Software Engineering Department
ORT Braude Academic College

Capstone Project – Phase B

Transfer large files over the internet in secure and compress manner

Project number: 23-1-D-27

Authors:

1. ID: 314660002, Full name: Bashar Bashir, Email: Basharjob1@gmail.com
2. ID: 207734195, Full name: Mohamed Soboh, Email: Mohammed.s@outlook.co.il

Supervisor:

Full Name: Mr. Ronen Zilber
Email address: rzilber1@gmail.com

GitHub link: https://github.com/BasharBashir/Fast-Secure-File-Transfer

Contents

1.1 Introduction and Problem Formulation

The information transfer in this project will refer to the Internet network as it reflects the problems of transmission rate limitations and exposure to cyber hacks. There are various ways to transfer a file from one computer to another, such as using medium storage and web applications.
Files are not secure to be transferred because there are so many security threats that need to be considered while sending files over a network. Hackers can use many ways to get the content of the file.One of the solutions to secure communication is cryptography.
There is a large size that can hinder data transmission quickly and save on existing storage in the computer. To overcome the problem of information or data to be transmitted can be done more quickly than required compression that can save storage and transmission of data to be done.
To improve the speed of transferring large files, we can fragment the file into smaller chunks and send these chunks simultaneously from multiple clients. This process, known as parallel transfer, allows for faster transfer times as the workload is distributed among multiple sources. By fragmenting the file and utilizing parallel transfer, we can significantly reduce the time it takes to transfer large files.

This project presents the development of a new file transfer application that uses the TCP protocol with sliding window technology. The application is designed for client-server communication and aims to provide a fast, reliable, and secure way to transfer files. The project is divided into two parts. In the first part, a client requests a file from other connected clients and establishes peer-to-peer connections with those clients to retrieve the file. The client also determines the bandwidth available for file transfer and requests the file in chunks based on this information. The connected clients encrypt and send the requested chunks to the client, who then verifies the integrity of the received file. In the second part of the project, a client sends a file to all connected clients using peer-to-peer connections. The file is first compressed to reduce its size, and then processed and encrypted before being sent to the clients. Each client receives a different chunk of the file and immediately sends it to the other clients, while also processing and decrypting the received chunk. The sender and receiver clients verify the transferred file using a secure hashing algorithm. Overall, this project aims to provide a reliable and efficient way to transfer files in a client-server environment, utilizing the benefits of peer-to-peer communication.

1.2 Related Work

1.2.1 Compression

Compression is the art of representing the information in a compact form rather than its original or uncompressed form. In other words, using the data compression, the size of a particular file can be reduced. This is very useful when processing, storing or transferring a huge file, which needs lots of resources.When data compression is used in a data transmission application, speed is the primary goal. Speed of transmission depends upon the number of bits sent, the time required for the encoder to generate the coded message and the time required for the decoder to recover the original ensemble.

1.2.2 Cryptography

Cryptography is one of the most significant and popular techniques to secure the data from attackers by using two vital processes that are Encryption and Decryption. Encryption is the process of encoding data to prevent intruders from reading the original data easily. This stage has the ability to convert the original data (Plaintext) into an unreadable format known as Cipher text. The next process that has to be carried out by the authorized person is Decryption. Decryption is contrary to encryption. It is the process to convert cipher text into plain text without missing any words in the original text. To perform this process cryptography relies on mathematical calculations along with some substitutions and permutations with or without a key.
These days, there are a number of algorithms to encrypt and decrypt sensitive data which are typically divided into three types. First one is symmetric cryptography, which is the same key that is used for encryption and decryption data. Second one is Asymmetric cryptographic. This type of cryptography relies on two different keys for encryption and decryption. Finally, a cryptographic hash function using no key instead key is mixed with the data[1].
A cryptographic hash function is a mathematical function that takes an input of any size and produces an output of a fixed size, known as a "hash value" or "message digest". The main property of a cryptographic hash function is that it is practically impossible to generate the same hash value from two different inputs, even if the inputs are only slightly different. This makes cryptographic hash functions useful for a variety of applications, including data integrity, authentication, and digital signatures.

1.2.3 Simultaneous Transfer

Simultaneous transfer, also known as parallel transfer, is a method used to optimize the speed and efficiency of data transfer operations. It involves dividing the data into

smaller chunks and sending these chunks concurrently, rather than sequentially. This technique can be applied to a variety of data transfer scenarios, including the transfer of large files over a network, communication between devices, and the transfer of data between systems. The use of simultaneous transfer can greatly reduce the time required to complete a transfer and improve the overall performance of the system by distributing the workload among multiple sources.

1.2.4 Transport Layer protocols

Transport Layer protocols are responsible for the end-to-end delivery of data packets between devices on a network. They provide services such as error checking, flow control, and congestion avoidance to ensure reliable and efficient data transfer. There are three main Transport Layer protocols: UDP, TCP, and TCP with a sliding window.

TCP, or Transmission Control Protocol is a connection-oriented protocol that ensures the reliable transmission of data, uses an ARQ method called a "three-way handshake" to establish a connection between devices and ensure reliable transmission of data. In this method, the sender sends a packet requesting a connection, the receiver acknowledges receipt of the packet, and the sender confirms the establishment of the connection. While this ensures reliable transmission, it can also slow down the communication process.

TCP with sliding window is a variation of TCP that uses a sliding window protocol to increase the efficiency of data transmission. In this method, the sender can transmit multiple packets without waiting for an acknowledgement from the receiver, as long as the total number of unacknowledged packets does not exceed the size of the sliding window. This allows for faster transmission of data, as the sender does not have to wait for an acknowledgement before sending each packet.

UDP, or User Datagram Protocol, is a connectionless protocol that does not use an Automatic Repeat Request (ARQ) method for error detection and correction. This means that if a packet is lost or corrupted during transmission, there is no mechanism in place to detect or correct the error. While this makes UDP faster than TCP, it also means that it is less reliable. It is often used for real-time applications such as online gaming or voice over IP (VoIP) where a small amount of lost data may not be noticeable.
UDP is fast but unreliable, TCP with a sliding window provides a balance between speed and reliability.

1.2.5 Sockets

A socket is a software endpoint that establishes a bidirectional communication channel between two computers over a network. It allows a client to send data to a

server and receive data back in response. Sockets can be used to establish a connection between a client and a server in a client-server architecture, or between two clients in a peer-to-peer architecture.

In the context of a client-server architecture, the client initiates a request for data or service to the server, and the server responds with the requested data or performs the requested service. The client and server communicate with each other using a predetermined protocol, such as TCP or UDP, which defines the format of the data and the rules for exchanging messages. Sockets are a powerful tool for building networked applications, as they provide a simple and efficient way to establish and maintain communication between clients and servers.

Multithreading can be used to improve the performance of a socket-based application by allowing multiple threads to handle incoming connections and data transfer concurrently. This can be particularly useful in a peer-to-peer application, where multiple clients may be sending data at the same time.
To use multithreading with sockets in a peer-to-peer application, the application can create a separate thread for each outgoing connection, with each thread responsible for managing the data transfer for a specific peer. This allows the application to send data to multiple peers concurrently without the overhead of creating a separate thread for each one.
Using multithreading and multiple sockets can significantly improve the performance of a socket-based application by allowing it to send and receive data from multiple peers concurrently. This can be particularly useful in situations where the application needs to send large amounts of data or where the network connection is slow.

1.3 Proposed Method

1.3.1 AES Algorithm

AES is a strong encryption technique. It is resistant to all known attacks and is efficient across a wide range of platforms. AES encrypts data for transmission and decrypts it for receipt, preventing hackers from intercepting the transmission with packet sniffers[2].

AES Algorithm Structure

AES is an iterative cipher, using substitution and permutation networks to encrypt and decrypt data[3]. It operates on a fixed block size of 128 bits (16 bytes) in a 4x4 matrix and the number of rounds used depends on the key size: 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys[4].

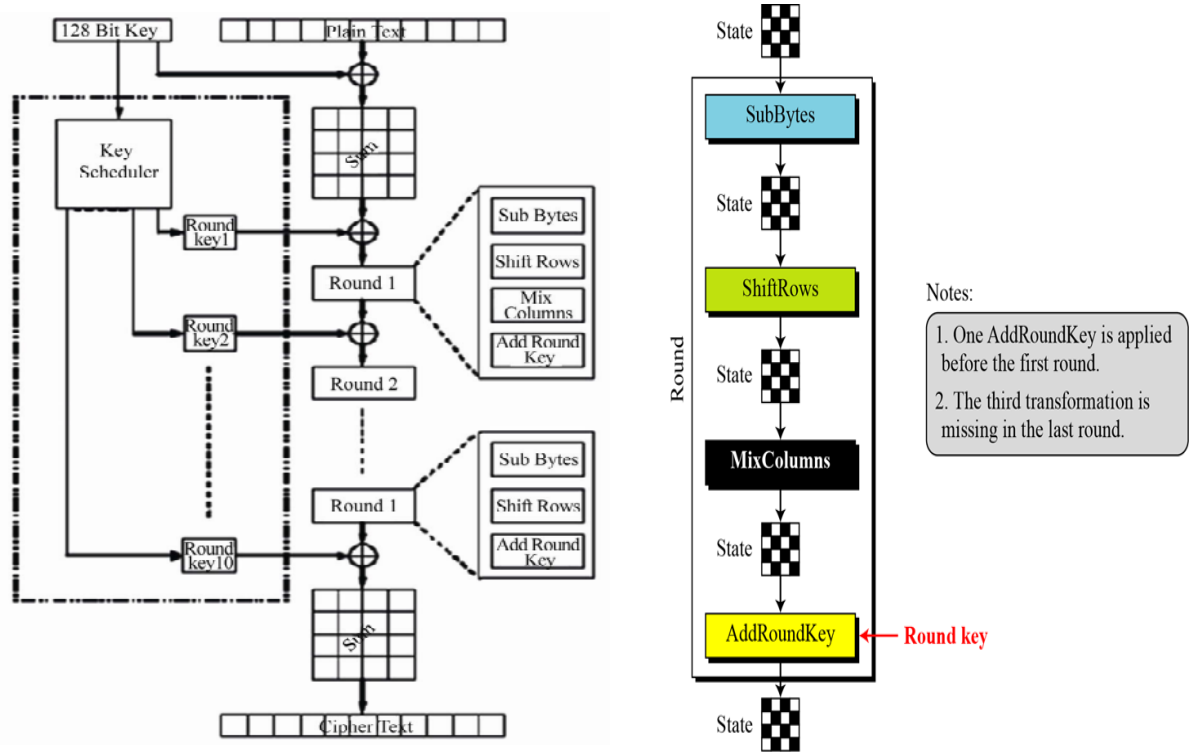


Fig. 1 AES Algorithm Structure

Electronic Code Book

The ECB mode is the simplest mode of operation for encryption[5], where the message is divided into blocks of equal length and each block is encrypted separately using the same key[5]. The encryption algorithm is Cj = EK(Pj) and decryption algorithm is Pj = DK(Cj), where EK is the encryption map with the key (K) and DK is decryption map with the same key (K)[6].

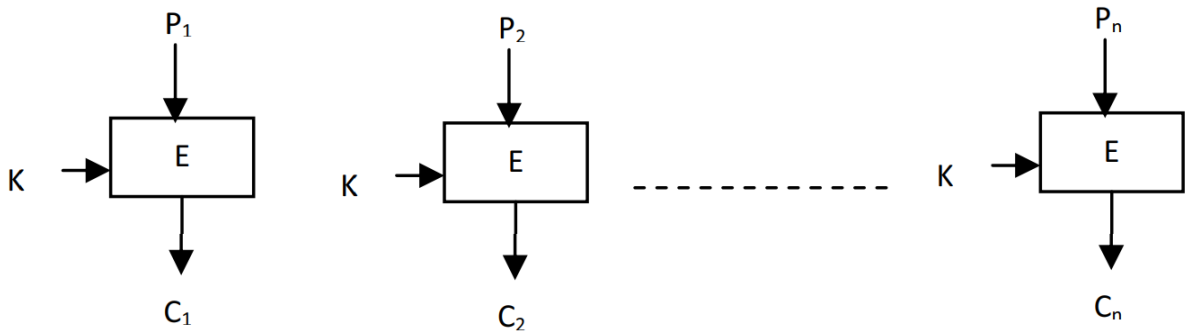


Fig. 2 ECB Structure

The size of the blocks is determined by the block size of the encryption algorithm being used. If the data is not evenly divisible, the remaining bits may be padded to complete the final block.

1.3.1.2 Encryption Process

Encryption is a popular technique that plays a major role to protect data from intruders. AES algorithm uses a particular structure to encrypt data to provide the best security. To do that it relies on a number of rounds and inside each round comprise of four sub-process. Each round consists of the following four steps to encrypt 128 bit block. There are four steps in AES algorithm

Subbytes

SubBytes is a transformation in AES encryption where a byte is substituted using an ASCII lookup table. The left digit of the hexadecimal representation of the byte defines the row, and the right digit defines the column of the substitution table. The state is treated as a 4x4 matrix of bytes and each byte is transformed independently. There are 16 distinct byte-to-byte transformations in the SubBytes process.

(row index) (column index) S-Box *GF* (2')

0011	0010

32	88	31	e0
43	5a	31	37
f6	30	98	07
a8	8d	a2	34

16 times

32	c4	c7	e1
1a	be	c7	9a
42	04	46	c5
c2	5d	3a	18

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

x

Fig. 3 S-Box Table

Shift Row

Each of the four rows of the matrix are shifted to the left. Any entries that ‘fall off’ are re-inserted on the right side of the row. The shift is carried out as follows

- First row is not shifted.
- Second row is shifted one (byte) position to the left.
- Third row is shifted two positions to the left
- Fourth row is shifted three positions to the left.

The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

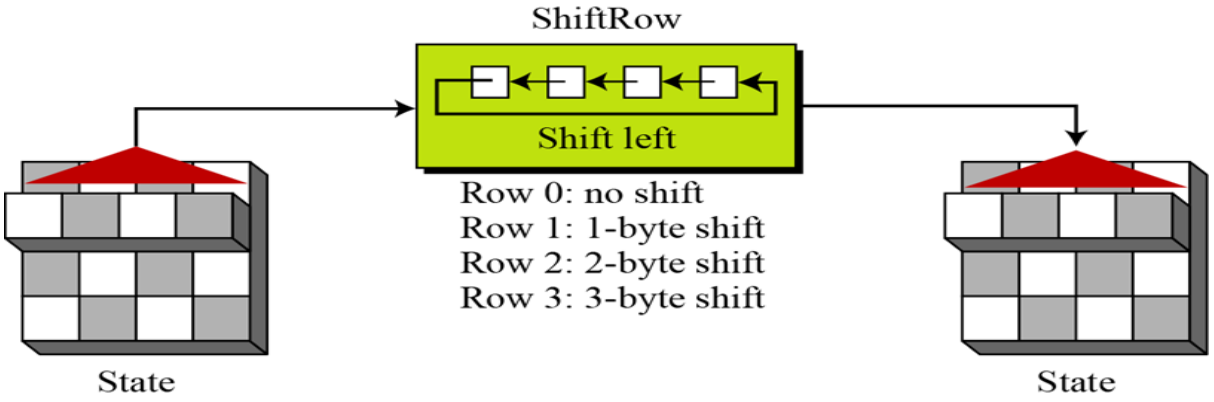


Fig. 4 Shift Row

Mix Columns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. The bytes that are in the state column and constant matrix are interpreted as 8 bits or polynomial with coefficient GF(2). The procedure is repeated with the next column of the state until there are no more state column. It should be noted that this step is not performed in the last round.

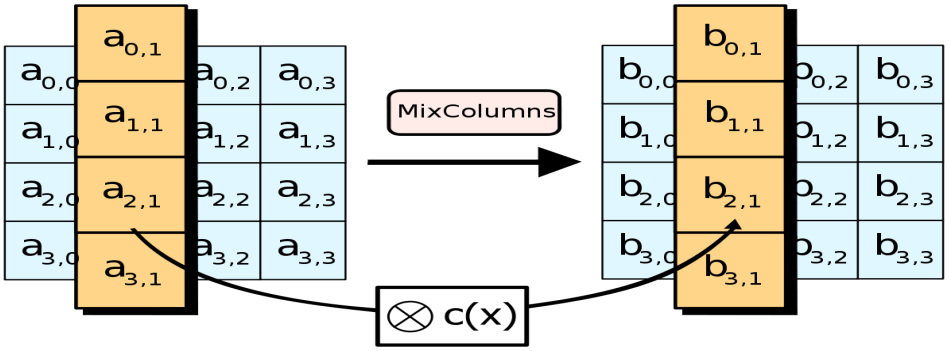


Fig. 5 Mix Columns

Add Round Key

In the AddRoundKey step, each byte of the state is combined with a byte of the round subkey using XOR operation. A subkey is derived from the main key Rijndael's key schedule.

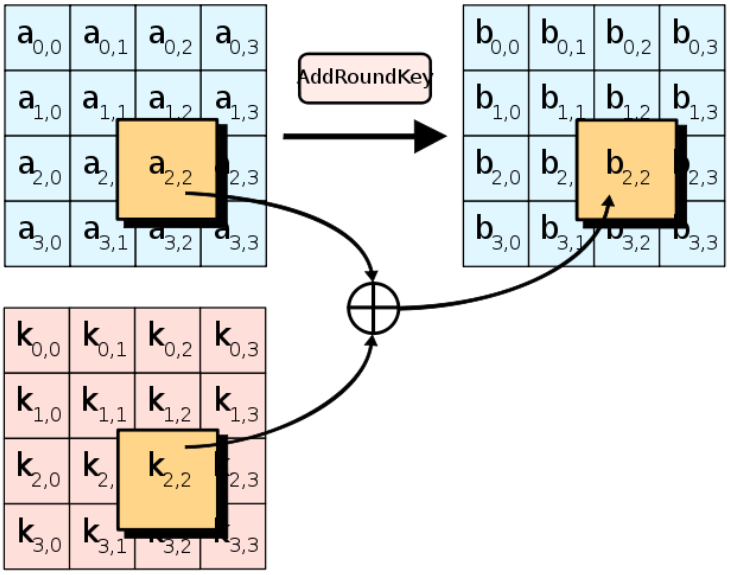


Fig. 6 Add Round Key

1.3.1.3 Decryption Process

The decryption is the process to obtain the original data that was encrypted. This process is based on the key that was received from the sender of the data. The decryption processes of an AES is similar to the encryption process in the reverse order and both sender and receiver have the same key to encrypt and decrypt data. The last round of a decryption stage consists of three stages such as InvShiftRows, InvSubBytes, and AddRoundKey as illustrated in Fig. 7.

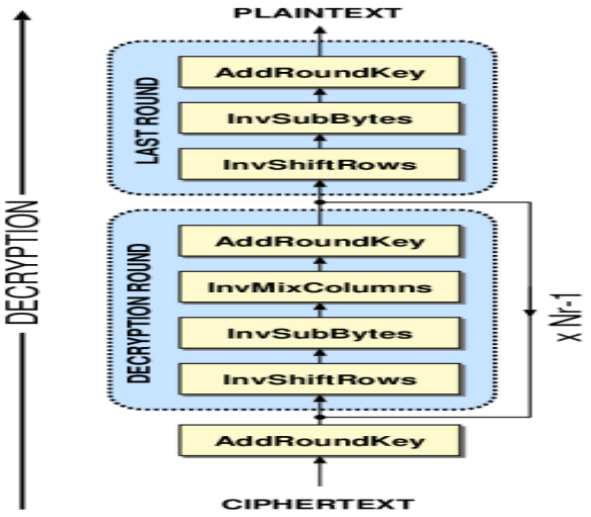


Fig. 7 Decryption Processes

1.3.1.4 AES Key Expansion

The AES Key Expansion is an important step in AES encryption that generates round keys word by word. The key expansion routine creates 4x(Nr+1) words, where Nr is the number of rounds. The process starts by using the initial key (cipher key) to create the first four words. The key size consists of 16 bytes, which are divided into 4 words of 4 bytes each. Subsequent words are generated using the previous word and a particular equation. A different equation is used for the first word (w0). The key for each round is generated using the equation: K[n]: w[i] = k[n-1]: w[i] XOR k[n]: w[i].

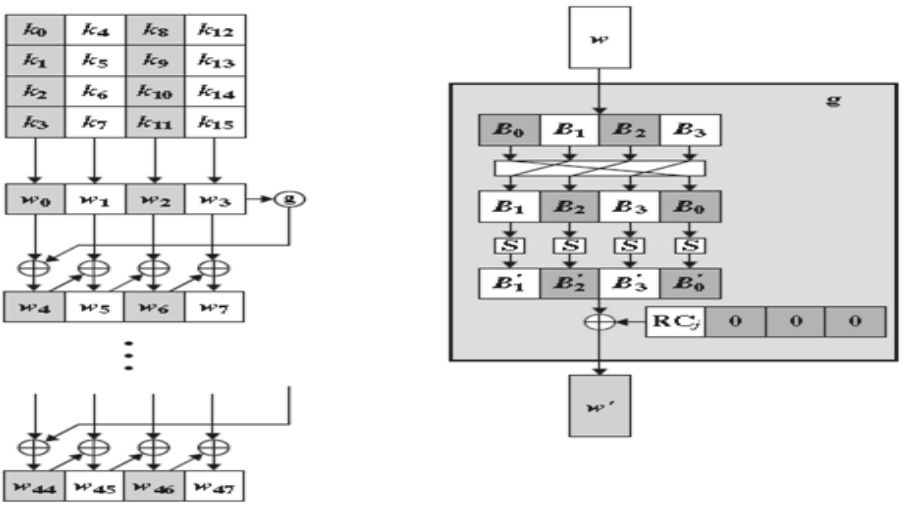


Fig. 8 AES Key Expansion

1.3.2 Diffie Hellman Key Exchange

Diffie-Hellman key exchange algorithm is a method for establishing a shared secret between two parties (Alice and Bob) over an unprotected communications channel. It was invented in 1976 by Whitfield Diffie and Martin Hellman. It uses the multiplicative group of integers modulo p, where p is a prime number. The two parties (Alice and Bob) choose two numbers p and g, where p is a prime number and g is a primitive root of order (p-1) known as the generator[7]. These numbers are public and can be sent over the internet.

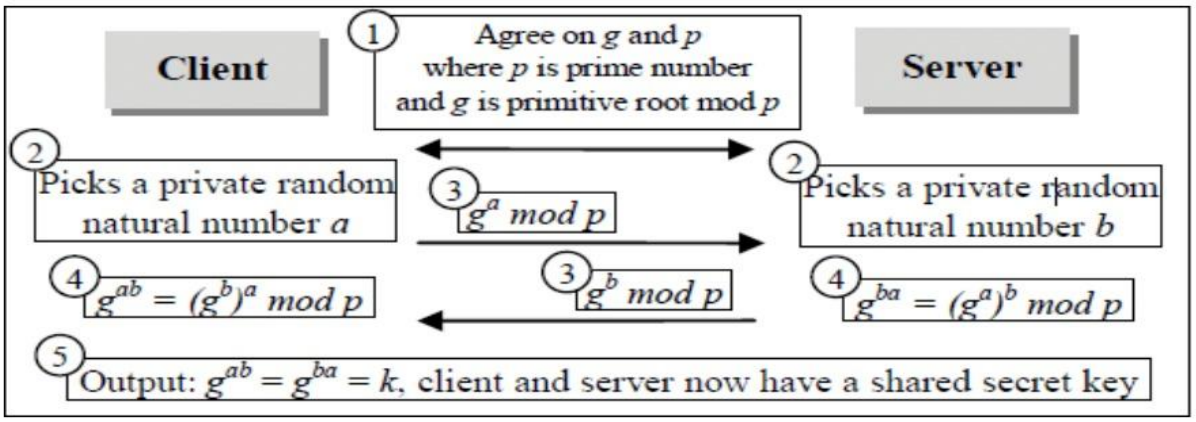


Fig. 9 Diffie-Hellman Exchange

1.3.3 Secure Hash Algorithm 256-Bit

SHA-256 is a cryptographic hash function that is used to verify the integrity of data by taking an input of any size and producing a fixed-size output, known as a "hash value" or "message digest", that is unique to that input. It is a one-way function, meaning any change to the input will result in a different output. It is one of the variants of the Secure Hash Algorithm (SHA) developed by NIST for use in secure communication systems. It is widely used in various applications such as digital signatures, message authentication, and random number generation, and considered to be very secure.

SHA-256 Algorithm Structure

SHA or the secure hash algorithm is used in digital signatures and password storage[8].
This hash function takes a string and outputs a hash value that is 256 bits long. The whole algorithm can be divided into four parts.

Insert Padding bits

The SHA-256 algorithm works on the block of 512 bits which is the standard length for this particular algorithm. The padding is done by adding some extra bits to the original message. The appending starts with 1 and the other bits following it are zero. The last 64 bits is left out of multiple of 512 which is filled in the next step. The length of the original message is M, while the number of padding bits appended or added to the initial message is L. From the following equation, the number of padding bits can be calculated.

$$M + L + 64 = n \times 512$$

Add Length Bits

The appending to the original message is done in the first step, and the remaining 64 bits are added in this step. The original message length multiplied by 8 (8 bit ASCII) is added in binary to the padded message. The message has a length that is a multiple of 512 bits. The big-endian convention used in the algorithm which indicates the left most bit is stored in the most significant bit position.

Buffer Initialization

The predefined values are initialized which is used in the algorithm that is implemented in the next step. The initialization includes eight hash values and 63 keys. The 64 key values are used in the 64 rounds of SHA-256 algorithm.

- Hash_1 = 0x6a09e667
- Hash_2 = 0xbb67ae85

- Hash_3 = 0x3c6ef372
- Hash_4 = 0xa54ff53a
- Hash_5 = 0x510e527f
- Hash_6 = 0x9b05688c
- Hash_7 = 0x1f83d9ab
- Hash_8 = 0x5be0cd19

1.3.3.1 Compression Algorithm

The padded message and the default predefined values are used in this step, and this is the important part of the hashing algorithm. The whole message is divided into “N” 512 bits blocks and passed the block into a compression algorithm. The block of 512 bits undergoes a 64 round (Fig. 10).
Each round takes 32 bit words [i] and key [i] as input. For the first sixteen rounds, 512 bit block is divided into 16 blocks of 32 bits. These 16 blocks of 32 bits act as input for the first sixteen rounds. The words[i] for the remaining rounds are calculated using the following formulae.

$$Wj = \sigma 1\ Wj{-}2 + Wj{-}7 + \sigma 0\ Wj{-}15 + Wj{-}16$$

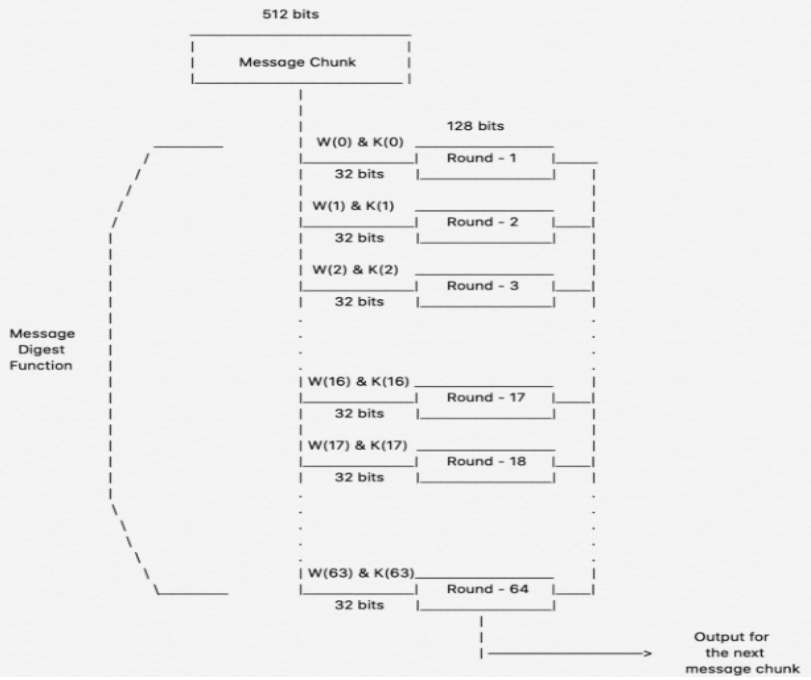


Fig. 10 Algorithm structure

Total of six logical functions are used in the SHA-256 algorithm. The logical functions are defined below.

- ch(x,y,z)=(x&y)⊕(~x&z)
- Maj(x,y,z)=(x&y)⊕(x⊕z)⊕(y&z)

- $\Sigma(x)=S^{2}(x) \oplus S^{13} \oplus S^{22}$:x=0
- $\Sigma(x)=S^{6}(x) \oplus S^{11} \oplus S^{25}$:x=1
- $\sigma o=S^{7} \oplus S^{18} \oplus R^{3}$
- $\sigma 1=S^{17} \oplus S^{19} \oplus R^{10}$

In the last 4 points, S means rotate right x by n bits. The above functions are used in each of the 64 rounds, and it is also applied on “N” 512 bits block. The below image shows the operations happening in each round (Fig. 2).
We get eight hash values by processing one 512 bits block, the result we obtained is added with the previous hash value, and it is given as input to the next round.

Hash1i = a + Hash1i - 1
Hash2i = a + Hash2i - 1
Hash3i = a + Hash3i - 1
.
.
.
Hash8i = a + Hash8i - 1

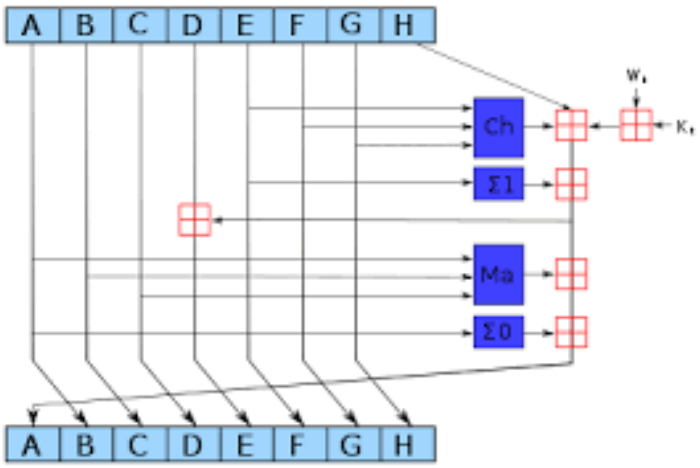


Fig. 11 Each round description[9]

The process of adding hash with the previous one keeps continuing till the last bit of the message. The result of the last round of the Nth 512 bits message gives the result. The final result is 256 bits.

1.3.4 Lempel-Ziv Welch Algorithm

Lempel-Ziv-Welch (LZW) algorithm is a method for compressing files by identifying and replacing repeated sequences of data. This results in smaller file sizes and can save time and bandwidth when transferring files.
The Lempel-Ziv-Welch (LZW) algorithm works by creating a dictionary of strings, where each string is a sequence of characters found in the input data. The algorithm starts by initializing the dictionary with the individual characters of the input data as the dictionary's keys and assigns them a unique code. As the algorithm reads through the input data, it looks for repeating sequences of characters. When a repeating sequence is found, the algorithm adds the sequence as a new key to the dictionary and assigns it a unique code. This new key-value pair is then used to replace the original repeating sequence in the input data, thus reducing the overall number of characters in the data. This process continues until the end of the input data is reached. The resulting data, now containing only the unique codes, is smaller in size than the original data and can be decompressed back to its original form using the same dictionary.

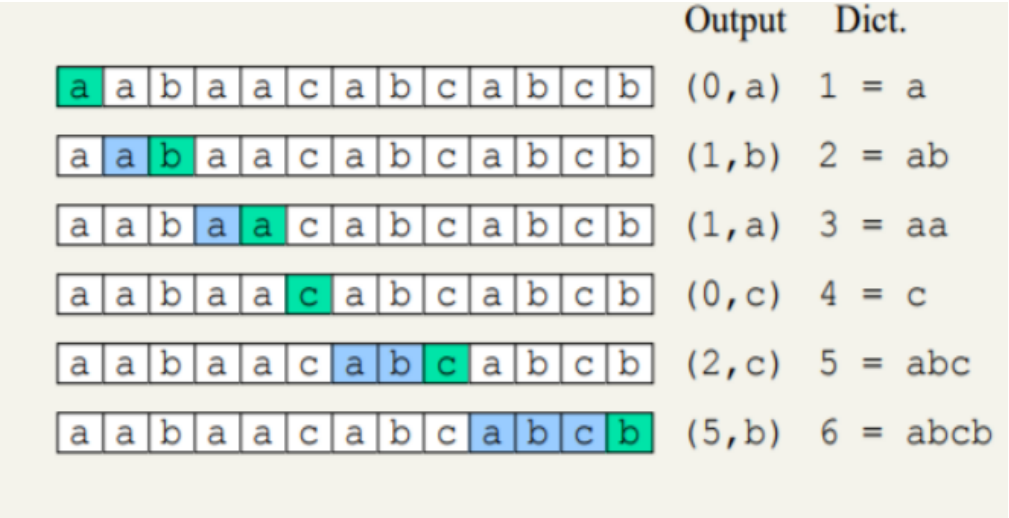


Fig. 12 LZW Dictionary

The decompression process involves reading the compressed data, looking up the corresponding uncompressed data in the dictionary, and outputting the uncompressed data. The dictionary used in the decompression process is the same one that was used in the compression process, and it is typically stored along with the compressed data. Once the data is uncompressed, it should be identical to the original, pre-compressed data.

1.4 Implementation

1.4.1 Process

The proposed file transfer application aims to provide fast, reliable and secure file transfer by utilizing TCP with sliding window protocol and a client-server architecture. The project is divided into two main parts: requesting a file and sending a file.

In the first part, the client requests the IP addresses of clients who possess the file from the server. The server management data structure contains IP addresses of all the clients, files each client owns, and the number of chunks each file is divided into. The requesting client also requests the number of chunks of the file from the server that he intends to request. The client then establishes connections with these clients and uses the ping time to determine the best way to request chunks of the file. To ensure efficient transfer, the client creates multiple sockets and connects them to the clients who possess the file. The connected clients encrypt their respective chunks before sending them to the requesting client. Once all the chunks are received, the client verifies the completeness of the file and requests any missing chunks. Finally, the client decrypts the received chunks.

In the second part, the client who wants to send a file requests IP addresses of the connected clients from the server. All clients establish connections with each other in a peer-to-peer fashion using sockets. The sender client uses the Lempel-Ziv-Welch (LZW) algorithm to compress the file and then fragments it into labeled chunks. The sharing client updates the server and the clients with the number of chunks the file has been divided into. The chunks are encrypted using the AES algorithm in ECB mode. The sender then sends the chunks to the clients based on the chunk number modulo the number of clients. Each receiver client sends the received chunks to the other clients. Every client then assembles and decrypts the received chunks and verifies the files using SHA-256.

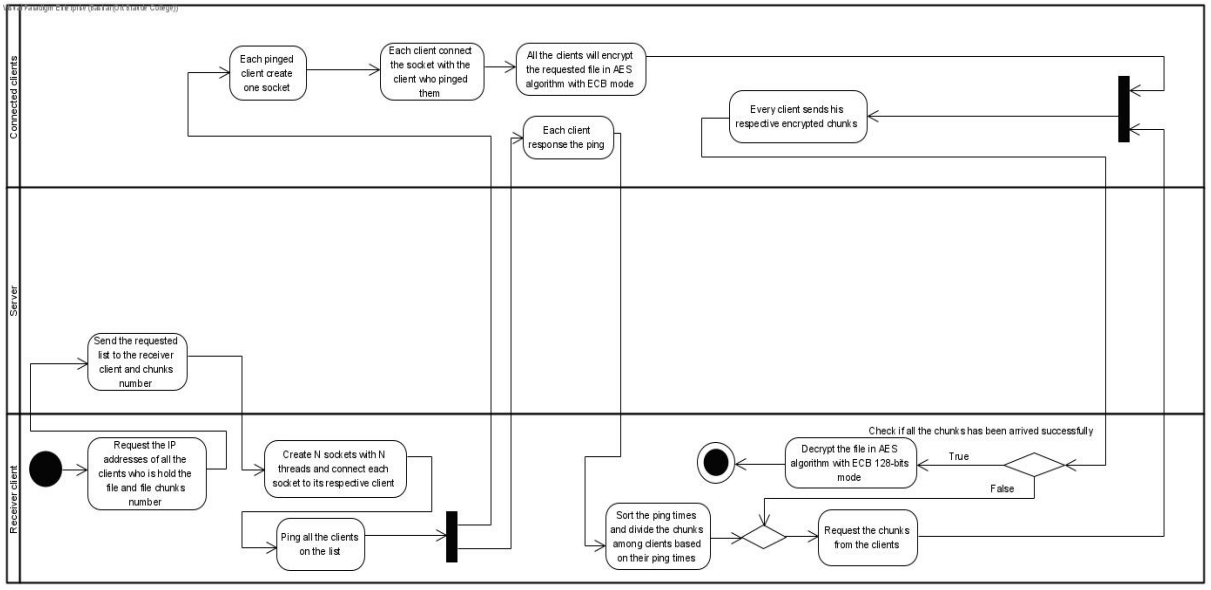
Overall, this project aims to provide a secure and efficient file transfer mechanism by utilizing a combination of encryption, compression, and peer-to-peer communication. The server management data structure contains IP addresses of all the clients, files each client owns and number of chunks each file is divided into. The requesting client requests the number of chunks of the file from the server that he intends to request and the sharing client updates the server and the clients with the number of chunks the file has been divided into.

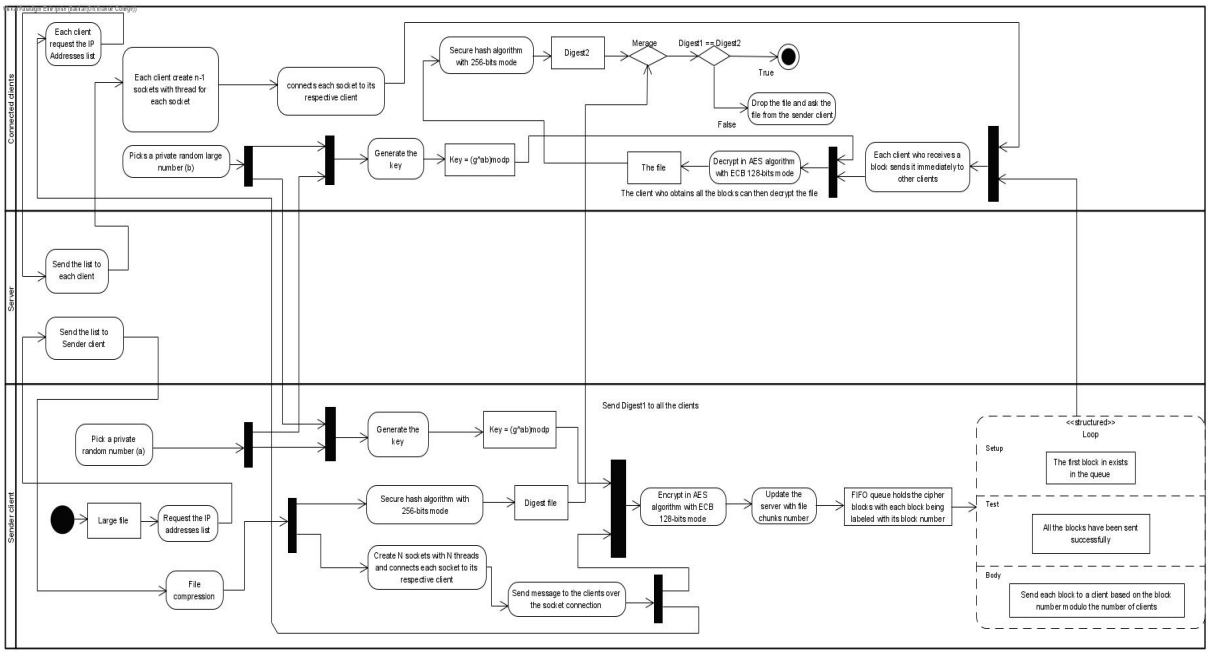
1.4.2 UML Diagrams

For every activity, sequence, and use case model, there are two diagrams included. The first diagram represents the process of a client requesting a file from the server

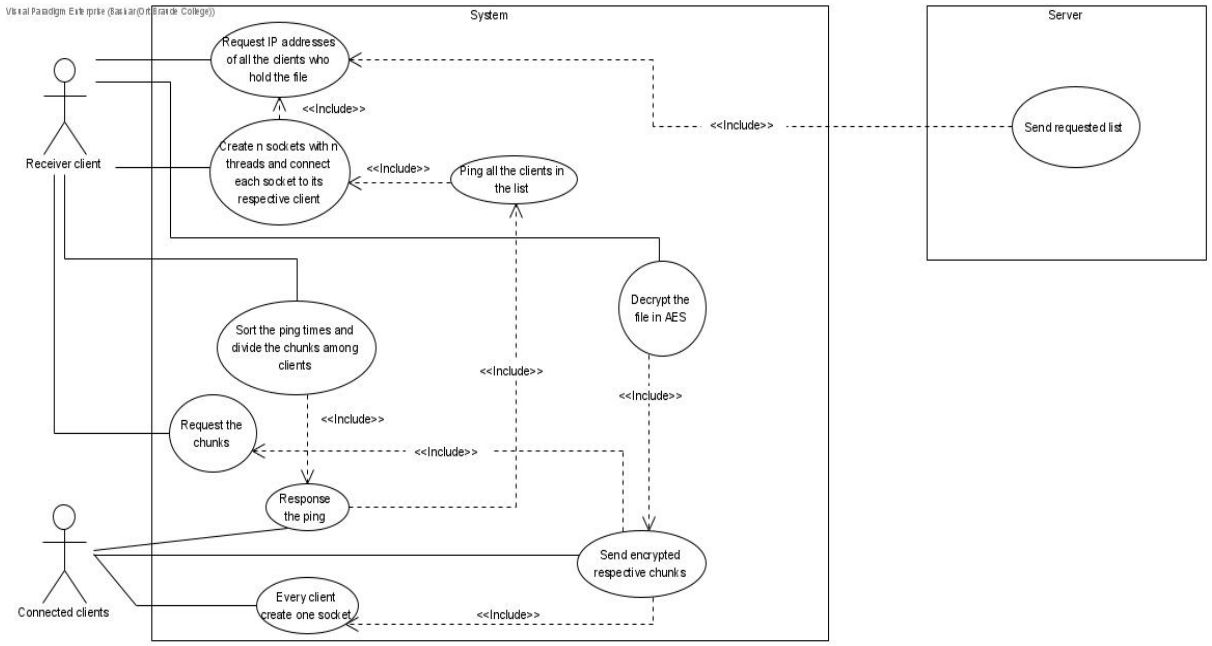
and other connected clients, while the second diagram represents the process of a client sending a file to all the connected clients.

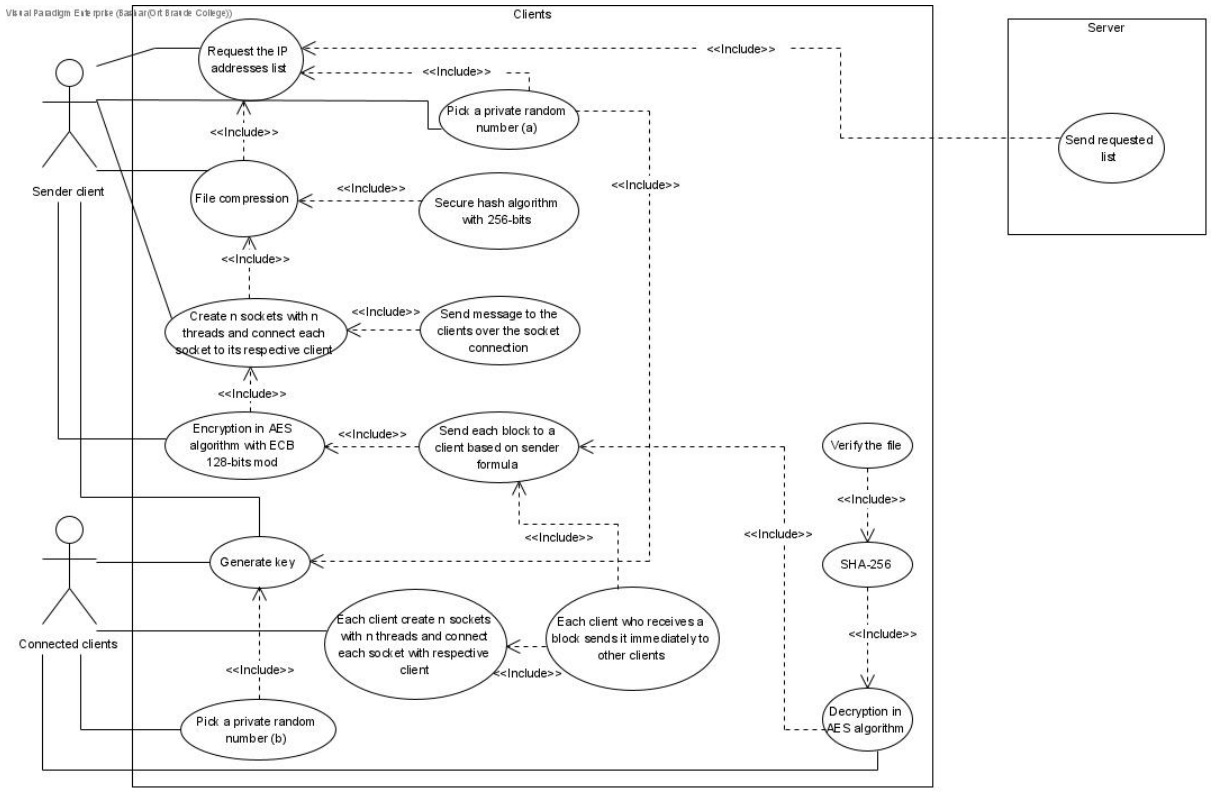
Activity Diagram



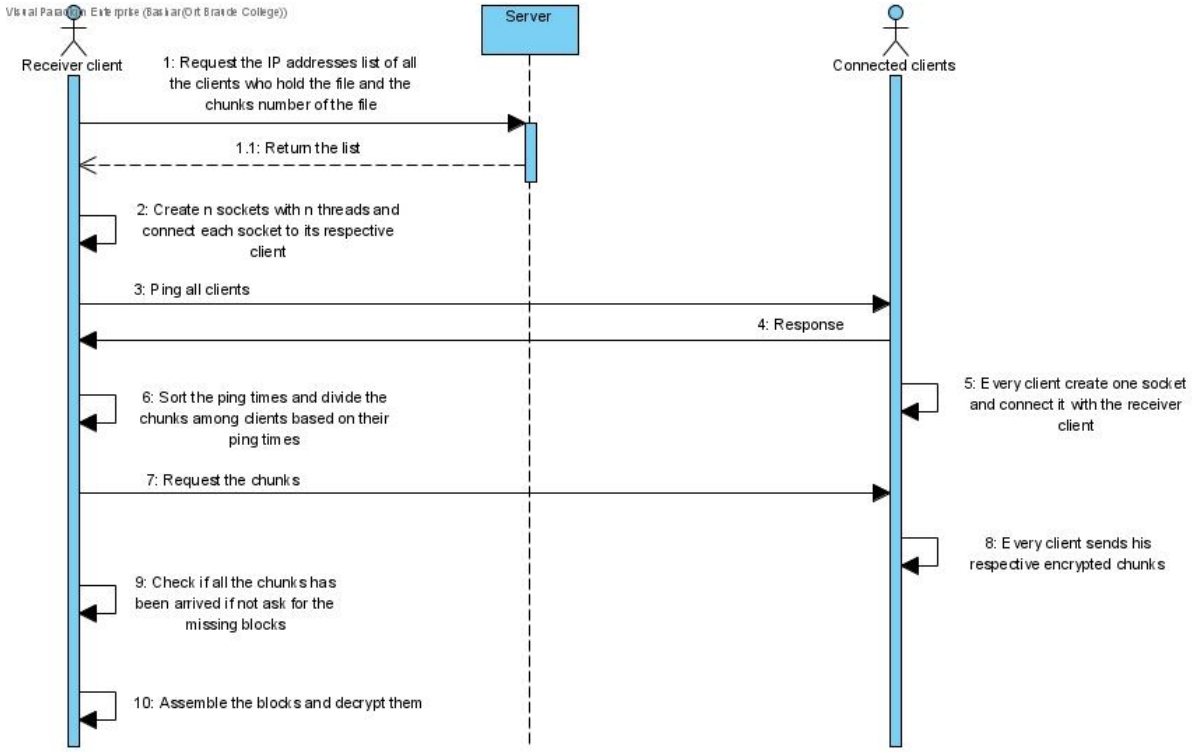


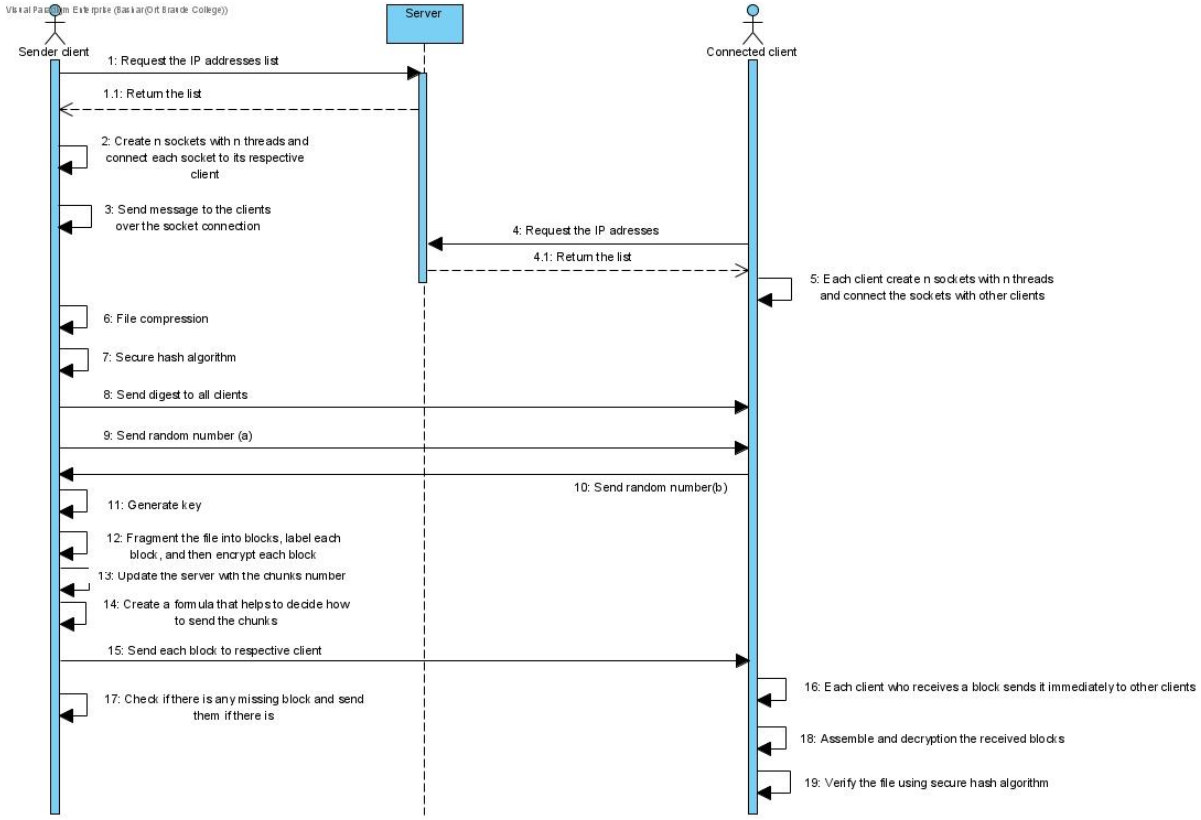
Use Case Diagram


Visual Paradigm Enterprise (Bashar(Ort Brande College))
System
Server
Request IP addresses of all the clients who hold the file
Receiver client
Create n sockets with n threads and connect each socket to its respective client
Ping all the clients in the list
Send requested list
Decrypt the file in AES
Sort the ping times and divide the chunks among clients
Request the chunks
Response the ping
Connected clients
Send encrypted respective chunks
Every client create one socket
<<Include>>


Visual Paradigm Enterprise (Bashar(Ort Brande College))
Clients
Server
Request the IP addresses list
Sender client
Pick a private random number (a)
Send requested list
File compression
Secure hash algorithm with 256-bits
Create n sockets with n threads and connect each socket to its respective client
Send message to the clients over the socket connection
Encryption in AES algorithm with ECB 128-bits mod
Send each block to a client based on sender formula
Verify the file
SHA-256
Generate key
Connected clients
Each client create n sockets with n threads and connect each socket with respective client
Each client who receives a block sends it immediately to other clients
Pick a private random number (b)
Decryption in AES algorithm
<<Include>>

Sequence Diagram





1.5. Verification and Evaluation

1.5.1 Verification

Tests were conducted to verify the effectiveness and performance of our developed file transfer application. We utilized a comprehensive test case matrix to ensure thorough coverage of all key functions within the app. The matrix consisted of test numbers, modules, functions, and expected outcomes, allowing us to systematically evaluate the application's performance. The results were meticulously analyzed to assess the application's adherence to the file transfer standards and to confirm its successful implementation.

Test Number	Module	Tested Function	Expected Result
1	Client	File Request - IP requests	Request the IP addresses of all clients who possess a specific file.
2	Client	File Transfer - IP requests	Request the IP addresses of all connected clients from the server.
3	Client	IP Connections	Connect with the IP addresses that got from the server.
4	Client	File Decryption	Decrypt the received chunks.
5	Client	File Encryption	Encrypt the chunks of a file before sending them to other clients.
6	Client	Compression	Compress the file using the LZW algorithm before sending it.
7	Client	Verification	Verify the integrity of the file using SHA-256.
8	Client	File Assembly	Assemble the received chunks of a file.
9	Client	Chunk Validation	Validate that all clients received all chunks if not send missed chunks
10	Server	File Request	Receive file requests from clients

		Management	and provide them with a list of IP addresses of clients who possess the file.
11	Server	File Share Management	Receive file sharing requests from clients and provide them with the IP addresses of all currently connected clients.

1.5.2 Evaluation

In the second phase of our project, we embarked on a thorough evaluation of the file transfer application we developed. This evaluation aimed to assess the application's performance, reliability, and security by simulating it across various scenarios. Our evaluation had two primary objectives.

The first objective involved testing the application's functionalities to ensure they aligned with the requirements defined during the design phase. Through rigorous testing, we aimed to verify that the application met the specified criteria and operated as intended.

The second objective focused on analyzing performance metrics, including the speed of file transfer, resource utilization, and the level of security provided by the application. This comprehensive assessment allowed us to gain valuable insights into the application's overall performance and identify any potential issues or areas for improvement.

By conducting this evaluation, we aimed to ensure that our file transfer application not only met the initial design requirements but also delivered optimal performance, reliability, and security to our users.

2.1 User Guide

The "Fast Secure File Transfer (FSFT)" software serves the purpose of reducing file transfer time by decreasing the overall transfer time and dividing the file into smaller, manageable chunks. Additionally, it provides user assistance in securely transferring files from one location to another, ensuring strict security measures and file integrity guarantees.

The software is implemented using the Java programming language and developed within the Eclipse IDE version 2020. For network visualization, the software utilizes the "JavaFX" library, while the "Scene Builder" tool is employed for designing and building the graphical user interface (GUI).
Additionally, the software utilizes the OCSF (Object Client-Server Framework), which is a Java-based framework that provides a foundation for building client-server applications. The framework supports multithreading, allowing for concurrent processing of client requests and server responses. Furthermore, the software incorporates peer-to-peer functionality, enabling direct communication between clients without the need for a central server.

2.2 Operating Instructions

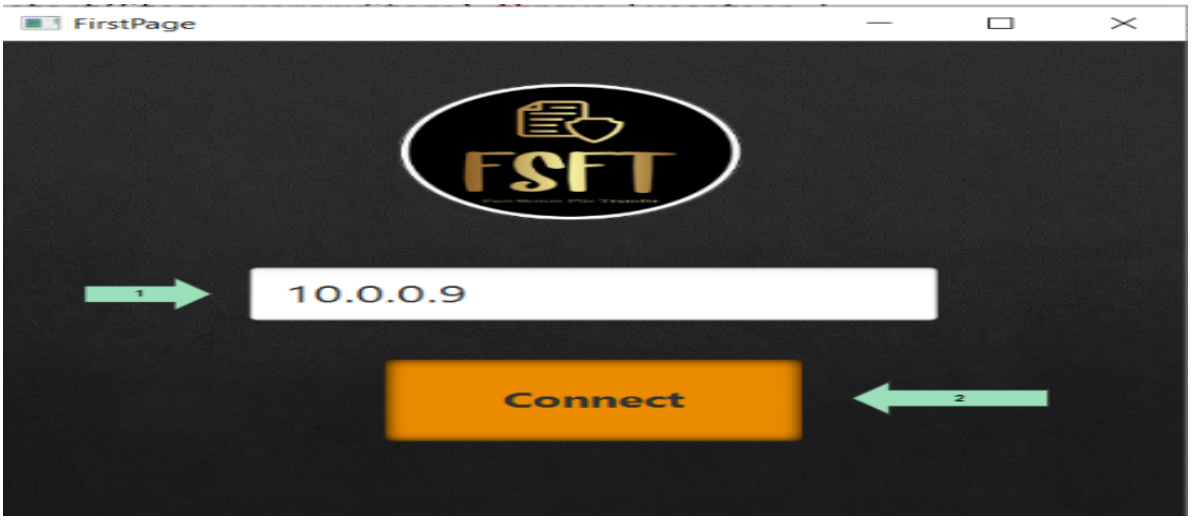
Server

To enable the server to listen for incoming client connections, you need to run the server application. By running the server

1)The IP address of the server .
2)An information table that presents a consolidated view of all connected clients.
3)After double-clicking an IP address in the table, a list of all files existing on that IP address will be displayed.
4)The text displays the port number of the server's socket.

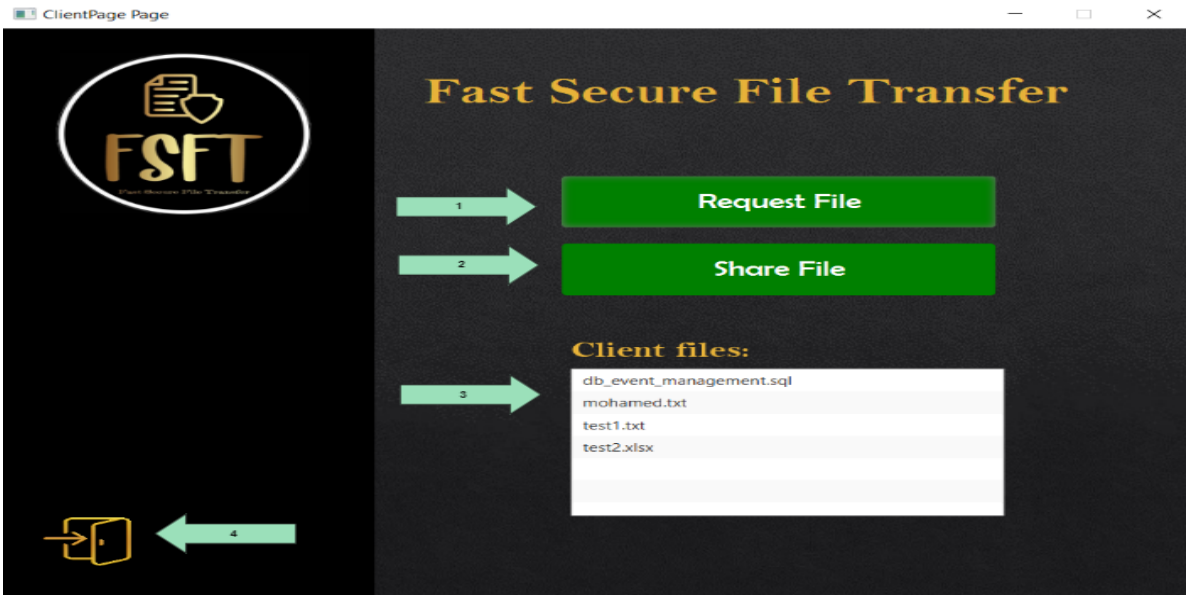
Clients

To establish a connection with the server, run the client application in a manner similar to the server and connect to the server's designated port. Furthermore, when connecting with other clients, it create a new port for establishing the connections. To connect to the server, you need the server's IP address.



Client Page:

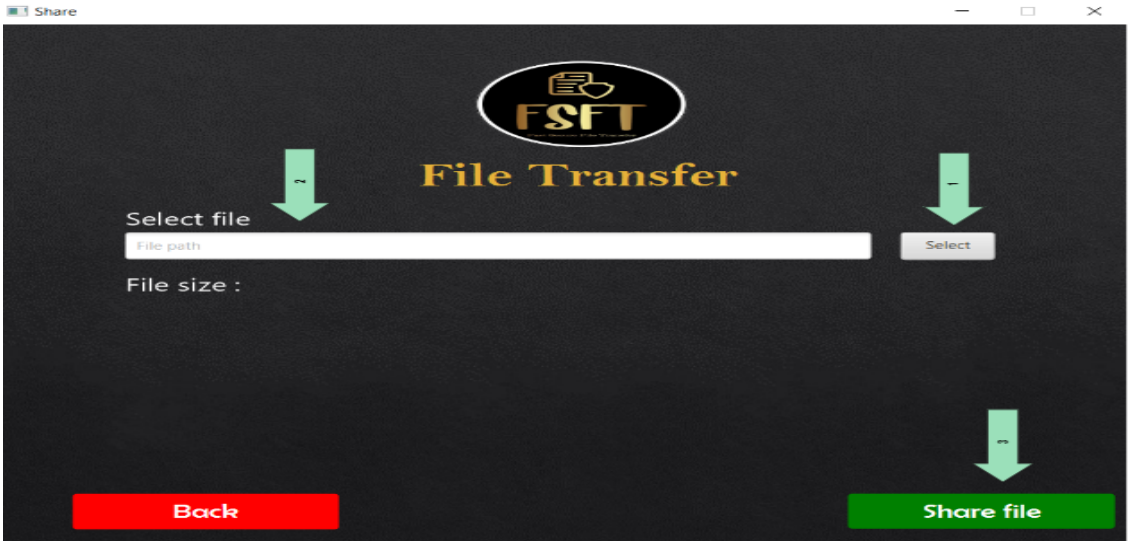
This is the main page of the client application, where you will find two options: "Share" and "Request File." These options allow you to perform specific actions related to file sharing.



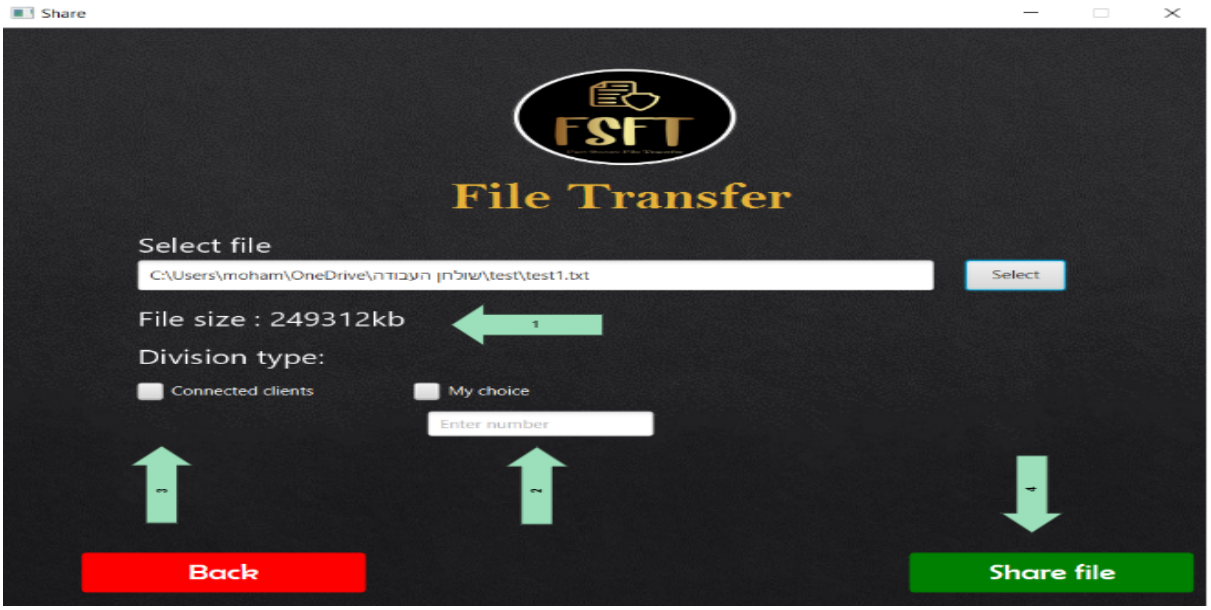
1)Selecting the "Share" option, you can share files with other clients .
2)Choosing the "Request File" option enables you to request files from other clients .
3)The table displays all the files that exist in the client's storage.
4)Exit button.

Share page:

On the share page, you can select the file that you want to share [1] by clicking in select. and it displays the path of file [2] .

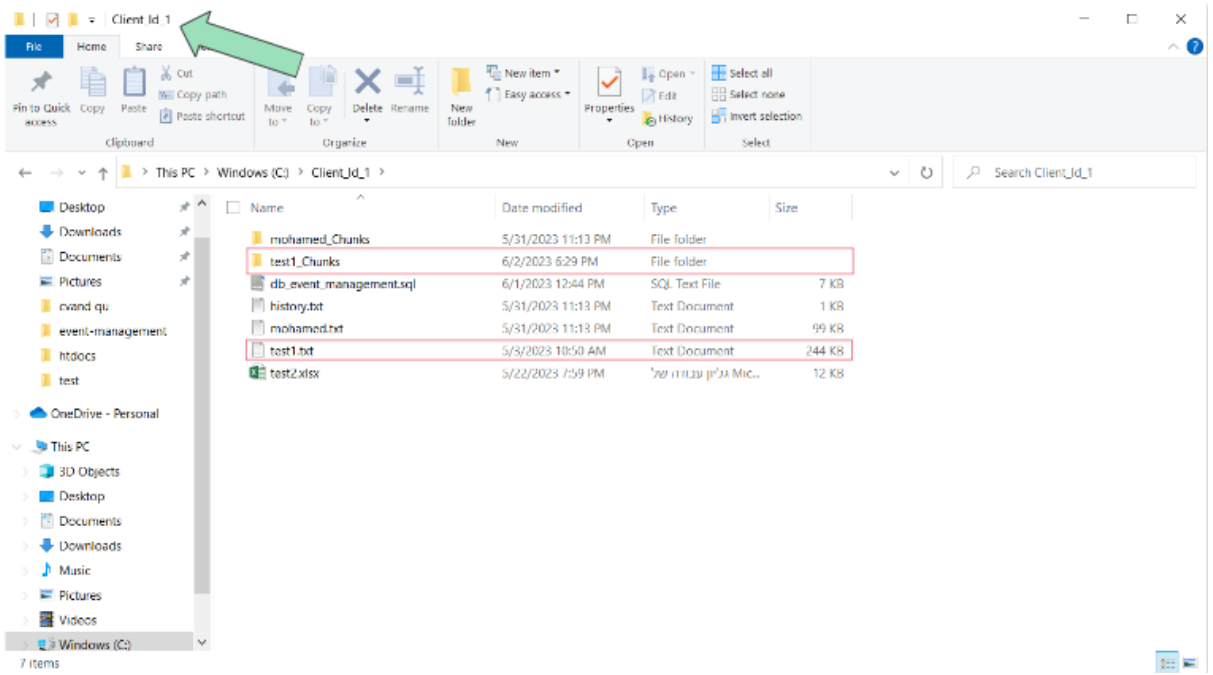


Based on the file's size[1], you have the option to choose how it will be divided. You can either divide the file into chunks based on the number of connected clients in the system[3], ensuring that each client receives a portion of the file, or you can divide it based on a specific number that you choose[2]. This allows you to customize the sharing process according to your preferences and the available resources,
and finally share button share the chunks to all connected clients in system .



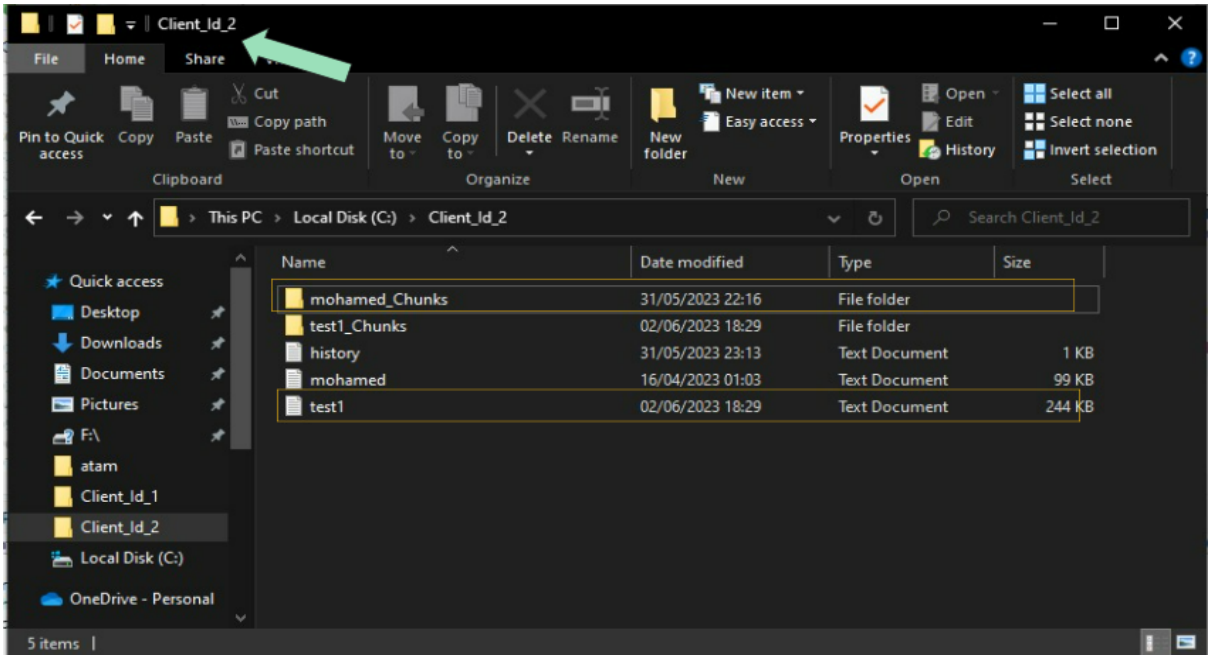
After sharing the file, the client receives a message indicating that the file has been encoded using LZW add ,encrypted using AES-256 (Advanced Encryption Standard) for enhanced security. Additionally, the encryption is further ensured by employing the Secure Hash Algorithm 256-Bit (SHA-256) for data integrity verification.

Client1 saves the completed file and the folder containing all the file chunks on C, By Client1 ensures that they have a local copy of the shared file and all its associated chunks readily available. This allows them to quickly share the file if any other clients request it in the future. Storing the file and its chunks in a designated location on the desktop provides easy access and facilitates the sharing process

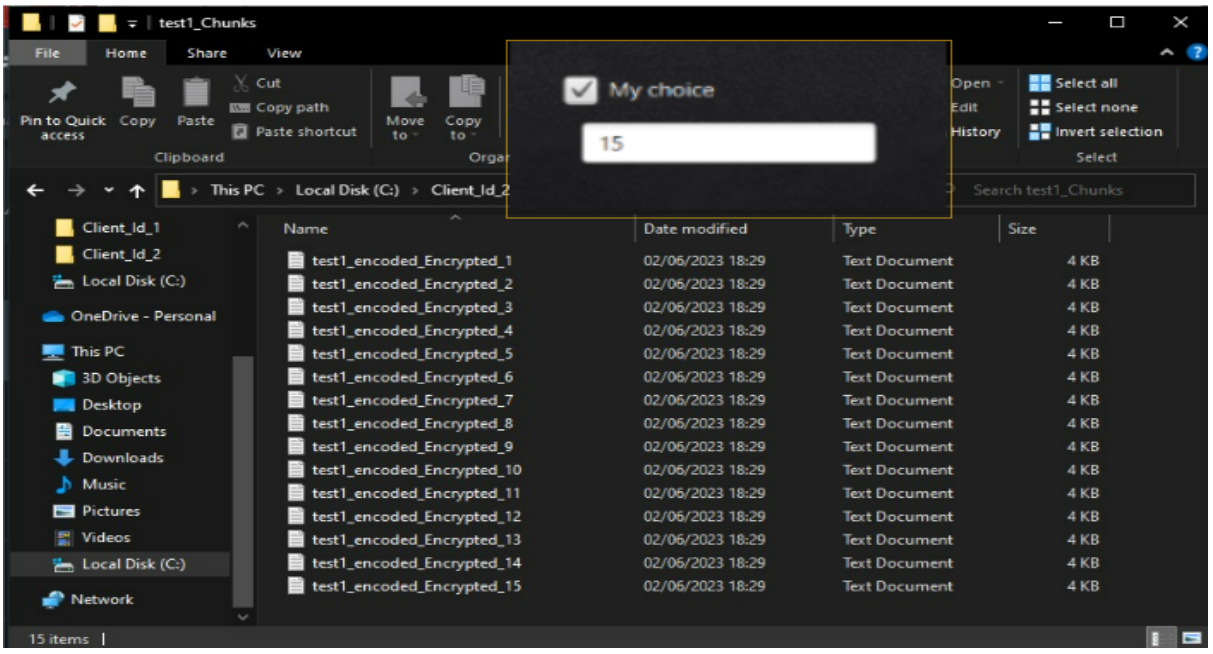


On the other hand, Client2 receives the file chunks from Client1. Client2 performs the necessary decryption process to decrypt the received chunks. After decryption, Client2 decodes the file using LZW and merges the chunks together to reconstruct

the original complete file. To ensure the integrity of the reconstructed file, Client2 applies the Secure Hash Algorithm 256-Bit (SHA-256) to generate a hash value.



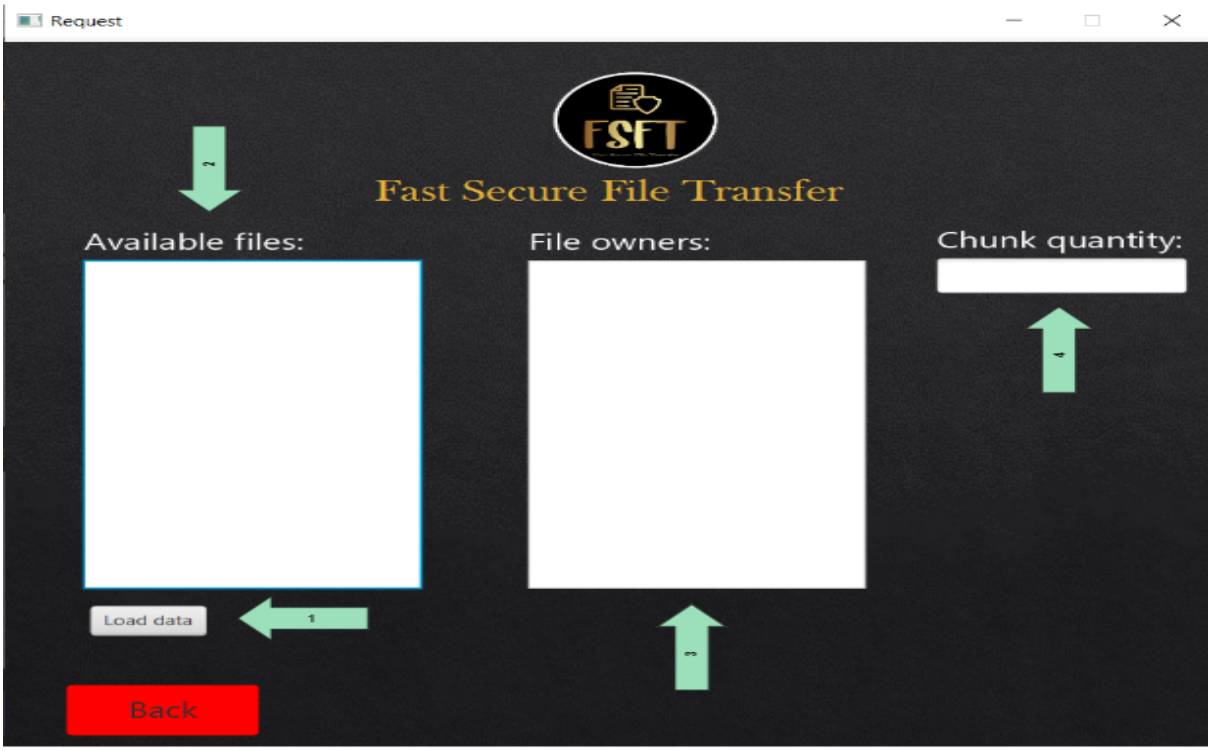
After selecting the number of chunks as 15, the file will be divided into 15 smaller parts. Each chunk represents a portion of the original file

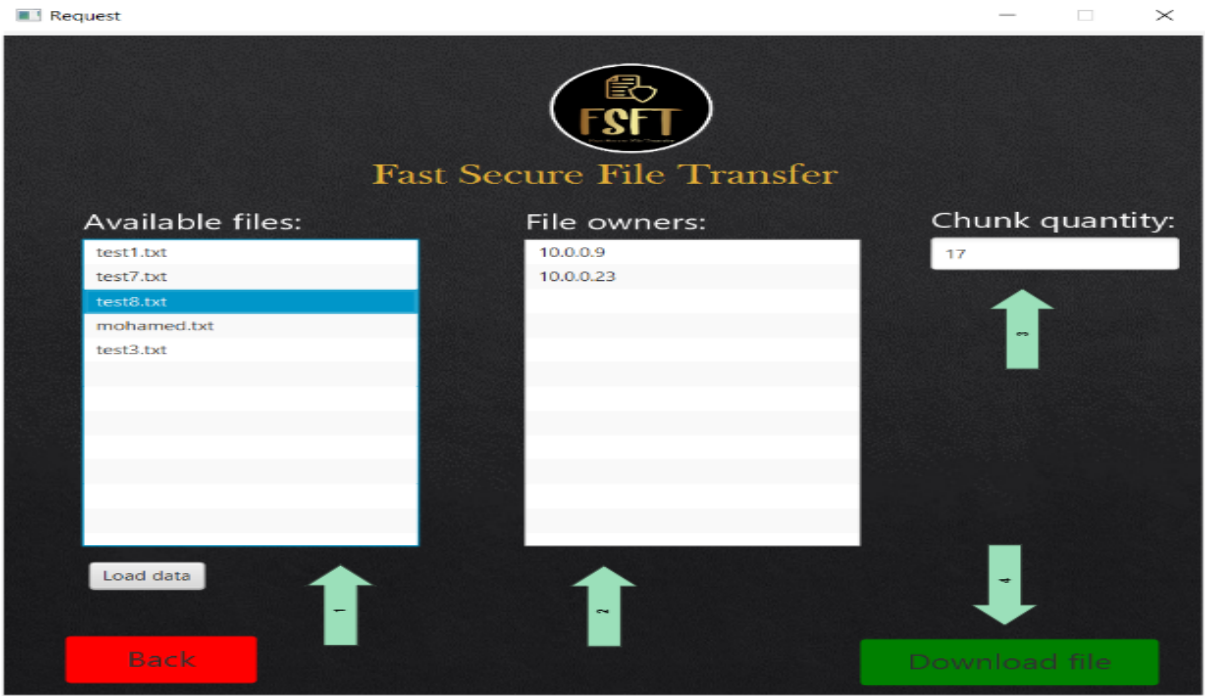


Request page:

On the request page, you will find two lists: "Available Files" and "File Owners." When you click on the "Load Data" button, all the available files in the system that can be requested will be displayed. By clicking on a file from the "Available Files" list, the user will be shown the IP address of the client that holds the file, along with the

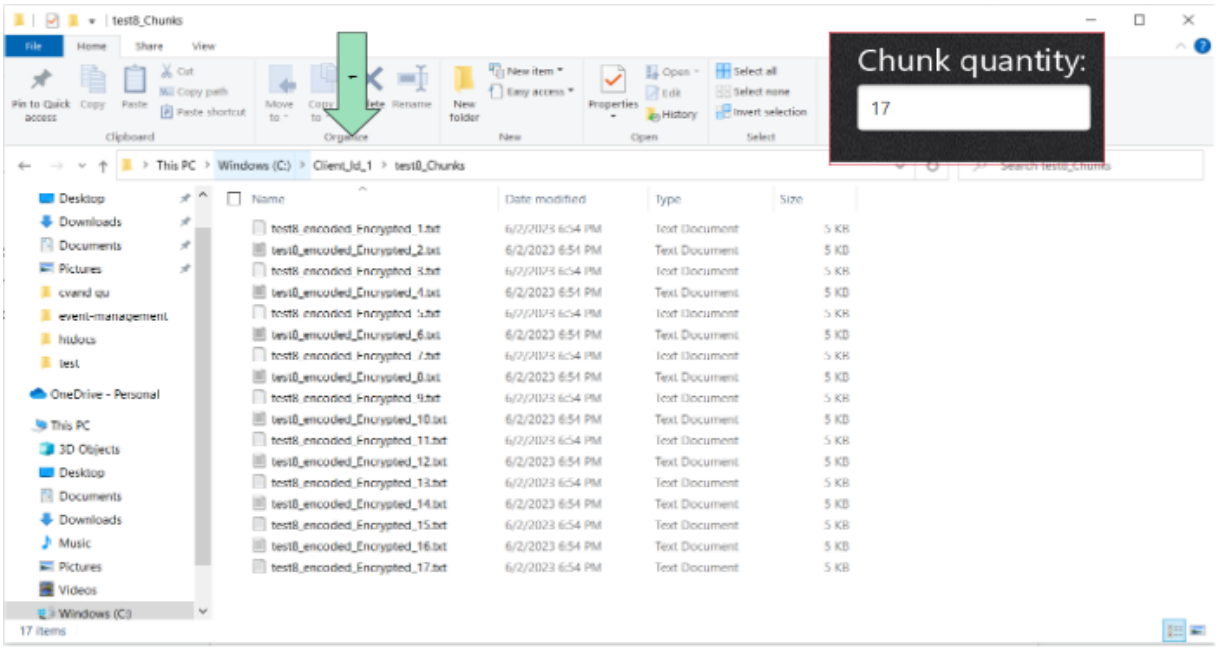
number of chunks that comprise the file. This information helps the user identify the source of the file and determine the file's integrity based on the number of chunks available.





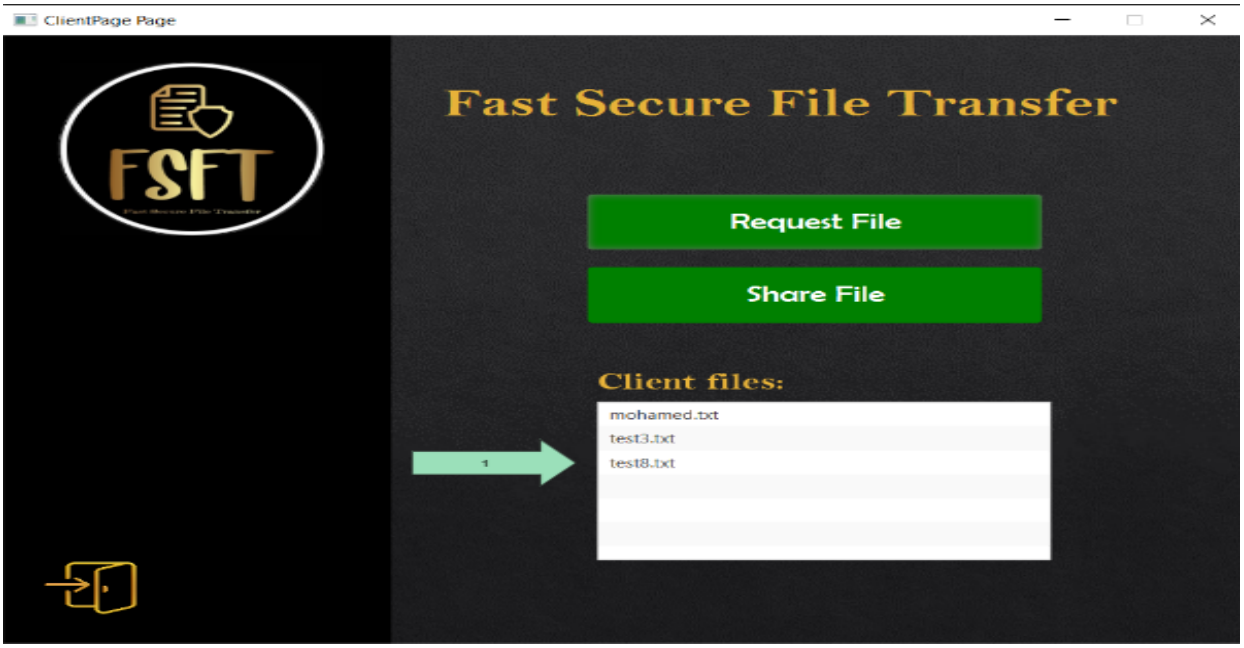
After clicking the "Download" button, the application will initiate a process to calculate the ping time of all connected clients in the system. This ping time

measurement helps assess the network latency and responsiveness of each client. Based on this information, the application divides the mission of downloading the file



The completed file and its chunks will be downloaded and stored in the client's local space. This ensures that the file is readily available if any other client needs it in the future.

Finally, after the download process, you can confirm that the file exists in the client's local space.



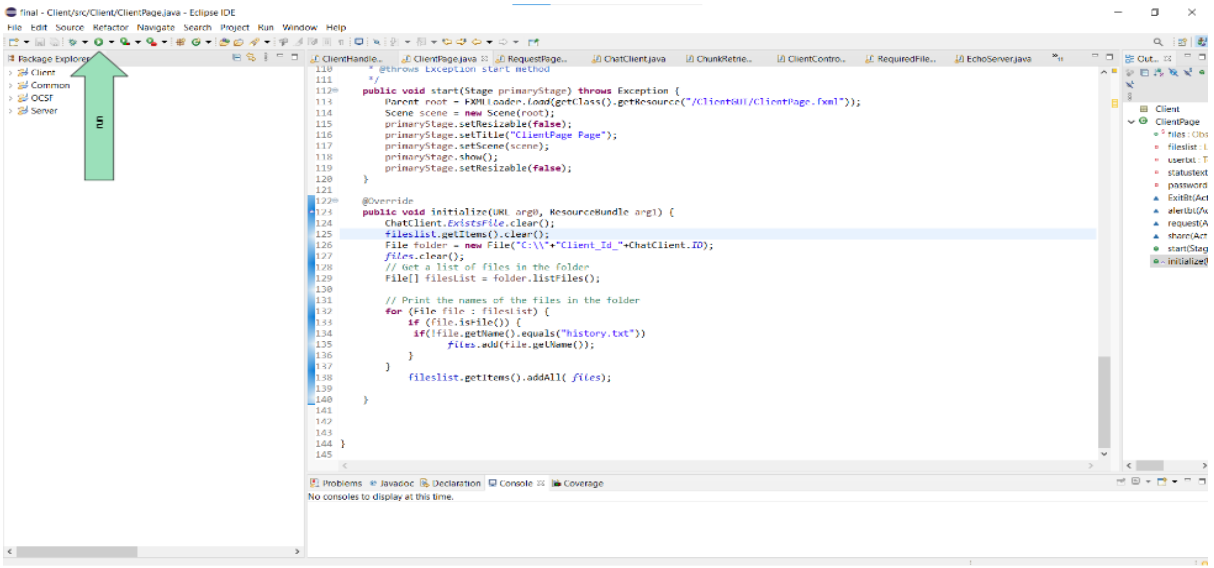
2.3 Maintenance Guide

To run the application, you will need to download the JavaFX library and add it to the build path of your clients and server. Start by obtaining the JavaFX library from the official Oracle website or a suitable repository.

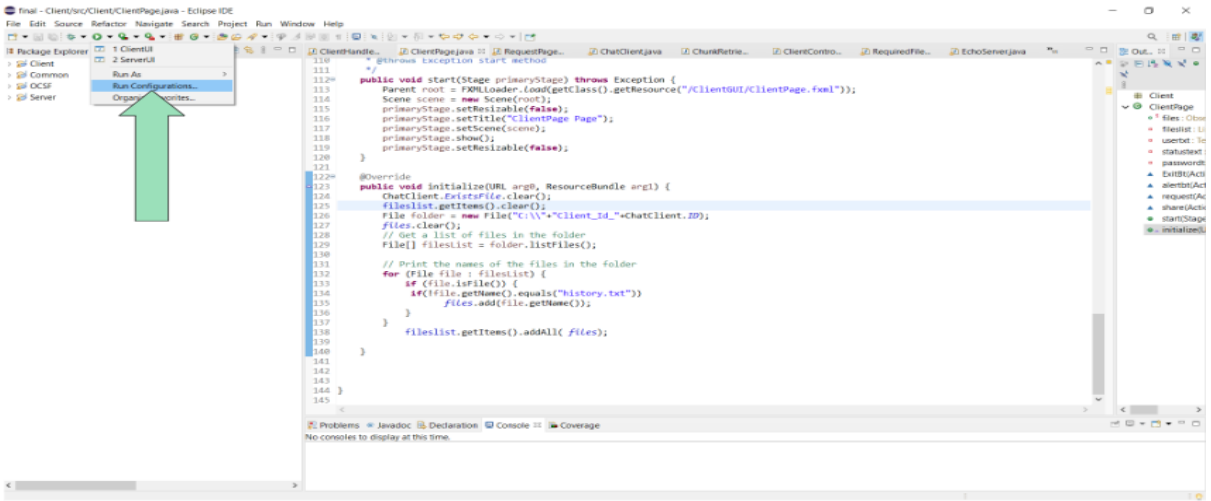
After downloading JavaFX, extract the contents of the library to a preferred location on your computer. Next, open the Eclipse IDE 2020 and create a new Java project for your application.

Configure the build path by right-clicking on the project in the Project Explorer, selecting "Build Path," and then "Configure Build Path." In the Libraries tab of the Configure Build Path window, click on "Add External JARs" or "Add Library" (depending on your Eclipse version).

Navigate to the location where you extracted the JavaFX library, select the appropriate JAR files, and add them to the build path of your project. Additionally, you will need to set up the JavaFX runtime in Eclipse.

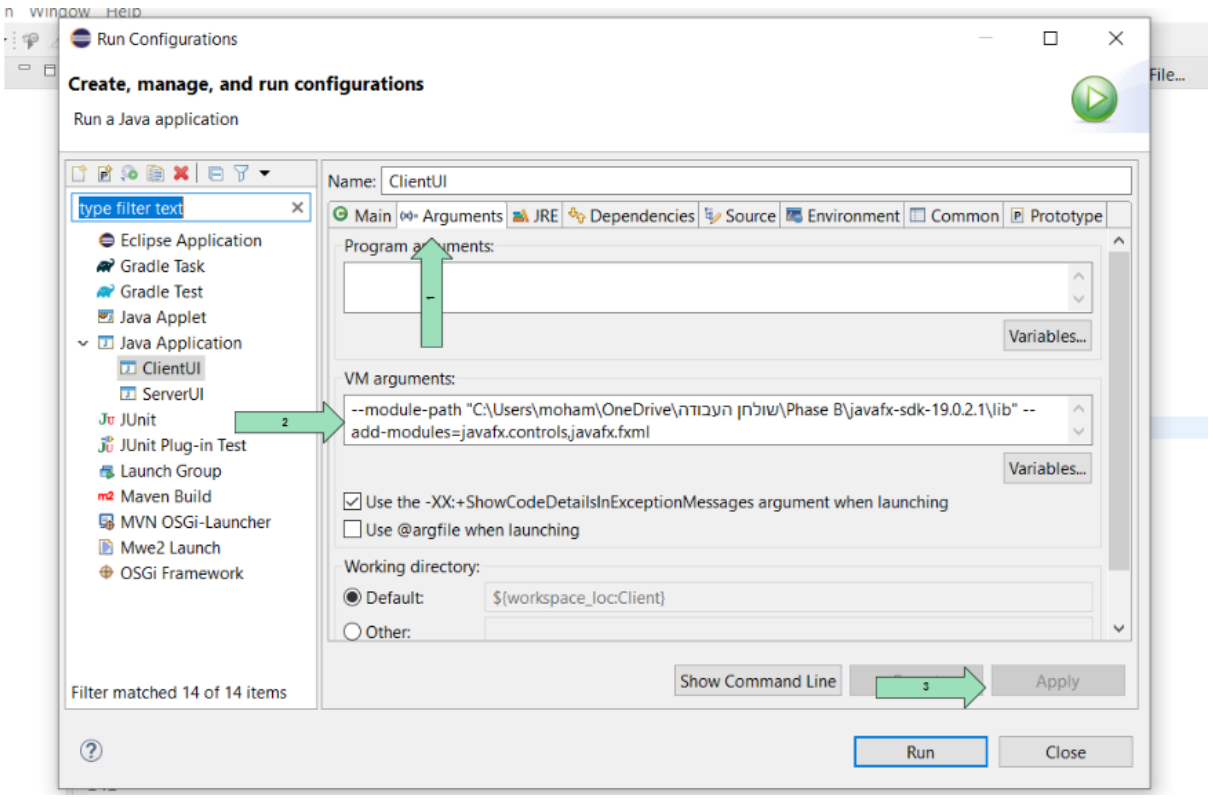


Go to the Run Configurations of your Eclipse project



Access the Arguments tab. In the VM arguments section ,add the following line, replacing the path with the actual location of the JavaFX runtime on your system:

`--module-path /path/to/javafx-sdk-VERSION/lib --add-modules javafx.controls,javafx.fxml`



3.Result and conclusion

The FSFT (Fast Secure File Transfer) app, as its name suggests, is a highly efficient and secure file transfer solution designed to operate in both client-server and peer-to-peer environments. Through meticulous development and testing, FSFT has demonstrated its ability to ensure fast, reliable, and protected file transfers, catering to the needs of diverse users.

FSFT employs an innovative approach to file transfer, where files are divided into smaller, manageable chunks upon transmission. This segmentation technique not only optimizes transfer speeds but also enhances overall efficiency. Moreover, each chunk is individually encrypted using robust encryption algorithms, guaranteeing the confidentiality and security of the transferred data.

To add an extra layer of protection, FSFT utilizes encoding techniques on the encrypted chunks. Encoding transforms the data into a different representation while preserving its integrity, thereby reducing the risk of corruption during transit and bolstering the app's resistance to unauthorized access.

Upon reaching the receiving side, FSFT intelligently merges and decrypts the encoded chunks, reconstructing the original file. By employing the same encryption algorithms and keys used during the initial encryption process, the app ensures secure and accurate file reconstruction.

conclusion :

The FSFT (Fast Secure File Transfer) app is a reliable, efficient, and secure solution for file transfers in clients server and peer-to-peer environments. Its unique features, such as file segmentation, encryption, encoding, and robust authentication, make it an excellent choice for individuals and organizations seeking fast and secure file transfers. Whether in a client-server or peer-to-peer setup, FSFT offers a user-friendly interface, optimized performance, and stringent security measures, making it a valuable asset for users looking to exchange files quickly and securely.

4. References

1. Singh, G. (2013). A study of encryption algorithms (RSA, DES, 3DES and AES) for information security. International Journal of Computer Applications, 67(19).

2. Padate, R., & Patel, A. (2014). Encryption and decryption of text using AES algorithm. International Journal of Emerging Technology and Advanced Engineering, 4(5),54-9.

3. Mohamed, A. A., & Madian, A. H. (2010, December). A Modified Rijndael Algorithm and it's Implementation using FPGA. In Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on (pp. 335-338).

4. Pramstaller, N., Gurkaynak, F. K., Haene, S., Kaeslin, H., Felber, N., & Fichtner, W. (2004, September). Towards an AES crypto-chip resistant to differential power analysis. In Solid-State Circuits Conference, 2004. ESSCIRC 2004.Proceeding of the 30th European IEEE (pp. 307- 310).

5. Ibrahim F. Elashry, Osama S. Farag Allah, Alaa M. Abbas,S. El-Rabaie and Fathi E. Abd El-Samie ‘’ Homomorphic image encryption’’ Journal of Electronic Imaging 18(3), 1 (Jul–Sep 2009)

6. N. El-Fishawy and O. M. Abu Zaid, “Quality of encryption measurement of bitmap images with RC6, MRC6, and Rijndael block cipher algorithms,” Int. J. Network Security 53, 241–251 2007.

7. RFC 2631 – Diffie–Hellman Key Agreement Method E. Rescorla June 1999.

8. Secure Hash Standard (SHS), N. I. of Standards and Technology (2012)

9. A. Anand, Breaking Down:SHA-256 algorithm (2019). [Online]. Available https://medium.com/bugbountywriteup/breaking-down-sha-256-algorithm-2ce61d86f7a3. Accessed: 29 Jun 2020