateKey[1])]

                for x in range(len(lines)):

                    userFile.writelines(lines[x] + "\n")

                userFile.close()

                database = Database()

                Interface.console()

            except:

                print("ERROR: User already exists")

    def updateUserConfig(): #write saved variable values to user config file

        userFile = open("./config/" + str(username) + ".hg", "w")

        lines = ["[N]", str(myPublicKey[0]), "[PUBLIC KEY E]", str(myPublicKey[1]), "[PRIVATE KEY D]", str(myPrivateKey[1])]

        for x in range(len(lines)):

            userFile.writelines(lines[x] + "\n")

    def globalConfig(): #subroutine for accessing data in the global config file 'Global.hg'

        global globalFile, serverIP, serverName

        try: #Try creating a global file, redirects to read if exists

            globalFile = open("Global.hg", "x") #creates the specified file

            globalFile.close() #closes the file to enable access to writing/appending mode

            Config.updateGlobalConfig()

        except: #Read global file

            lines = []

            globalFile = open("Global.hg", "r")

            for x in globalFile.readlines():

                lines.append(x.strip()) #remove new line at end of each string

            serverIP = (lines[1], int(lines[3]))

            serverName = str(lines[5])

    def updateGlobalConfig(): #creates or updates the global config file if data is altered in the program

        globalFile = open("Global.hg", "w")

        lines = ["[IP]", serverIP[0], "[PORT]", str(serverIP[1]), "[SERVER NAME]", serverName]

        for x in range(len(lines)):

            globalFile.writelines(lines[x] + "\n")

    def login(user):

        global username, ownCircuit, ProgramQuit, loginStatus

        #config all from config file

        username = users[int(user) - 1]

        Config.userConfig()

        ownCircuit = OwnCircuit() #instantiate OwnCircuit object as "ownCircuit"

        ownCircuit.buildCircuit()

        if ProgramQuit != True: #if the quit flag has been raised elsewhere in the program do not run the login subroutine

            ownCircuit.login()

        if loginStatus == False: #if after running the login subroutine the server returns a failed login request, quit the program

            print("Credentials do not match that of the server! Quitting.")

            ProgramQuit = True

        Interface.console()

class Database(): #collection of database-oriented subroutines and functions - initialised when program is run as 'database' (lowercase)

    def \_\_init\_\_(self):

        databases = os.listdir('./databases/')

        for x in range(len(databases)):

            databases[x] = (databases[x])[:-3]

        if username not in databases:

            print("Database doesn't exist for specified user. Creating database.")

        self.connection = sqlite3.connect('./databases/' + str(username) + '.db', check\_same\_thread=False)

        self.cursor = self.connection.cursor()

        #initialise the various tables (known, online and messages)

        self.knownInit()

        self.onlineInit()

        self.messagesInit()

    def quit(self):

        self.cursor.execute("DELETE FROM online") #empty online database on exit

        self.connection.commit()

        self.connection.commit()

        self.connection.close()

    def knownInit(self): #create database 'known'

        try:

            self.cursor.execute("""CREATE TABLE IF NOT EXISTS known (

            NodeID text NOT NULL PRIMARY KEY,

            PublicKeyN int,

            PublicKeyE int,

            Online int,

            LastOnline text

            )""")

            self.connection.commit()

        except:

            print("ERROR: Database Error")

    def onlineInit(self): #create database 'online'

        try:

            self.cursor.execute("""CREATE TABLE IF NOT EXISTS online (

            CircuitID text NOT NULL PRIMARY KEY,

            ExitIP text,

            ExitPort int,

            NodeID text,

            FOREIGN KEY (NodeID) REFERENCES known(NodeID)

            )""")

            self.connection.commit()

        except:

            print("ERROR: Database Error")

    def messagesInit(self): #create database 'messages'

        try:

            self.cursor.execute("""CREATE TABLE IF NOT EXISTS messages (

            Timestamp int,

            NodeID text,

            Own int,

            Message text,

            FOREIGN KEY (NodeID) REFERENCES known(NodeID)

            )""")

            self.connection.commit()

        except:

            print("ERROR: Database Error")

    def getMessages(self, NodeID): #return all messages where provided NodeID matches Foreign Key 'NodeID'

        self.cursor.execute(f"SELECT Timestamp, Own, Message FROM messages WHERE NodeID='{NodeID}' ORDER BY Timestamp ASC")

        data = self.cursor.fetchall()

        return data

    def writeMessages(self, NodeID, MessageList): #write new set of messages to database after a conversation is over

        #list layout: [[timestamp, own, message], [timestamp, own, message], ...]

        try:

            for x in range(len(MessageList)):

                self.cursor.execute(f"""INSERT or IGNORE INTO messages VALUES (

                    {int(MessageList[x][0])},

                    '{NodeID}',

                    {int(MessageList[x][1])},

                    '{MessageList[x][2]}'

                    )""")

                self.connection.commit()

        except:

            pass

    def showKnown(self, NodeID = None): #return data from 'known' SQL table where parameter equals Primary Key NodeID

        if NodeID == None:

            self.cursor.execute(f"SELECT \* FROM known")

        else:

            self.cursor.execute(f"SELECT \* FROM known WHERE NodeID='{NodeID}'")

        data = self.cursor.fetchall()

        return data

    def showOnline(self, NodeID = None): #return data from 'online' SQL table where parameter equals Primary Key NodeID

        if NodeID == None:

            self.cursor.execute(f"SELECT \* FROM online")

        else:

            self.cursor.execute(f"SELECT \* FROM online WHERE NodeID='{NodeID}'")

        data = self.cursor.fetchall()

        return data

    def getKeys(self, NodeID): #get public key of a user with NodeID name 'NodeID' from SQL table known

        hashID = Cryptography.hash(NodeID)

        self.cursor.execute(f"SELECT PublicKeyN, PublicKeyE FROM known WHERE Hash='{hashID}'")

        data = self.cursor.fetchall()

        if data == []:

            return (None, None)

        else:

            return data[0]

    def writeKnown(self, NodeID, PublicKeyN, PublicKeyE, Online, LastOnline): #write new data to SQL table known

        self.cursor.execute(f"INSERT or IGNORE INTO known VALUES ('{NodeID}', {PublicKeyN}, {PublicKeyE}, {Online}, '{LastOnline}')")

        self.connection.commit()

    def writeOnline(self, CircuitID, ExitIP, NodeID): #write new data to SQL table online

        self.cursor.execute(f"INSERT or IGNORE INTO online VALUES ('{CircuitID}', '{ExitIP[0]}', {ExitIP[1]}, '{NodeID}')")

        self.connection.commit()

    def deleteKnown(self, NodeID): #delete data from known

        self.cursor.execute(f"DELETE FROM known WHERE NodeID='{NodeID}'")

        self.connection.commit()

    def deleteOnline(self, NodeID): #delete data from online

        self.cursor.execute(f"DELETE FROM online WHERE NodeID='{NodeID}'")

        self.connection.commit()

class OwnCircuit(): #the own circuit of the user (ownership belongs to current user)

    def \_\_init\_\_(self):

        global joinableNodes

        self.\_\_idSet = False

        self.\_\_CircuitID = self.\_\_GenerateCircuitID()

        self.\_\_DownNode = ('EMPTY', 0)

        self.\_\_endIP = ('EMPTY', 0)

        Transmission.getNodes()

    def buildCircuit(self): #self-explanatory

        self.\_\_request("JOINCIRCUIT")

    def getCircuitID(self): #self-explanatory

        return self.\_\_CircuitID

    def setDownNode(self, IP): #set value of DownNode IP Address

        self.\_\_DownNode = IP

    def getDownNode(self): #return value of DownNode IP Address

        return self.\_\_DownNode

    def setEndNode(self, IP): #set value of EndNode IP Address

        self.\_\_endIP = IP

    def \_\_GenerateCircuitID(self): #generate CircuitID for new circuit, can only be used once

        if self.\_\_idSet == False:

            self.\_\_idSet = True

            tempID = ''

            for x in range(8):

                tempID = tempID + base64Chars[random.randint(0, len(base64Chars) - 1)]

            return tempID

        else:

            print("ERROR: CircuitID already set!")

    def \_\_GenerateMsgID(self): #generate msgID for new packet to be sent

        msgID = ""

        for x in range(8):

            msgID = msgID + base64Chars[random.randint(0, len(base64Chars) - 1)]

        return msgID

    def \_\_send(self, msg): #send packet via Transmission class

        Transmission.send(msg, self.\_\_DownNode)

    def login(self): #login subroutine, sends request to central server

        print("Logging into server...")

        timestamp = time.time()

        login\_payload = username

        login\_payload = Cryptography.asymEncrypt(login\_payload, serverKey)

        msgID = self.\_\_GenerateMsgID()

        msg = [timestamp, "LOGIN", "SERVER", [msgID, self.\_\_CircuitID, myPublicKey], login\_payload]

        signature = Cryptography.sign(pickle.dumps(msg), myPrivateKey)

        msg.append(signature)

        msg = pickle.dumps(msg)

        ongoingPackets.append("LOGIN")

        self.\_\_send(msg)

        timeWait = Transmission.wait(timestamp, "LOGIN")

        if (timeWait == 'TIMEOUT') or (self.\_\_endIP[0] == 'EMPTY'):

            print("Unable to connect to the network, please try again later.")

            ProgramQuit = True

    def register(self): #register subroutine, sends request to central server

        global ProgramQuit

        print(f"Registering User: {username}")

        timestamp = time.time()

        register\_payload = username

        register\_payload = Cryptography.asymEncrypt(register\_payload, serverKey)

        msgID = self.\_\_GenerateMsgID()

        msg = [timestamp, "REGISTER", "SERVER", [msgID, self.\_\_CircuitID, myPublicKey], register\_payload]

        signature = Cryptography.sign(pickle.dumps(msg), myPrivateKey)

        msg.append(signature)

        msg = pickle.dumps(msg)

        ongoingPackets.append("REGISTER")

        self.\_\_send(msg)

        timeWait = Transmission.wait(timestamp, "REGISTER")

        if (timeWait == 'TIMEOUT') or (self.\_\_endIP[0] == 'EMPTY'):

            ProgramQuit = True

    def nodeInfo(self, NodeID): #node information request for central server

        global temp\_pubKey, temp\_CircuitID, temp\_ExitIP

        timestamp = time.time()

        info\_payload = Cryptography.asymEncrypt(NodeID, serverKey)

        msgID = self.\_\_GenerateMsgID()

        msg = [timestamp, "NODEINFO", "SERVER", [msgID, self.\_\_CircuitID, myPublicKey], info\_payload]

        signature = Cryptography.sign(pickle.dumps(msg), myPrivateKey)

        msg.append(signature)

        msg = pickle.dumps(msg)

        ongoingPackets.append("NODEINFO")

        self.\_\_send(msg)

        timeWait = Transmission.wait(timestamp, "NODEINFO")

        if (timeWait == 'TIMEOUT') or (self.\_\_endIP[0] == 'EMPTY'):

            ProgramQuit = True

        else:

            exists = database.showKnown(NodeID)

            if exists == [] and temp\_pubKey != (None, None):

                #if data for the given node does not exist, and the key provided by the server is not (None, None)

                if temp\_CircuitID == None:

                    database.writeKnown(NodeID, temp\_pubKey[0], temp\_pubKey[1], 0, 0)

                else:

                    database.writeKnown(NodeID, temp\_pubKey[0], temp\_pubKey[1], 1, time.time())

                    database.writeOnline(temp\_CircuitID, temp\_ExitIP, NodeID)

            elif temp\_pubKey != (None, None):

                #if data exists, write only to the online database

                database.writeOnline(temp\_CircuitID, temp\_ExitIP, NodeID)

    def \_\_request(self, type): #function to encompass various 'request' commands

        #currently only contains one type "REQUEST:JOINCIRCUIT", more to be added in the next version of the code

        global ProgramQuit

        if type == "JOINCIRCUIT":

            #recursive circuit generation algorithm spanning complex client-server model (see documentation for more details)

            print("Setting up circuit...")

            successful = False

            x = 0

            while successful == False:

                #iterates until the process is successful or until it times out

                timestamp = time.time()

                msg = [timestamp, "REQUEST:JOINCIRCUIT", self.\_\_CircuitID, myPublicKey]

                signMsg = pickle.dumps(msg)

                signature = Cryptography.sign(signMsg, myPrivateKey)

                msg.append(signature)

                msg = pickle.dumps(msg)

                randomNode = joinableNodes[x]

                print(f"Asking {randomNode} to join circuit...")

                ongoingPackets.append(self.\_\_CircuitID)

                Transmission.send(msg, randomNode)

                waitResult = Transmission.wait(timestamp, self.\_\_CircuitID, 10)

                if waitResult == True:

                    successful = True

                    self.setDownNode = randomNode

                    break

                elif waitResult == 'TIMEOUT':

                    print(f"No reply from {randomNode}")

                    x += 1

                elif waitResult == 'PACKET\_FAIL':

                    print("ERROR: Request failed to send!")

                    x += 1

                    break

                else:

                    x += 1

                if x == (len(joinableNodes)):

                    break

            waitResult = False

            ongoingPackets.append(f"END{self.\_\_CircuitID}")

            if successful == True:

                print(f"{self.\_\_DownNode} accepted request!")

                print("Waiting for EndNode...")

                waitResult = Transmission.wait(timestamp, f"END{self.\_\_CircuitID}", 40)

                #if the user was able to connect to a node, it waits for the end node to send a reply across the circuit

            if waitResult == False or waitResult == 'TIMEOUT':

                successful = False

            if successful == False:

                print("Failed to connect to any nodes, quitting...")

                ProgramQuit = True

            else:

                print(f"End Node is: {self.\_\_endIP}")

    def quit(self, fromOther=False): #quit circuit

        if fromOther == True:

            pass

class NonOwnCircuit(): #same as above but for circuits the user has no ownership of

    #dormant code as there is a port-forwarding error that is too complex to fix, does not work with loopback (beyond the scope of the NEA)

    #functions same as the class above

    def \_\_init\_\_(self, CircuitID, PublicKey, EndStatus):

        self.\_\_CircuitID = CircuitID

        self.\_\_publicKey = PublicKey

        self.\_\_EndStatus = EndStatus

        if self.\_\_EndStatus == True:

            self.\_\_UpNode = None #node towards owner

        else:

            self.\_\_DownNode = None #node towards end

            self.\_\_UpNode = None

    def getCircuitID(self):

        return self.\_\_CircuitID

    def sendUp(self, data): #sends data up the circuit (to the owner)

        #[timestamp, header, destinationID, [msgID, circuitID, publicKey], payload, signature]

        destinationID = data[2]

        msgID = data[3][0]

        circuitID = data[3][1]

        data = pickle.dumps(data)

        ongoingPackets.append(msgID) #adds packet to list of packets without reply

        while True:

            Transmission.send(data, self.\_\_UpNode)

            time.sleep(5)

            if msgID not in ongoingPackets:

                break

    def sendDown(self, data): #sends data out of the circuit or down the circuit (towards the end node)

        #get ip of end node of destination from server and send

        #[timestamp, header, destinationID, circuitID, payload, signature]

        destinationID = data[2]

        msgID = data[3][0]

        tempData = data

        signature = tempData.pop(5)

        tempData = pickle.dumps(tempData)

    def exit(self, sender = None): #exit circuit by sending exit packets to up and down nodes

        quitMessage = [time.time(), "QUIT", self.\_\_CircuitID]

        quitMessage = pickle.dumps(quitMessage)

        if sender == None:

            Transmission.send(quitMessage, self.\_\_UpNode)

            Transmission.send(quitMessage, self.\_\_DownNode)

        elif sender == self.\_\_UpNode:

            Transmission.send(quitMessage, self.\_\_DownNode)

        elif sender == self.\_\_DownNode:

            Transmission.send(quitMessage, self.\_\_UpNode)

class Transmission(): #class that is a grouping of functions related to the transmission of data across the network

### INTERNAL FUNCTIONS

    def send(msg, destination): #sends packet to destination via outgoing UDP socket

        global packetFailure, ProgramQuit

        try:

            sock.sendto(msg, destination)

        except:

            #should any packet fail to send, it indicates an error with the socket - therefore meaning the program must restart

            print("ERROR: Could not send packet")

            ProgramQuit = True

            packetFailure = True

    def receiveAgent(): #run as a thread, looks for incoming packets and parses them as a separate thread so multiple can be handled at once

        while True:

            try:

                data, sender = sock.recvfrom(1024)

                threading.Thread(target = Transmission.packetHandler, args=(data, sender)).start()

                if ProgramQuit == True:

                    break

            except:

                break

    def quit(): #transmission quit, closes socket to avoid port allocation error when opening program again

        sock.close()

    def checkOwnership(data): #checks ownership of incoming packet, can be either: on the sender's circuit, the recipient's circuit, or is with the recipient

        #[timestamp, "DATA", destinationID, [msgID, circuitID, publicKey], payload, signature]

        destinationID = data[2]

        circuitID = data[3][1]

        found = False

        inSenderCircuit = False

        isOwnCircuit = False

        index = 0

        for x in range(len(nonOwnCircuits)): #check if packet is on correct route

            if circuitID == nonOwnCircuits[x].getCircuitID(): #is sill on the sender's circuit

                found = True

                inSenderCircuit = True

                index = x

                break

            elif destinationID == nonOwnCircuits[x].getCircuitID(): #is on the recipient's circuit

                found = True

                index = x

                break

        if destinationID == ownCircuit.getCircuitID(): #message is for self

            found = True

            isOwnCircuit = True

        if found == True: #packet is on correct route

            if isOwnCircuit == True: #packet belongs to self

                return 'OWN', ownCircuit

            elif inSenderCircuit == True: # packet is still on the sender's circuit

                return 'IN\_SENDER', nonOwnCircuits[x]

            else: # packet is on the recipient's circuit

                return 'IN\_RECIPIENT', nonOwnCircuits[x]

        else: #packet is random

            return 'RANDOM', None

    def packetHandler(data, sender): #general packet handler

        try:

            data = pickle.loads(data)

            timestamp = data[0]

            header = data[1]

            if time.time() > timestamp + TIMEOUT:  #if the timestamp shows that the packet is older than the timeout value, forget packet

                pass # do nothing as packet has timed out

            elif header == "MSG": #message handler

                if messenger != None:

                    msg = Cryptography.asymDecrypt(data[3], myPublicKey, myPrivateKey)

                    messenger.printMessage(timestamp, msg, 0, True)

            elif (header == "LOGIN") or (header == "REGISTER") or (header == "NODEINFO"): #to server

                pass

            elif header == "DATA": #to/from anyone

                status, circuit = Transmission.checkOwnership(data)

                if status == "OWN":

                    Transmission.ownPacketHandler(data, sender)

                elif status == "IN\_SENDER":

                    pass

                    #circuit.sendDown(data, sender)

                elif status == "IN\_RECIPIENT":

                    pass

            elif header[:5] == 'REPLY':

                Transmission.replyHandler(data, sender)

            elif header[:7] == 'REQUEST':

                Transmission.requestHandler(data, sender)

        except:

            print("ERROR: Packet incorrectly formatted!")

    def replyHandler(data, sender): #reply handler

        global serverName, serverAnnouncements, serverOnline, joinableNodes, serverKey, ownCircuit

        header = data[1]

        msgID = header[6:] #get characters after 'REPLY:'

        if msgID == "TEST":

            #[timestamp, header, servername, announcements]

            serverName = data[2]

            serverAnnouncements = data[3]

            serverKey = data[4]

            serverOnline = True

            Config.updateGlobalConfig()

        elif msgID == "GETNODES":

            #[timestamp, header, [NodeList]]

            joinableNodes = data[2]

        elif msgID == "JOINCIRCUIT": #circuit routed

            #[timestamp, header, CircuitID, recipientIP]

            destination = data[3]

            if destination != currentIP:

                return #exit if reply is not meant for self

            ownCircuit.setDownNode(sender)

            msgID = data[2]

        elif msgID == f"ENDCIRCUIT": #circuit routed

            #[timestamp, header, CircuitID, endIP, recipientIP]

            if data[2] == ownCircuit.getCircuitID():

                ownCircuit.setEndNode(data[3])

                msgID = "END" + data[2]

            else:

                for x in range(len(nonOwnCircuits)):

                    if nonOwnCircuits[x].getCircuitID() == data[2]:

                        nonOwnCircuits[x].handle(data, sender)

                        break

        else:

            destination = data[2]

            if destination != currentIP:

                return #exit if reply is not meant for self

        index = DataManipulation.linearSearch(msgID, ongoingPackets)

        if index != -1:

            ongoingPackets.pop(index)

    def requestHandler(data, sender): #request-type packet handler

        #[timestamp, "REQUEST:type", ----]

        header = data[1]

        requestType = header[8:]

        if requestType == "JOINCIRCUIT":

            Transmission.joinCircuit(data, sender)

            Transmission.reply(requestType, sender)

    def ownPacketHandler(data, sender): #handler for packets belonging to self

        global temp\_pubKey, temp\_CircuitID, temp\_ExitIP

        #[timestamp, "DATA", destinationID, [msgID, circuitID, publicKey], payload, signature] #from another user

        #[timestamp, "DATA", destinationID, [msgID, "SERVER", serverKey] payload, signature] #from server

        timestamp = data[0]

        messageInfo = data[3]

        payload = data[4] #encrypted payload

        #payload layout [sender, contents]

        try:

            sender = payload[0]

            contents = payload[1]

            if sender == "SERVER":

                senderKeys = serverKey

            else:

                senderKeys = database.getKeys(sender)

            if senderKeys == (None, None):

                Transmission.nodeInfo(sender)

            if contents[0] == "SERVERREPLY":

                decryptedSection = Cryptography.asymDecrypt(contents[1], myPublicKey, myPrivateKey)

                Transmission.serverReply(decryptedSection)

            elif contents[0] == "INFO":

                temp\_pubKey = contents[1][0]

                temp\_CircuitID = contents[1][1]

                temp\_ExitIP = contents[1][2]

                index = DataManipulation.linearSearch("NODEINFO", ongoingPackets)

                if index != -1:

                    ongoingPackets.pop(index)

                print(contents[1])

            else:

                pass #do nothing as the packet is invalid

        except:

            pass #packet has formatting error

    def serverReply(reply): #handles replies from server

        global loginStatus, registerStatus

        print('\_\_\_\_\_\_\_\_\_\_' + reply + '\_\_\_\_\_\_\_\_\_\_')

        if reply == "LOGIN:SUCCESS":

            index = DataManipulation.linearSearch("LOGIN", ongoingPackets)

            if index != -1:

                ongoingPackets.pop(index)

            loginStatus = True

        elif reply == "LOGIN:FAIL":

            index = DataManipulation.linearSearch("LOGIN", ongoingPackets)

            if index != -1:

                ongoingPackets.pop(index)

            loginStatus = False

        elif reply == "REGISTER:SUCCESS":

            index = DataManipulation.linearSearch("REGISTER", ongoingPackets)

            if index != -1:

                ongoingPackets.pop(index)

            registerStatus = True

        elif reply == "REGISTER:FAIL":

            index = DataManipulation.linearSearch("REGISTER", ongoingPackets)

            if index != -1:

                ongoingPackets.pop(index)

            registerStatus = False

    def joinCircuit(data, sender): #join a given circuit and append it as a python object to the nonOwnCircuit array

        #[timestamp, "REQUEST:JOINCIRCUIT", circuitID, publicKey, signature]

        timestamp = data[0]

        CircuitID = data[2]

        publicKey = data[3]

        signature = data.pop(4)

        testData = pickle.dumps(data)

        chance = random.randint(1, RANDOM\_CONST)

        if (chance == 1) or (time.time() > timestamp + TIMEOUT - 20):

            nonOwnCircuits.append(NonOwnCircuit(CircuitID, publicKey, True))

        else:

            nonOwnCircuits.append(NonOwnCircuit(CircuitID, publicKey, False))

    def wait(startTime, msgID, length=60): #WAIT ANIMATION FUNCTION, waits for a packet with a certain header to arrive before quitting

        global ongoingPackets, packetFailure

        loadingStrings = ["Waiting for reply --- |", "Waiting for reply --- /", "Waiting for reply --- -", "Waiting for reply --- \\"]

        while (startTime + length) > time.time():

            for x in range(4):

                print(loadingStrings[x], end='\r')

                if (startTime + length) < time.time():

                    print("                        ", end='\r') #clear last line

                    return 'TIMEOUT'

                elif msgID not in ongoingPackets:

                    print("                        ", end='\r') #clear last line

                    return True

                elif packetFailure == True:

                    packetFailure = False

                    print("                        ", end='\r')

                    return 'PACKET\_FAIL'

                time.sleep(0.5)

        print("                        ", end='\r') #clear last line

        index = DataManipulation.linearSearch(msgID, ongoingPackets)

        if index != -1:

            ongoingPackets.pop(index) #to avoid double messageID occurrence after timeout

        return False

### DIRECT PACKETS

    def testServer(): #send test packet to server, server replies with it's name and any new announcements

        startTime = time.time()

        ongoingPackets.append("TEST")

        Transmission.send(pickle.dumps([time.time(), "TEST"]), serverIP)

        sendWait = Transmission.wait(startTime, "TEST", 10)

    def getCircuitIP(DestinationID): #redundant code, kept for compatibility

        pass

    def reply(msgID, sender):

        msg = [time.time(), "REPLY:" + msgID, sender]

        Transmission.send(msg, sender)

    def getNodes(): #send node request packet to server, server replies with an array of nodes the user can create a circuit with

        global ongoingPackets

        print("Getting Nodes...")

        timestamp = time.time() #unix time

        msg = [timestamp, "GETNODES"]

        msg = pickle.dumps(msg)

        ongoingPackets.append("GETNODES")

        while True:

            Transmission.send(msg, serverIP)

            if Transmission.wait(timestamp, "GETNODES") == True:

                break

class DataManipulation():

    def linearSearch(val, array): #linear search algorithm for finding the index of a specified value within an array, used for unsorted lists

        result = None

        for x in range(len(array)):

            if array[x] == val:

                result = x

        if result == None:

            return -1

        else:

            return result

    def decToBin(val): #converts decimal to binary

        global binresult

        binresult = "" #as the variable is set as global, it must be emptied before each use

        def main(val): #advanced nested function, to allow recursion of a specified part of the code to run while excluding others

            global binresult

            if val > 1:

                main(val//2) #recursive section, continues until a zero is reached

            binresult = binresult + str(val%2)

        main(val) #executes recursive function

        if len(binresult) < 6: #changes the result to a six-bit binary integer, as a minimum

            add = (6 - len(binresult))

            zeroes = ""

            for x in range(add):

                zeroes = zeroes + "0"

            binresult = zeroes + binresult

        return binresult

    def binToDec(val): #converts unsigned integer binary to decimal

        length = len(val)

        result = 0

        for x in range(length): #iterates through each bit in a given binary value

            if val[x] == "1":

                result = result + (2\*\*(length - (x + 1))) #adds the calculated denary value of the point to the result variable

        return result

###

    def base64ToDec(val): #converts Base64 to Decimal

        global base64Chars

        result = "" #initialise local variable 'result' as empty string to allow concatenation

        for x in range(len(val)): #iterates through all characters in the specified value

            if val[x] == "=": #buffer

                pass

            else:

                temp = DataManipulation.linearSearch(val[x], base64Chars) # get decimal value of base64 character

                result = result + DataManipulation.decToBin(temp)

        result = DataManipulation.binToDec(result)

        return result

###

    def decToHex(val): #converts a decimal value to hexadecimal

        global hexChars

        modVal = val % 16

        tempVal = val // 16

        if tempVal == 0:

            return hexChars[modVal] #the final value is returned if it is divisible by 16 without remainder

        else:

            return DataManipulation.decToHex(tempVal) + hexChars[modVal] #using recursion, the algorithm passes through each hexadecimal digit until the resulting number is divisible by 16 without remainder (and is therefore a hexadecimal)

def inputUnrec():

    print("ERROR: Command not recognised, please try again.\n") #exception for when commands are unrecognised by the console

def initAnim(): #to initiate loading bar, use initAnim() followed by a succession of a varied amount of progress based on program execution via loadAnim(x)

    print("|LOADING...          |") #Note: x in loadAnim(x) must add up to 20 for each individual execution of initAnim(), not more, not less

    print(" ", end='', flush=True)

def loadAnim(count):

    for x in range(count):

        print("#", end='', flush=True)

        time.sleep(0.075) #sleeps for an arbitrary amount

if \_\_name\_\_ == '\_\_main\_\_':

    try:

        Interface.bootstrap()

        if ProgramQuit == True:

            try:

                database.quit()

            except:

                pass

            Transmission.quit()

            quit()

    except KeyboardInterrupt:

        exit()

# 4.4 Server File

from logging import exception

import socket, threading, time, pickle, sqlite3, random

import HGCrypto as Cryptography

from structures import LimitlessQueue

serverName = "HGServer" #default name

announcements = "No New Announcements" #default announcement

ProgramQuit = False

choicesIP = []

tempSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

tempSock.connect(("10.255.255.255", 80)) #arbitrary IP address used to get local IP address of device rather than loopback

choicesIP.append(tempSock.getsockname()[0])

tempSock.close()

choicesIP.append(socket.gethostbyname(socket.gethostname()))

knownIPs = []

usedCircuits = []

TIMEOUT = 120

#### SPECIAL TESTING CODE ###

test = True

def testprint(string): #Console output used for testing, only prints when the test flag is set to true

    if test == True:

        print(string)

### END ###

def linearSearch(val, array): #linear search algorithm for finding the index of a specified value within an array, used for unsorted lists

    result = None

    for x in range(len(array)):

        #result is returned if the value in the current index matches the value to be searched

        if array[x] == val:

            result = x

    if result == None:

        return -1

    else:

        return result

def updateGlobalConfig(): #creates or updates the global config file if data is altered in the program

    globalFile = open("Global.hg", "w")

    lines = ["[N]", str(myPublicKey[0]), "[PUBLIC KEY E]", str(myPublicKey[1]), "[PRIVATE KEY D]", str(myPrivateKey[1])]

    #data stored in plaintext form on individual lines

    for x in range(len(lines)):

        globalFile.writelines(lines[x] + "\n")

def globalConfig(): #subroutine for accessing data in the global config file 'Global.hg'

    global myPublicKey, myPrivateKey

    try: #Try creating a global file, redirects to read if exists

        globalFile = open("Global.hg", "x") #creates the specified file

        globalFile.close() #closes the file to enable access to writing/appending mode

        myPublicKey, myPrivateKey = Cryptography.asymKeyGen(100)

        updateGlobalConfig()

    except: #Read global file

        lines = []

        globalFile = open("Global.hg", "r")

        for x in globalFile.readlines():

            lines.append(x.strip()) #remove new line at end of each string

        myPublicKey = (int(lines[1]), int(lines[3]))

        myPrivateKey = (int(lines[1]), int(lines[5]))

class TempCircuit(): #non-transmitting temporary nodes created by the server as a circuit needs more than 2 people to connect, but a person cannot connect without a circuit

    def \_\_init\_\_(self):

        fail = True

        self.\_\_CircuitID = None

        self.\_\_Complete = False

        while fail == True:

            try:

                self.internal\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

                self.\_\_IP = (currentIP[0], random.randint(300, 1000))

                self.internal\_sock.bind(self.\_\_IP)

                fail = False

                #program attempts to bind the circuit to the random port above

                #if the port is occupied, the whole process is started again until success

            except:

                fail = True

        threading.Thread(target = self.internal\_recv).start()

    def internal\_recv(self): #internal socket only used by the circuit

        while True:

            try:

                data, sender = self.internal\_sock.recvfrom(1024)

                threading.Thread(target = self.internal\_packetHandler, args=(data, sender)).start()

                if ProgramQuit == True:

                    break

            except:

                break

    def internal\_packetHandler(self, data, sender): #internal circuit handling

        try:

            data = pickle.loads(data)

            timestamp = data[0]

            header = data[1]

            if time.time() > timestamp + TIMEOUT:

                pass # do nothing as packet has timed out

            elif header == "MSG":

                #if header indicated that the packet is a message packet, send it up since messages originate directly from senders

                self.sendUp(pickle.dumps(data))

            elif (header == "LOGIN") or (header == "REGISTER") or (header == "NODEINFO"): #to server

                #if header matches any of the above headers, send it down since it is for the server

                self.sendDown(pickle.dumps(data))

            elif header == "DATA": #to/from anyone

                #ownership of the packet must be checked before sending up or down

                status = self.checkOwnership(data)

                if status == "IN\_SENDER":

                    #if the packet is in the sender's circuit send it down

                    self.sendDown(pickle.dumps(data))

                elif status == "IN\_RECIPIENT":

                    #else send it up

                    self.sendUp(pickle.dumps(data))

            elif header[:5] == 'REPLY': #reply packet handler

                msgID = header[6:] #get characters after 'REPLY:'

                if msgID == "JOINCIRCUIT":

                    #[timestamp, header, CircuitID, recipientIP]

                    if (data[2] == self.\_\_CircuitID) and (data[3] == self.\_\_IP):

                        msgID = data[2] #stop asking nodes to join

                elif msgID == "ENDCIRCUIT":

                    #[timestamp, header, CircuitID, endIP, recipientIP]

                    if data[2] == self.\_\_CircuitID:

                        self.sendUp(pickle.dumps(data))

                        return

                index = linearSearch(msgID, self.\_\_ongoingPackets)

                #the messageID of all reply packets are stored in the 'ongoingPackets' array

                #subroutines continuously send out packets until the msgID times out or it is popped from the array

                #it is popped from the array if reply packet is received

                #linear search above checks if the reply packet exists in the array

                if index != -1:

                    #if it exists, pop it from the array

                    self.\_\_ongoingPackets.pop(index)

            elif header[:7] == 'REQUEST':

                #request packets are sent directly to the request handler

                self.requestHandler(data, sender)

        except:

            print("ERROR: Packet incorrectly formatted!")

    def requestHandler(self, data, sender):

        #[timestamp, "REQUEST:type", ----]

        header = data[1]

        requestType = header[8:]

        if requestType == "JOINCIRCUIT":

            #layout: [timestamp, "REQUEST:JOINCIRCUIT", CircuitID, PublicKey, Signature]

            #currently there is only one type of request packet

            #joincircuit packets establish a circuit using recursion spanning over a complex client-server model

            #recursion layout: sender -> node -> node -> node -> endnode

            #once function is completed at the end node, it returns back to the sender to indicate completion of the circuit creation subroutine

            #endnode -> node ... node -> sender

            key = data[3]

            signature = data.pop(4)

            if data[2] == self.\_\_CircuitID: #if the circuitID of the request matches the currently joined circuit, do nothing as you can't connect twice

                return

            if Cryptography.verify(pickle.dumps(data), signature, key) == True:

                ### PROBABALISTIC END NODE STATUS SELECTION (MAX-ANONYMITY) ###

                if random.randint(1,3) == 1:

                    end = True

                else:

                    end = False

                ### END ###

                #using probability, the system generates a number between 1 and 3, inclusive, and if the number is one it decides to become the end-node

                #1 in 3 chance of becoming the end node

                #therefore, mathematically using E(X) (expected value) the average size of every circuit should theoretically be 2

                #see documentation for more details

                data.append(signature)

                self.setup(data[2], key, end, sender, pickle.dumps(data))

    def checkOwnership(self, data):

        #[timestamp, "DATA", destinationID, [msgID, originID, publicKey], payload, signature]

        destinationID = data[2]

        originID = data[3][1]

        found = False

        inSenderCircuit = False

        if originID == self.\_\_CircuitID: #is sill on the sender's circuit

            found = True

            inSenderCircuit = True

        elif destinationID == self.\_\_CircuitID: #is on the recipient's circuit

            found = True

        if found == True: #packet is on correct route

            if inSenderCircuit == True: # packet is still on the sender's circuit

                return 'IN\_SENDER'

            else: # packet is on the recipient's circuit

                return 'IN\_RECIPIENT'

        else: #packet is random

            return 'RANDOM'

    def getIP(self):

        return self.\_\_IP

    def setup(self, CircuitID, PublicKey, EndStatus, UpNode, Full):

        if self.\_\_Complete == False:

            self.\_\_CircuitID = CircuitID

            self.\_\_publicKey = PublicKey

            self.\_\_EndStatus = EndStatus

            self.\_\_UpNode = UpNode #node towards owner

            self.\_\_ongoingPackets = []

            reply = [time.time(), "REPLY:JOINCIRCUIT", self.\_\_CircuitID, self.\_\_UpNode]

            self.sendUp(pickle.dumps(reply))

            #send reply packet to the node it received the request from

            #the node above will then stop sending requests to other nodes once a reply is received

            Transmission.moveCircuit(CircuitID)

            #the subroutine above removes this object from the temporary circuit array to the ongoing circuit one

            #once this happens, the server will stop sending the IP address of this node to other users via the GetNodes request

            if self.\_\_EndStatus == True:

                #since this node has decided to become the End Node, there is no need to ask another node to connect

                self.\_\_DownNode = None

            else:

                #get new node for down node

                Nodes = Transmission.getNodes(self.\_\_IP, True)

                x = 0

                while True:

                    selectedNode = Nodes[x]

                    self.internal\_sock.sendto(Full, selectedNode)

                    self.\_\_ongoingPackets.append(self.\_\_CircuitID)

                    sendWait = self.wait(time.time(), self.\_\_CircuitID, 8)

                    #see below for the 'wait' subroutine, essentially waits until a given node replies that they have received a sent packet

                    if sendWait == True:

                        self.\_\_DownNode = selectedNode

                        break

                    else:

                        x += 1

                    #iterate through every IP address in the array sent by the server through the getNodes request

                    #send request to join this circuit

                    #only stops until every node has been iterated through, or if a node connects to the circuit

                    if x == len(Nodes):

                        break

            if self.\_\_EndStatus == True:

                time.sleep(1) #wait to avoid out of sequence packets

                endReply = [time.time(), "REPLY:ENDCIRCUIT", self.\_\_CircuitID, self.\_\_IP]

                self.sendUp(pickle.dumps(endReply))

                #send packet up the circuit to indicate that it is complete

            try:

                knownIPs.remove(self.\_\_IP)

                #remove the IP of this node from the list of connectable IP addresses

            except:

                pass

            self.\_\_Complete = True

    def getCircuitID(self):

        return self.\_\_CircuitID

    def sendUp(self, data): #sends data up the circuit (to the owner)

        print(self.\_\_UpNode)

        self.internal\_sock.sendto(data, self.\_\_UpNode)

    def sendDown(self, data): #sends data out of the circuit or down the circuit (towards the end node)

        if self.\_\_EndStatus != True:

            self.internal\_sock.sendto(data, self.\_\_DownNode)

        else: #get IP of next

            #[timestamp, header, destinationID, [msgID, circuitID, publicKey], payload, signature]

            tempData = pickle.loads(data)

            destinationID = tempData[2]

            if destinationID == "SERVER":

                self.internal\_sock.sendto(data, currentIP) #serverIP for clients

            else:

                destination = database.getCircuitIP(destinationID)

                try:

                    self.internal\_sock.sendto(data, destination)

                except:

                    pass

    def wait(self, startTime, msgID, length=30):

        #subroutine to for a reply to a packet with a given msgID

        while (startTime + length) > time.time():

            #nested iteration, loops until the timeout is reached

            for x in range(4):

                if (startTime + length) < time.time():

                    return 'TIMEOUT'

                elif msgID not in self.\_\_ongoingPackets:

                    #if the given msgID is removed from the array self.\_\_ongoingPackets, it means that a reply was received

                    return True

                time.sleep(0.5)

        index = linearSearch(msgID, self.\_\_ongoingPackets)

        if index != -1:

            self.\_\_ongoingPackets.pop(index) #to avoid double messageID occurrence after timeout

        return False

    def quit(self, sender = None):

        #quit subroutine, either this node can decide to quit, or another node has

        quitMessage = [time.time(), "QUIT", self.\_\_CircuitID]

        quitMessage = pickle.dumps(quitMessage)

        if sender == None:

            #if this node has decided to quit, send it to both front and back nodes

            self.sendUp(quitMessage)

            self.sendDown(quitMessage)

        elif sender == self.\_\_UpNode:

            #if the node above has decided to quit, send it to the node below

            self.sendDown(quitMessage)

        elif sender == self.\_\_DownNode:

            #same as above, but vice versa

            self.sendUp(quitMessage)

class Database():

    def \_\_init\_\_(self):

        self.connection = sqlite3.connect('database.db', check\_same\_thread=False)

        self.cursor = self.connection.cursor()

        self.knownInit()

        self.onlineInit()

        #run database initialisation subroutine

        self.cursor.execute("DELETE FROM online") #empty online database on opening

        self.cursor.execute(f"""UPDATE known SET Online = 0""")

        self.connection.commit()

        threading.Thread(target = self.timeBasedLogout).start()

        #run thread to automatically logout users who have been online for more than a set amount of seconds

    def quit(self):

        self.cursor.execute("DELETE FROM online") #empty online database on exit

        self.cursor.execute(f"""UPDATE known SET Online = 0""")

        self.connection.commit()

        self.connection.close()

    def timeBasedLogout(self):

        global ProgramQuit

        while ProgramQuit == False:

            self.cursor.execute(f"""UPDATE known SET Online = 0 WHERE LastOnline < {time.time() - 600}""")

            time.sleep(5)

    def knownInit(self):

        try:

            self.cursor.execute("""CREATE TABLE IF NOT EXISTS known (

            Hash text NOT NULL PRIMARY KEY,

            NodeID text,

            PublicKeyN int,

            PublicKeyE int,

            Online int,

            LastOnline int

            )""")

            self.connection.commit()

        except:

            print("ERROR: Database Error")

    def onlineInit(self):

        try:

            self.cursor.execute("""CREATE TABLE IF NOT EXISTS online (

            Hash text NOT NULL PRIMARY KEY,

            CircuitID text,

            ExitIP text,

            ExitPort int,

            NodeID text,

            FOREIGN KEY (NodeID) REFERENCES known(NodeID)

            )""")

            self.connection.commit()

        except:

            print("ERROR: Database Error")

    def getKeys(self, NodeID):

        #parse get key request for a given NodeID

        hashID = Cryptography.hash(NodeID)

        #since we are accessing a hash table, we must first generate a hash for the provided NodeID

        self.cursor.execute(f"SELECT PublicKeyN, PublicKeyE FROM known WHERE Hash='{hashID}'")

        data = self.cursor.fetchall()

        if data != []:

            return data[0]

        else:

            return (None, None)

    def getLocation(self, NodeID):

        #parse get ExitIP request for a given NodeID

        #essentially same steps as above, but the NodeID itself is used since the number of online nodes is significantly less (maximise efficiency)

        self.cursor.execute(f"SELECT ExitIP, ExitPort FROM online WHERE NodeID='{NodeID}'")

        data = self.cursor.fetchall()

        if data != []:

            ipAddr = data[0]

        else:

            ipAddr = (None, None)

        self.cursor.execute(f"SELECT CircuitID FROM online WHERE NodeID='{NodeID}'")

        data = self.cursor.fetchall()

        try:

            CircuitID = data[0][0]

        except:

            CircuitID = None

        return [CircuitID, ipAddr]

    def getCircuitIP(self, CircuitID):

        #return the exit IP address for a given circuit

        #used by end nodes to send packets to other end nodes

        circuitHash = Cryptography.hash(CircuitID)

        self.cursor.execute(f"SELECT ExitIP, ExitPort FROM online WHERE Hash='{circuitHash}'")

        data = self.cursor.fetchall()

        if data != []:

            ipAddr = data[0]

        else:

            ipAddr = (None, None)

        return ipAddr

    def register(self, NodeID, PublicKey, CircuitID, EndIP):

        #registration subroutine

        self.cursor.execute(f"SELECT \* FROM known WHERE NodeID='{NodeID}'")

        data = self.cursor.fetchall()

        #check if the given NodeID already exists in the system

        if data != []:

            return False

            #already exists

        else:

            #doesn't exist, write data to database

            self.cursor.execute(f"""INSERT or IGNORE INTO known VALUES (

            '{Cryptography.hash(NodeID)}',

            '{NodeID}',

            {int(PublicKey[0])},

            {int(PublicKey[1])},

            1,

            {int(time.time())}

            )""")

            self.cursor.execute(f"""INSERT or IGNORE INTO online VALUES (

            '{Cryptography.hash(CircuitID)}',

            '{CircuitID}',

            '{EndIP[0]}',

            {EndIP[1]},

            '{NodeID}'

            )""")

            knownIPs.append(EndIP)

            self.connection.commit()

            return True

    def login(self, timestamp, NodeID, CircuitID, ExitIP, signature, toVerify):

        #login verification subroutine

        self.cursor.execute(f"SELECT \* FROM known WHERE NodeID='{NodeID}'")

        data = self.cursor.fetchall()

        key = self.getKeys(NodeID)

        if data == []:

            return False

        else:

            print(data)

            if Cryptography.verify(toVerify, signature, key) == True:

                #doesn't run unless the signature matches the public key stored in the database

                nodeIDHash = Cryptography.hash(NodeID)

                self.cursor.execute(f"""UPDATE known SET Online = 1,

                                        LastOnline = {int(timestamp)}

                                        WHERE Hash = '{nodeIDHash}'""")

                self.cursor.execute(f"""INSERT or IGNORE INTO online VALUES (

                '{Cryptography.hash(CircuitID)}',

                '{CircuitID}',

                '{ExitIP[0]}',

                {ExitIP[1]},

                '{NodeID}'

                )""")

                self.connection.commit()

                knownIPs.append(ExitIP)

                return True

            else:

                return False

DBQueue = LimitlessQueue()

#database processing queue

#since an error often occurs when two database tasks are carried out concurrently,

#every database request is sent to a queue data structure and is processed one by one

def serialDatabaseExecution():

    #serial task execution subroutine,

    #goes through the aforementioned queue and executes the commands

    while ProgramQuit == False:

        if DBQueue.isEmpty() == False:

            command = DBQueue.dequeue()

            #three types of tasks

            if command[0] == 'login':

                Transmission.login(command[1], command[2])

                time.sleep(1.5)

            elif command[0] == 'register':

                Transmission.register(command[1], command[2])

                time.sleep(1.5)

            elif command[0] == 'nodeInfo':

                Transmission.nodeInfo(command[1], command[2])

                time.sleep(1.5)

class Transmission():

    def interfaceInit():

        global choicesIP, sock, database, currentIP, tempCircuits

        validChoices = []

        print("Network Interface Selection")

        print("-=-=-=-=-=-=-=-=-=-=-=-=-=-")

        for x in range(len(choicesIP)):

            validChoices.append(str(x + 1))

            print(str(x + 1) + ". " + choicesIP[x])

        while True:

            choice = input("Please select the network interface you wish to use: ")

            if choice in validChoices:

                while True:

                    portChoice = input("Please enter the port you wish to use [Default: 107]: ")

                    try:

                        if portChoice == '':

                            portChoice = 107

                            break

                        else:

                            portChoice = int(portChoice)

                            break

                    except:

                        print("ERROR: Invalid Port Number")

                currentIP = (choicesIP[int(choice) - 1], portChoice)

                sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) #configures socket as UDP socket

                sock.bind(currentIP) #binds the specified IP address as the IP of the packet

                threading.Thread(target = Transmission.receiveAgent).start()

                break

            else:

                print("ERROR: Invalid Choice")

        print("-=-=-=-=-=-=-=-=-=-=-=-=-=-")

        print("")

        tempCircuits = [None] \* 60

        for x in range(25):

            #create (25) new temporary nodes for users to connect to

            newCircuit = TempCircuit()

            knownIPs.append(newCircuit.getIP())

            tempCircuits[x] = newCircuit

        database = Database()

    def moveCircuit(circuitID):

        global usedCircuits, tempCircuits

        #move a given temporary node from the temporary circuit array to the used one

        for x in range(len(tempCircuits)):

            if tempCircuits[x].getCircuitID() == circuitID:

                usedCircuits.append(tempCircuits.pop(x))

                break

    def send(msg, recipient):

        sock.sendto(msg, recipient)

    def receiveAgent():

        while True:

            try:

                data, sender = sock.recvfrom(1024)

                threading.Thread(target = Transmission.packetHandler, args=(data, sender)).start()

                if ProgramQuit == True:

                    break

            except:

                break

    def packetHandler(data, sender): #[timestamp, "DATA", destinationID, [msgID, circuitID, publicKey], payload, signature]

        global DBQueue

        data = pickle.loads(data)

        timestamp = data[0]

        header = data[1]

        if time.time() > timestamp + 100:

            pass #timeout, pass

        #circuit-routed packets

        #if the header matches any of the below, the request is added to the queue, DBQueue, for processing (see above for details)

        elif header == "LOGIN": #format: [timestamp, "LOGIN", "SERVER", [msgID, circuitID, publicKey], login\_payload, signature]

            DBQueue.enqueue(['login', data, sender])

        elif header == "REGISTER": #format: [timestamp, "REGISTER", "SERVER", [msgID, circuitID, publicKey], register\_payload, signature]

            DBQueue.enqueue(['register', data, sender])

        elif header == "NODEINFO": #format: [timestamp, "NODEINFO", "SERVER", [msgID, circuitID, publicKey], info\_payload, signature]

            DBQueue.enqueue(['nodeInfo', data, sender])

        #direct packets

        elif header == "TEST": #format: [timestamp, "TEST"]

            Transmission.test(sender)

        elif header == "GETNODES": #format: [timestamp, "GETNODES"]

            Transmission.getNodes(sender)

    def login(data, sender):

        #[timestamp, "LOGIN", "SERVER", [msgID, circuitID, publicKey], login\_payload, signature]

        #login\_payload(unencrypted) = NodeID

        login\_payload = data[4]

        NodeID = Cryptography.asymDecrypt(login\_payload, myPublicKey, myPrivateKey)

        CircuitID = data[3][1]

        key = data[3][2]

        signature = data.pop(5)

        result = database.login(data[0], NodeID, CircuitID, sender, signature, pickle.dumps(data))

        if result == True:

            #reply with success

            payload = ["SERVER", ["SERVERREPLY", Cryptography.asymEncrypt("LOGIN:SUCCESS", key)]]

        else:

            #reply with failure

            payload = ["SERVER", ["SERVERREPLY", Cryptography.asymEncrypt("LOGIN:FAIL", key)]]

        msg = [time.time(), "DATA", CircuitID, ["FROMSRV", "SERVER", myPublicKey], payload]

        tempMsg = pickle.dumps(msg)

        sig = Cryptography.sign(tempMsg, myPrivateKey)

        msg.append(sig)

        msg = pickle.dumps(msg)

        Transmission.send(msg, sender)

    def register(data, sender):

        #[timestamp, "REGISTER", "SERVER", [msgID, circuitID, publicKey], register\_payload, signature]

        #register\_payload(unencrypted) = NodeID

        register\_payload = data[4]

        NodeID = Cryptography.asymDecrypt(register\_payload, myPublicKey, myPrivateKey)

        CircuitID = data[3][1]

        key = data[3][2]

        result = database.register(NodeID, key, CircuitID, sender)

        if result == True:

            #reply with success

            payload = ["SERVER", ["SERVERREPLY", Cryptography.asymEncrypt("REGISTER:SUCCESS", key)]]

        else:

            #reply with failure

            payload = ["SERVER", ["SERVERREPLY", Cryptography.asymEncrypt("REGISTER:FAIL", key)]]

        msg = [time.time(), "DATA", CircuitID, ["FROMSRV", "SERVER", myPublicKey], payload]

        tempMsg = pickle.dumps(msg)

        sig = Cryptography.sign(tempMsg, myPrivateKey)

        msg.append(sig)

        msg = pickle.dumps(msg)

        Transmission.send(msg, sender)

    def getNodes(sender, internal=False):

        #run request to get nodes

        testprint("SENDING: GETNODE REPLY")

        NodeA = knownIPs[random.randint(0, len(knownIPs) - 1)]

        while True:

            NodeB = knownIPs[random.randint(0, len(knownIPs) - 1)]

            NodeC = knownIPs[random.randint(0, len(knownIPs) - 1)]

            if ((NodeA != NodeB) and (NodeA != NodeC) and (NodeB != NodeC)) or (len(knownIPs) < 3):

                break

        #iterates until a list of three individual nodes has been generated

        NodeList = [NodeA, NodeB, NodeC]

        for x in range(len(NodeList)):

            if NodeList[x] == sender:

                NodeList.pop(x)

                break

        if internal == False:

            #if the request was not sent internally, send the results back to the sender

            msg = [time.time(), "REPLY:GETNODES", NodeList]

            sig = Cryptography.sign(pickle.dumps(msg), myPrivateKey)

            msg.append(sig)

            msg = pickle.dumps(msg)

            Transmission.send(msg, sender)

            #knownIPs.append(sender)

        elif internal == True:

            #else simply return the nodelist though this subroutine (directly called by tempcircuit object)

            return NodeList

    def nodeInfo(data, sender):

        #format: [timestamp, "NODEINFO", "SERVER", [msgID, circuitID, publicKey], info\_payload, signature]

        #return information of a given node

        CircuitID = data[3][1]

        info\_payload = data[4]

        NodeID = Cryptography.asymDecrypt(info\_payload, myPublicKey, myPrivateKey)

        key = database.getKeys(NodeID)

        location = database.getLocation(NodeID)

        payload = [key, location[0], location[1]]

        payload = ["SERVER", ["INFO", payload]]

        msg = [time.time(), "DATA", CircuitID, ["FROMSRV", "SERVER", myPublicKey], payload]

        tempMsg = pickle.dumps(msg)

        sig = Cryptography.sign(tempMsg, myPrivateKey)

        msg.append(sig)

        msg = pickle.dumps(msg)

        Transmission.send(msg, sender)

    def test(sender):

        #return server name, announcements, and public key to a given sender

        testprint("SENDING: TEST REPLY")

        msg = [time.time(), "REPLY:TEST", serverName, announcements, myPublicKey]

        msg = pickle.dumps(msg)

        Transmission.send(msg, sender)

if \_\_name\_\_ == '\_\_main\_\_':

    try:

        globalConfig()

        serverName = input("Enter the name of the server: ")

        announcements = input("Enter any announcements you wish to send: ")

        threading.Thread(target = serialDatabaseExecution).start()

        Transmission.interfaceInit()

    except KeyboardInterrupt:

        exit()

[4] Testing

# 4.1 Interface Tests

## 4.1.1 Login Test

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test No. | Description | Entered Value | | Expected Outcome | | | Result |
| 1 | User Selection Menu | Erroneous: Blank Field | | ERROR: Command not recognised, please try again. | | | Expected |
| Comments: None | | |  | | | | |
| 2 | User Selection Menu | Erroneous: Invalid Option  Value: F | | ERROR: Command not recognised, please try again. | | | Expected |
| Comments: None | | |  | | | | |
| 3 | Server Selection Menu | Erroneous: Invalid IP Address  Value: ABC | | ERROR: Incorrect format for IP Address | | | Expected |
| Comments: IPv4 Address cannot contain any characters. Error shown even when there are four distinct sections (exception raised when trying to convert sections to integers). | | | | |  | | |
| 4 | Server Selection Menu | Erroneous: Invalid IP Address  Value: 192.168 | | ERROR: Incorrect format for IP Address | | | Expected |
| Comments: IP Address must be a valid IPv4 address (IPv6 not supported) and must have four distinct sections separated by dots. | | | | |  | | |
| 5 | Server Selection Menu | Erroneous: Invalid Port Number  Value: ABC | | ERROR: Incorrect format for IP Address | | | Expected |
| Comments: Port must be a valid integer (no specific range) | | | | | |  | |
| 6 | Server Selection Menu | Extreme: IP exceeds limit  Value: 256.256.256.256 | | ERROR: Incorrect format for IP Address | | | Expected |
| Comments: All sections of the IPv4 Address are checked to see whether or not they are between 0 and 255 (inclusive). | | | | | |  | |
| 7 | Server Selection Menu | Typical: Both IP and Port are valid  Value: 127.0.0.1/107 | | No error shown. New settings are saved to global config file. | | | Expected |
| Comments: Provided IP and PORT is saved to config file and is loaded when the program is terminated and re-executed. | | | | | |  | |

## 4.1.2 Console Test

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No. | Description | Entered Value | | Expected Outcome | Result |
| 8 | Console Input | Erroneous: Blank Field | | Do Nothing. | Expected |
| Comments: Similar to a normal operating system console, the protocol console doesn’t show any errors when a command isn’t entered. | | |  | | |
| 9 | Console Input | Erroneous: Invalid Option  Value: ABC | | ERROR: Command not recognised, please try again. | Expected |
| Comments: None | | |  | | |
| 10 | Console Argument | Erroneous: Invalid Argument  Value: show -a | | ERROR: Command not recognised, please try again. | Expected |
| Comments: None | | |  | | |
| 11 | Console Argument | Typical: Help Argument for command  Value: show -h | | Brings up help menu for command ‘show’. | Expected |
|  | | | | | |

## 4.1.3 Help Menu Test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description | Entered Value | Expected Outcome | Result |
| 12 | General Help | Typical: General help command  Value: help | Brings up general help menu. | Expected |
|  | | | | |
| 13 | General Help | Erroneous: Capitalised help command  Value: Help | ERROR: Command not recognised, please try again. | Expected |
| Comments: Like an operating system console, commands are case-sensitive. | | |  | |

## 4.1.4 Messenger Application Test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Desc. | Entered Value | Expected Outcome | Result |
| 14 | Start Chat | Erroneous: Blank Field  Value: Empty | Program prints help menu for the messenger/chat command | Expected |
| Comments: None | | |  | |
| 15 | Start Chat | Erroneous: Unknown User  Value:  ErroneousUser | Program prints error, where the given user hasn’t had their credentials accessed via the server | Expected |
| Comments: If there really was a user named ErroneousUser, the program would have to use ‘getinfo’ first to retrieve their exit IP before chatting with them | | |  | |
| 16 | Start Chat | Erroneous: Chat with self  Value:  Current Username | Program prints error along the lines of “Cannot chat with self!” | Expected |
| Comments: None | | |  | |
| 17 | Start Chat | Typical: Chat with known user  Value: Username of known user | Messenger module loads successfully, starting chat with provided Node. | Expected |
| Comments: Note how the getinfo command was used before trying to chat with ‘TestUserTwo’ | | |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 18 | Chat Actions | Typical: Run valid command with ‘/’ before  Value: ‘/help’ | Help menu for messenger command is printed.  The menu printed should differ from the one printed in the client file console. | Expected |
| Comments: Notice how the user input is overwritten by “Running command: command”, this was to ensure that input doesn’t interfere with the stream of incoming messages. | | |  | |
| 19 | Chat Actions | Erroneous: Blank Field and invalid command  Value: ‘/’ and “/errortest” | Messenger prints “Running command: (command)” but doesn’t print anything after that (returns to user input on new line). | Expected |
| Comments: Similar to above, and unlike the Hourglass console, the messenger does not display an error when an invalid command is entered – it simply doesn’t run. This was to reduce the amount of non-messaging output. | | |  | |

# 4.2 Algorithm Tests

## 4.2.1 Asymmetrical Encryption Test

## 4.2.2 Symmetrical Encryption Test

## 4.2.3 Hashing Test

## 4.2.4 Validation Test

# 4.3 Connection Tests

## 4.3.1 Circuit Establishment Test

## 4.3.2 Server Access Test

## 4.3.3 Recipient Access Test

[5] Evaluation

# 5.1 Objective Completion Evaluation

1. **Without leaving the scope of the A-level Computer Science syllabus, the protocol should strive for maximum possible anonymity of users.** 
   1. **Transmission of data across the network should be in such a way that reduces the ways in which the identity of users can be uncovered (e.g., using Onion Routing or Proxies)**
   2. **Metadata used within the protocol should be kept to a bare minimum; it must be implemented in such a way that it cannot be used to fingerprint a certain user (e.g., it should only contain things that require the protocol to function such as timestamps and destination IP addresses)**

OBJECTIVE 1) The final system utilised a custom variant of onion routing to transmit data between two given nodes on the network. Each node on the network connected to other nodes via a system of interconnected proxy nodes, storing the IP address of the end proxy to a centralised server. Other users may then send data to the user via this end IP.

The typical layout of a packet in the Hourglass Protocol is: [timestamp, header, destinationID, [msgID, circuitID, publicKey], payload, signature].

As seen above, the only data required is a UNIX timestamp, a generic header, the NodeID of the destination (DestinationID), and randomly generated MsgIDs and CircuitIDs.

Unlike the other systems we looked at in the analysis section of this documentation, such as Telegram’s MTProto 2.0 transmission system, the Hourglass protocol only uses metadata essential to the transmission of packets across the network.

1. **Data sent across the network using the protocol must be accessible to the recipient and the recipient alone.**
   1. **This can be achieved using a combination of asymmetric and symmetric encryption, as well as limiting the number of nodes that can access a particular packet in the network**

OBJECTIVE 2) Every data packet sent using the Hourglass protocol is encrypted using the RSA cryptosystem, meaning that only those with access to the recipient’s private key has access to the contents of a given packet.

As suggested in 2.a, packets are only accessible to four groups of people as highlighted below:

|**Sender**| => |**Nodes in the Sender’s Circuit**| => |**Nodes in the Recipient’s Circuit**| => |**Recipient**|

As discussed in the design section, the average number of nodes in a given circuit is 3 (calculated using E(X) for the geometric distribution), so the **maximum** **average** number of people who have access to a given packet is 6 (not including sender/recipient).

1. **Reliance on a centralised server should be kept to a bare minimum also; data should never have to pass through a central server to reach its destination.**
   1. **A central server should only serve the purpose of indicating whether or not a specific user is online, and what proxy they are accessible from**

OBJECTIVE 3) The Hourglass protocol uses a central server for the purpose of holding user credentials such as those outlined in 3.a (online status, end IP Address, public keys), and unlike conventional data transmission applications, data is never directly routed through the central server.

1. **Algorithms such as those for cryptography or data manipulation should be self-coded to the restraint that python allows; if required, certain functions can be programmed in languages such as C and then called by Python.**
   1. **For data manipulation, algorithms such as linear searching and number conversion should be self-coded rather than using the in-built python functions bin() and hex()**
   2. **For cryptography, algorithms such as asymmetric encryption, decryption, and key generation should be self-coded instead of using external modules**

OBJECTIVE 4) Rather than relying on libraries, I have coded algorithms for the following functions: prime generation, greatest common divisor calculation, linear search, decimal to binary conversion, binary to decimal conversion, base64 to decimal conversion, decimal to hexadecimal conversion, and database handling.

Other specialised algorithms include RSA key generation, encryption, and decryption; hashing, signing, and verification algorithms; etc.

1. **Instead of implementing a regular database, a hash table should be used server-side to maximise efficiency (considering that in a real-world scenario, a very large amount of people will be connecting to a server at once).**
   1. **The NodeID (the username) should be hashed to retrieve information from the database, rather than using the NodeID itself in an SQL query in order to maximise efficiency**

OBJECTIVE 5) Hash tables were used to store credentials in the known and online databases of the server program to minimise search times due to the use-case of this project (a large volume of credentials, with a large volume of people trying to access them).

### As a demonstration, a simple application of the protocol, such as a messenger function, should be implemented into the program to showcase an aspect of the various uses of the protocol.

### OBJECTIVE 6) In the final iteration of the program, a messenger module was created, allowing two users to participate in an instant message conversation via the Hourglass protocol. Due to issues with latency, users send message packets directly to the end IP of a given circuit in order to reduce the message lag by half – but also potentially reducing anonymity.

# 5.2 Interview with Stakeholders

# 5.3 Critique of Code Aspects

## 5.3.1 Functionality

## 5.3.2 Ease of Use

## 5.3.3 Reliability

## 5.3.4 Longevity and Futureproofing

# 5.4 Next Steps

[6] Appendix

# Questionnaires

Pre-Design Questionnaire for Communication System Project

Name: Aidan Goodwin

**Multiple Choice Questions:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **YES** | **NO** | **No Opinion** |
| Do you view yourself as a privacy-conscious person? | **X** |  |  |
| Are you familiar with the concept of *Peer-to-peer Networking*? |  | **X** |  |
| Are you familiar with the act of *torrenting* (a file)? | **X** |  |  |
| Have you heard of the term *Blockchain*? | **X** |  |  |
| Have you ever heard of *Bitcoin*? | **X** |  |  |
| Does it matter to you if someone has the ability to read your private messages? | **X** |  |  |
| Do you trust companies such as *Google* and *Facebook* to use your data properly? |  | **X** |  |
| Do you trust your government enough to have access to all your personal and private data? |  |  | **X** |

Pre-Design Questionnaire for Communication System Project

Name: *Wishes to Remain Anonymous*

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **YES** | **NO** | **No Opinion** |
| Do you view yourself as a privacy-conscious person? |  | **X** |  |
| Are you familiar with the concept of *Peer-to-peer Networking*? | **X** |  |  |
| Are you familiar with the act of *torrenting* (a file)? | **X** |  |  |
| Have you heard of the term *Blockchain*? | **X** |  |  |
| Have you ever heard of *Bitcoin*? | **X** |  |  |
| Does it matter to you if someone has the ability to read your private messages? |  |  | **X** |
| Do you trust companies such as *Google* and *Facebook* to use your data properly? |  |  | **X** |
| Do you trust your government enough to have access to all your personal and private data? |  |  | **X** |

**Multiple Choice Questions:**

Pre-Design Questionnaire for Communication System Project

Name: Steven Carey

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **YES** | **NO** | **No Opinion** |
| Do you view yourself as a privacy-conscious person? | **X** |  |  |
| Are you familiar with the concept of *Peer-to-peer Networking*? |  | **X** |  |
| Are you familiar with the act of *torrenting* (a file)? |  | **X** |  |
| Have you heard of the term *Blockchain*? |  | **X** |  |
| Have you ever heard of *Bitcoin*? | **X** |  |  |
| Does it matter to you if someone has the ability to read your private messages? | **X** |  |  |
| Do you trust companies such as *Google* and *Facebook* to use your data properly? |  | **X** |  |
| Do you trust your government enough to have access to all your personal and private data? |  | **X** |  |

**Multiple Choice Questions:**

Pre-Design Questionnaire for Communication System Project

Name: *Wishes to Remain Anonymous*

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **YES** | **NO** | **No Opinion** |
| Do you view yourself as a privacy-conscious person? |  |  | **X** |
| Are you familiar with the concept of *Peer-to-peer Networking*? |  | **X** |  |
| Are you familiar with the act of *torrenting* (a file)? |  | **X** |  |
| Have you heard of the term *Blockchain*? |  | **X** |  |
| Have you ever heard of *Bitcoin*? | **X** |  |  |
| Does it matter to you if someone has the ability to read your private messages? |  |  | **X** |
| Do you trust companies such as *Google* and *Facebook* to use your data properly? |  |  | **X** |
| Do you trust your government enough to have access to all your personal and private data? |  |  | **X** |

**Multiple Choice Questions:**

Pre-Design Questionnaire for Communication System Project

Name: Rakesh Balasubramanian

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **YES** | **NO** | **No Opinion** |
| Do you view yourself as a privacy-conscious person? | **X** |  |  |
| Are you familiar with the concept of *Peer-to-peer Networking*? | **X** |  |  |
| Are you familiar with the act of *torrenting* (a file)? | **X** |  |  |
| Have you heard of the term *Blockchain*? | **X** |  |  |
| Have you ever heard of *Bitcoin*? | **X** |  |  |
| Does it matter to you if someone has the ability to read your private messages? | **X** |  |  |
| Do you trust companies such as *Google* and *Facebook* to use your data properly? |  | **X** |  |
| Do you trust your government enough to have access to all your personal and private data? |  | **X** |  |

**Multiple Choice Questions:**

Pre-Design Questionnaire for Communication System Project

Name: Quinn Marshall

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **YES** | **NO** | **No Opinion** |
| Do you view yourself as a privacy-conscious person? | **X** |  |  |
| Are you familiar with the concept of *Peer-to-peer Networking*? | **X** |  |  |
| Are you familiar with the act of *torrenting* (a file)? | **X** |  |  |
| Have you heard of the term *Blockchain*? | **X** |  |  |
| Have you ever heard of *Bitcoin*? | **X** |  |  |
| Does it matter to you if someone has the ability to read your private messages? | **X** |  |  |
| Do you trust companies such as *Google* and *Facebook* to use your data properly? |  | **X** |  |
| Do you trust your government enough to have access to all your personal and private data? |  |  | **X** |

**Multiple Choice Questions:**

Pre-Design Questionnaire for Communication System Project

Name: *Wishes to Remain Anonymous*

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **YES** | **NO** | **No Opinion** |
| Do you view yourself as a privacy-conscious person? |  | **X** |  |
| Are you familiar with the concept of *Peer-to-peer Networking*? |  | **X** |  |
| Are you familiar with the act of *torrenting* (a file)? |  | **X** |  |
| Have you heard of the term *Blockchain*? |  | **X** |  |
| Have you ever heard of *Bitcoin*? |  | **X** |  |
| Does it matter to you if someone has the ability to read your private messages? |  | **X** |  |
| Do you trust companies such as *Google* and *Facebook* to use your data properly? | **X** |  |  |
| Do you trust your government enough to have access to all your personal and private data? | **X** |  |  |

**Multiple Choice Questions:**

Pre-Design Questionnaire for Communication System Project

Name: Marko Drescher

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **YES** | **NO** | **No Opinion** |
| Do you view yourself as a privacy-conscious person? | **X** |  |  |
| Are you familiar with the concept of *Peer-to-peer Networking*? | **X** |  |  |
| Are you familiar with the act of *torrenting* (a file)? | **X** |  |  |
| Have you heard of the term *Blockchain*? | **X** |  |  |
| Have you ever heard of *Bitcoin*? | **X** |  |  |
| Does it matter to you if someone has the ability to read your private messages? |  |  | **X** |
| Do you trust companies such as *Google* and *Facebook* to use your data properly? |  | **X** |  |
| Do you trust your government enough to have access to all your personal and private data? |  |  | **X** |

**Multiple Choice Questions:**

Pre-Design Questionnaire for Communication System Project

Name: Kasib Abdul-Quddus

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **YES** | **NO** | **No Opinion** |
| Do you view yourself as a privacy-conscious person? | **X** |  |  |
| Are you familiar with the concept of *Peer-to-peer Networking*? | **X** |  |  |
| Are you familiar with the act of *torrenting* (a file)? | **X** |  |  |
| Have you heard of the term *Blockchain*? | **X** |  |  |
| Have you ever heard of *Bitcoin*? | **X** |  |  |
| Does it matter to you if someone has the ability to read your private messages? | **X** |  |  |
| Do you trust companies such as *Google* and *Facebook* to use your data properly? |  | **X** |  |
| Do you trust your government enough to have access to all your personal and private data? |  | **X** |  |

**Multiple Choice Questions:**

Pre-Design Questionnaire for Communication System Project

Name: *Wishes to Remain Anonymous*

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **YES** | **NO** | **No Opinion** |
| Do you view yourself as a privacy-conscious person? | **X** |  |  |
| Are you familiar with the concept of *Peer-to-peer Networking*? | **X** |  |  |
| Are you familiar with the act of *torrenting* (a file)? | **X** |  |  |
| Have you heard of the term *Blockchain*? |  | **X** |  |
| Have you ever heard of *Bitcoin*? | **X** |  |  |
| Does it matter to you if someone has the ability to read your private messages? | **X** |  |  |
| Do you trust companies such as *Google* and *Facebook* to use your data properly? |  | **X** |  |
| Do you trust your government enough to have access to all your personal and private data? |  |  | **X** |

**Multiple Choice Questions:**

Pre-Design Questionnaire for Communication System Project

Name: Hollie Butler

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **YES** | **NO** | **No Opinion** |
| Do you view yourself as a privacy-conscious person? |  | **X** |  |
| Are you familiar with the concept of *Peer-to-peer Networking*? |  | **X** |  |
| Are you familiar with the act of *torrenting* (a file)? |  | **X** |  |
| Have you heard of the term *Blockchain*? |  | **X** |  |
| Have you ever heard of *Bitcoin*? | **X** |  |  |
| Does it matter to you if someone has the ability to read your private messages? | **X** |  |  |
| Do you trust companies such as *Google* and *Facebook* to use your data properly? |  |  | **X** |
| Do you trust your government enough to have access to all your personal and private data? |  | **X** |  |

**Multiple Choice Questions:**