Secure File Transfer Application

•••

Asymmetric Encryption with SSL Certificates

Group Details

Group Number: 6

Group Members

1.	Ajay Dayma	17114006
2.	Bhavye Jain	17114020
3.	Kaustubh Trivedi	17114044
4.	Shiva Reddy	17114045
5.	Sai Krishna Abhiram	17114054
6.	Ritik Kumar	17114063
7.	Saurabh Singh	17114068

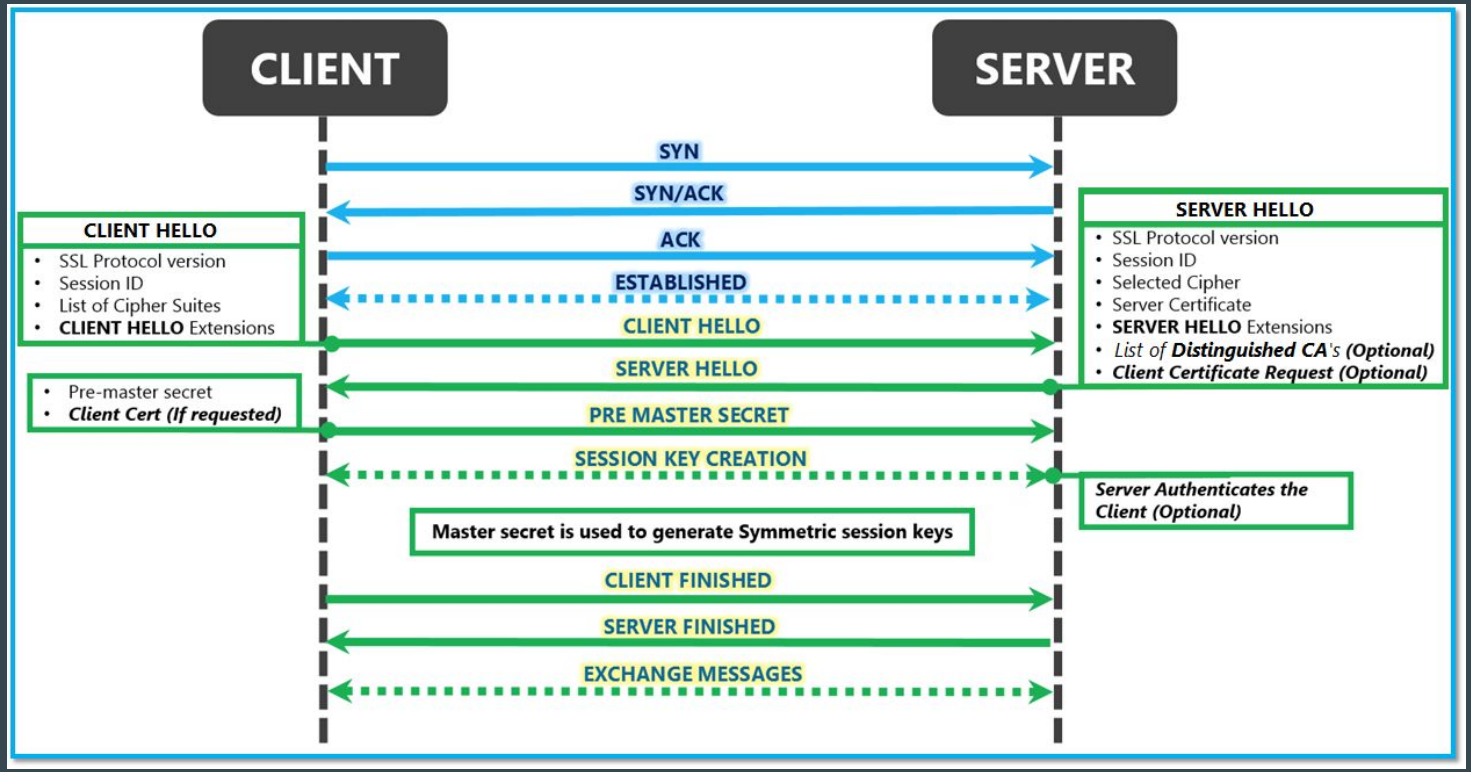
The Problem Statement

An organization needs an application which can help their employees to transfer files between them securely on the same network. Develop an application using socket programming to send files between two machines and secure the data transfer using a strong encryption algorithm. Capture these packets using a sniffing tool like Wireshark and show that data transfer is secure

Assumptions

1. The required system is built to operate inside the network of an organization.
2. Since the clients belong to the same organization, they need not communicate to decide a cypher between them.
3. The certificates for secure communication are digitally generated by a master authentication server inside the organization. This server has the root certificate and key pair.
4. All the clients agree to encrypt data using asymmetric public-private key encryption.
5. The public ports of other nodes are already known within the network.
6. Every device entering our network already has the organization's root certificate pre-installed on their devices, which would be used for key chain validation.
7. Auth server is online and it's accessible to the user in the network.

Security on the internet - TLS/SSL



Security in our application

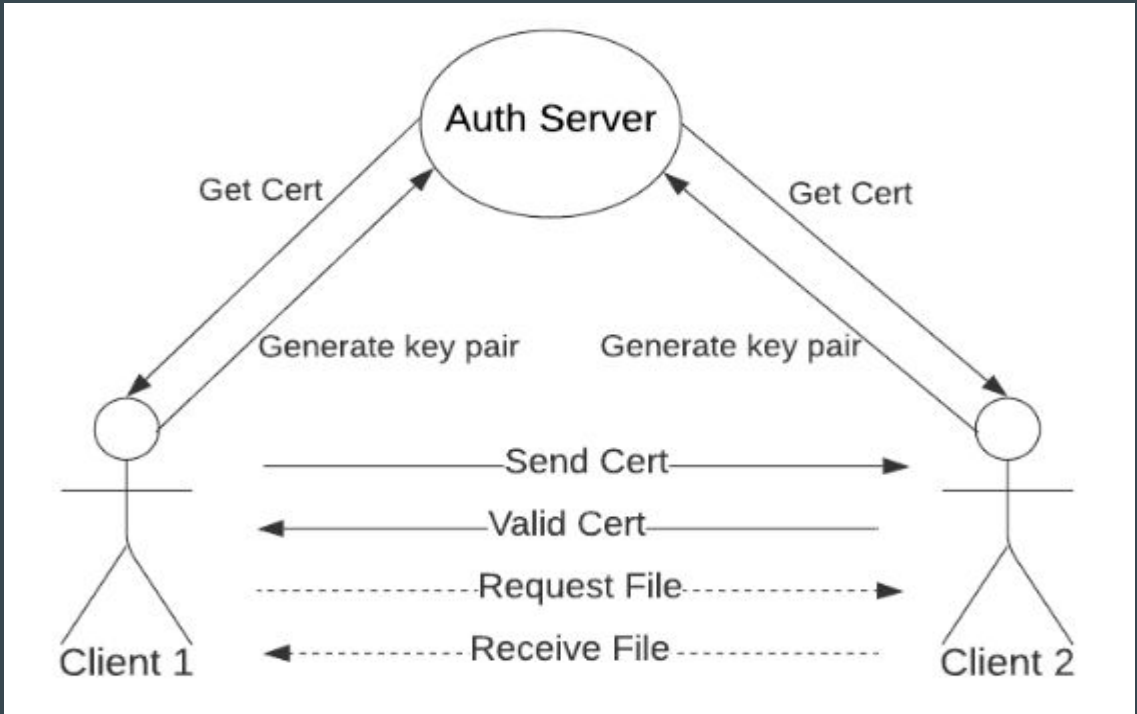
Both the sender and receiver have a certificate issued by the authentication server. First, a TCP handshake is done between the two parties while creating a socket.

Further steps-

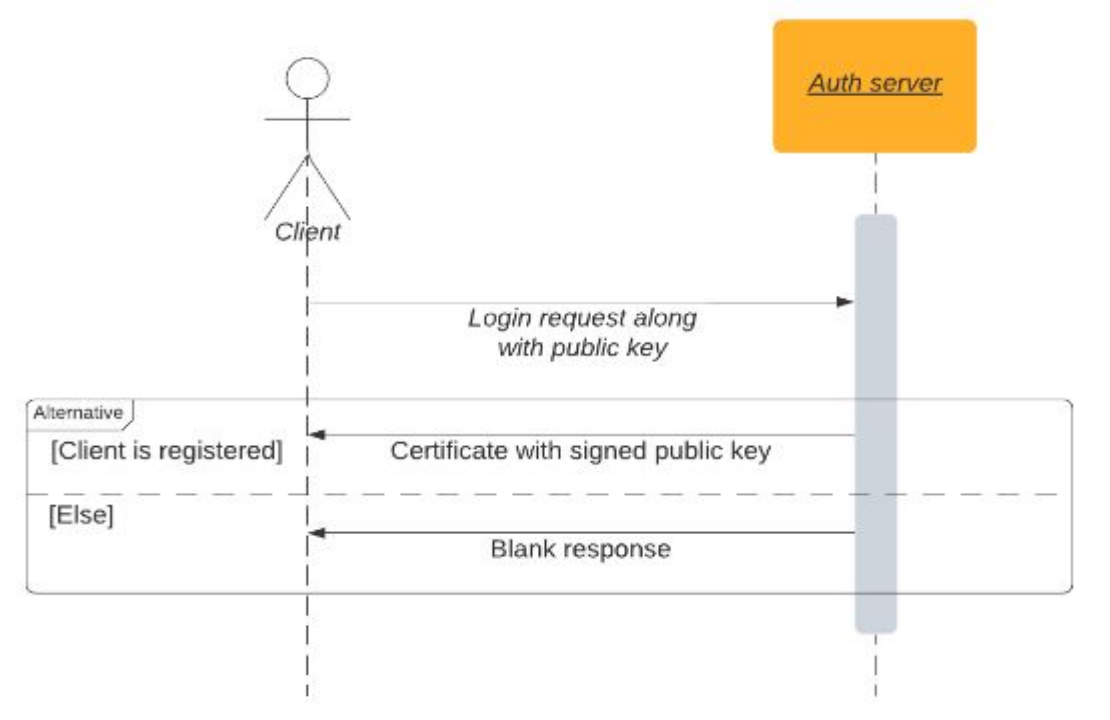
- Client 1 Certificate Send: The client that wants to initiate a file transfer sends a request for a secure connection with the server. It sends it’s signed certificate to the other client.
- Client 2 Certificate Auth and Send: The receiver receives the transfer request, and verifies the authenticity of the certificate sent by client 1 using the auth server’s trust anchor certificate .The secure connection is then established.

Working of the System

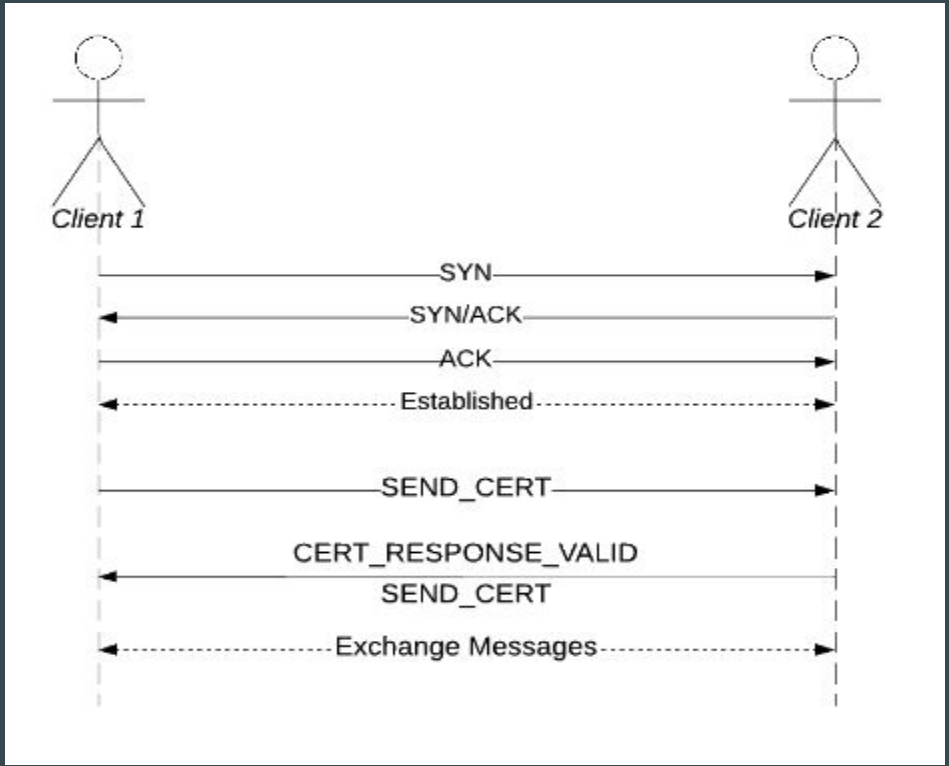
Architecture



Client Auth Server



Client to client communication



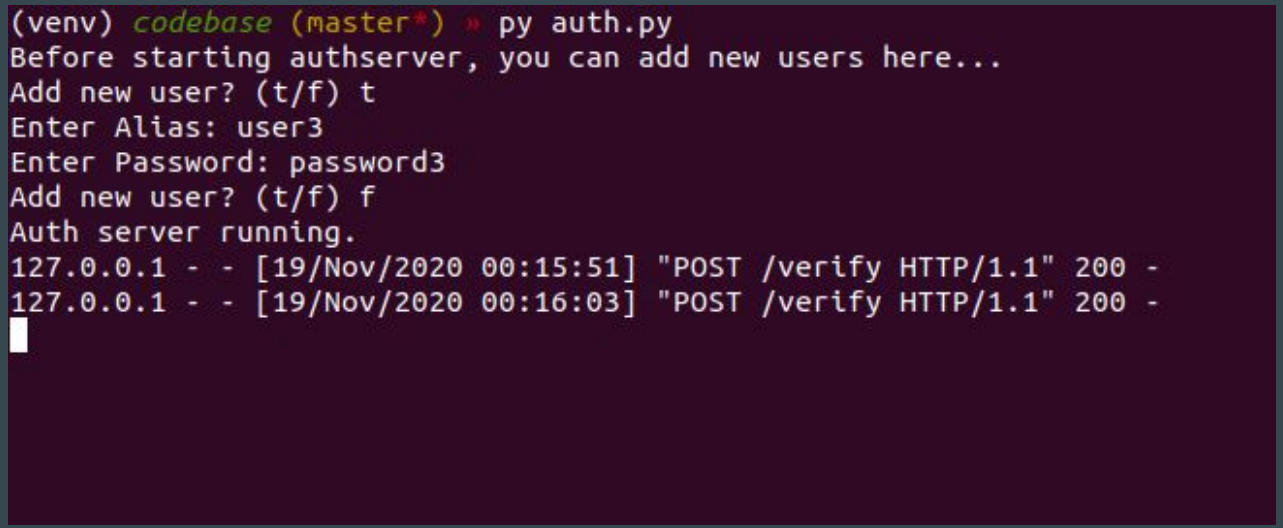
Code Components [Modularization]

- auth.py - The master authentication server for the organization. Functions as the trust anchor for the system and signs certificates of the clients.
  - SHA256 hashing to encrypt the certificate
  - X.509 format for the certificate
- client.py - The front end script for each individual client.
  - Gets the user logged in safely and gets the user's key certified.
  - Initiates a connection to another client for file transfer

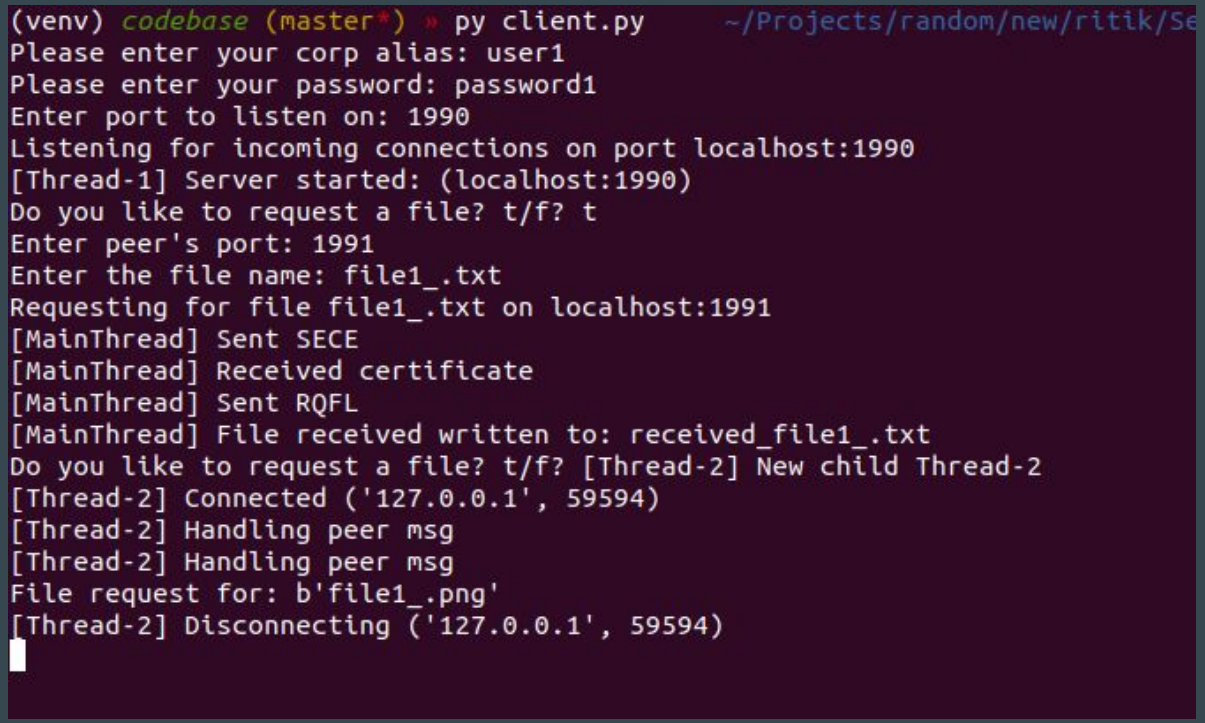
Code Components [Modularization]

- peer.py - File to store the state of a client
  - Stored data like received public key for a particular connection.
  - Also responsible for sending and receiving data from connected peer.
  - Data sent is in the form of a struct: `!4sL<payload_length>s`
- connection.py - Helper script containing procedures for requesting files and to handle incoming connections and messages
- encrypt.py - A helper script to handle and generate keys and certificates

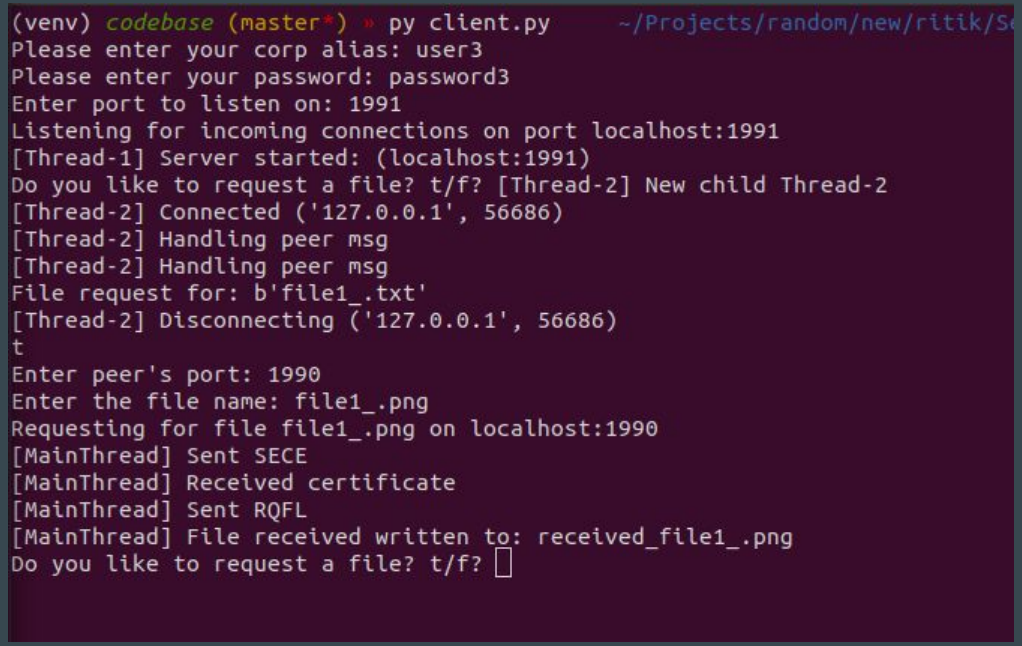
Working Demonstration



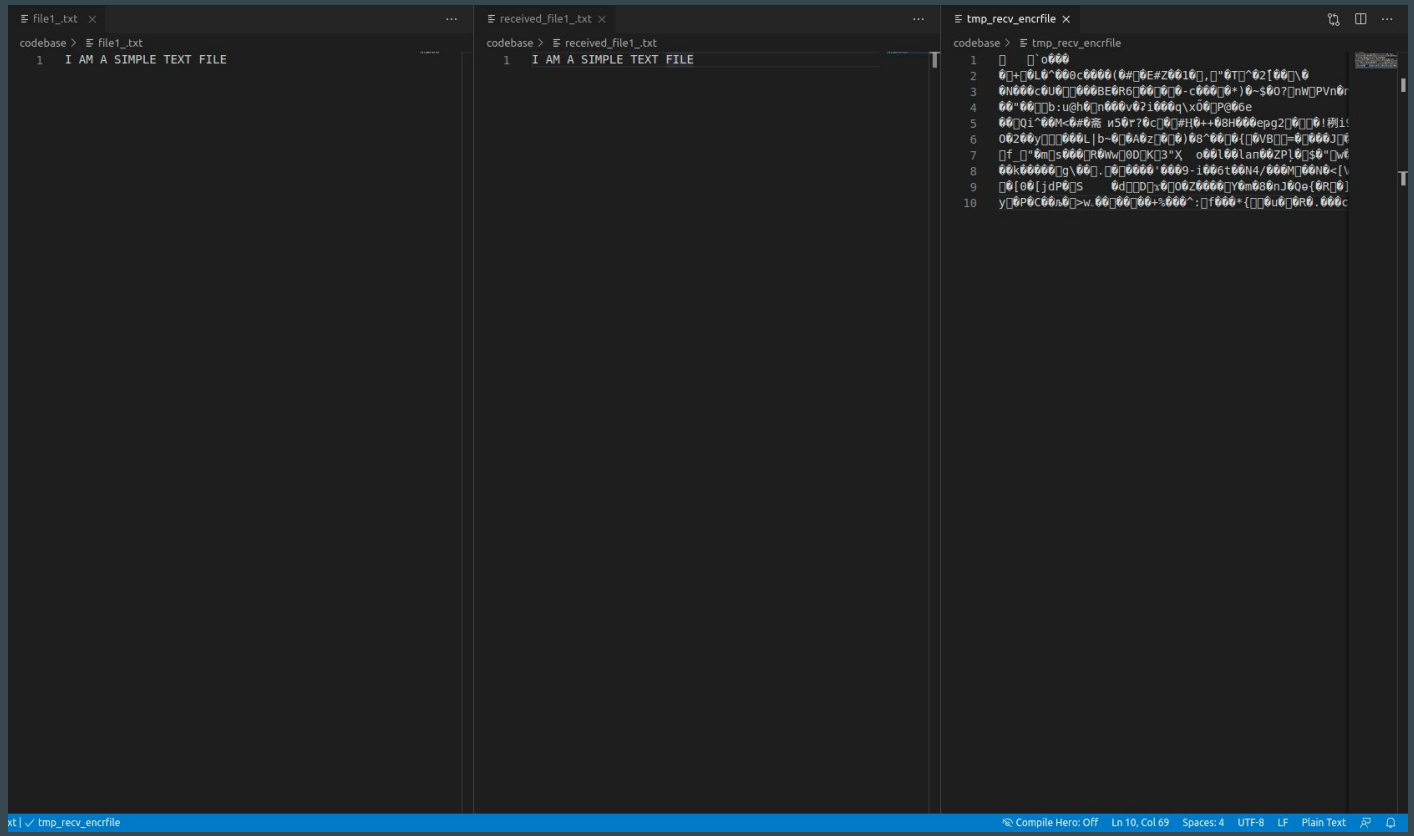
Working Demonstration



Working Demonstration

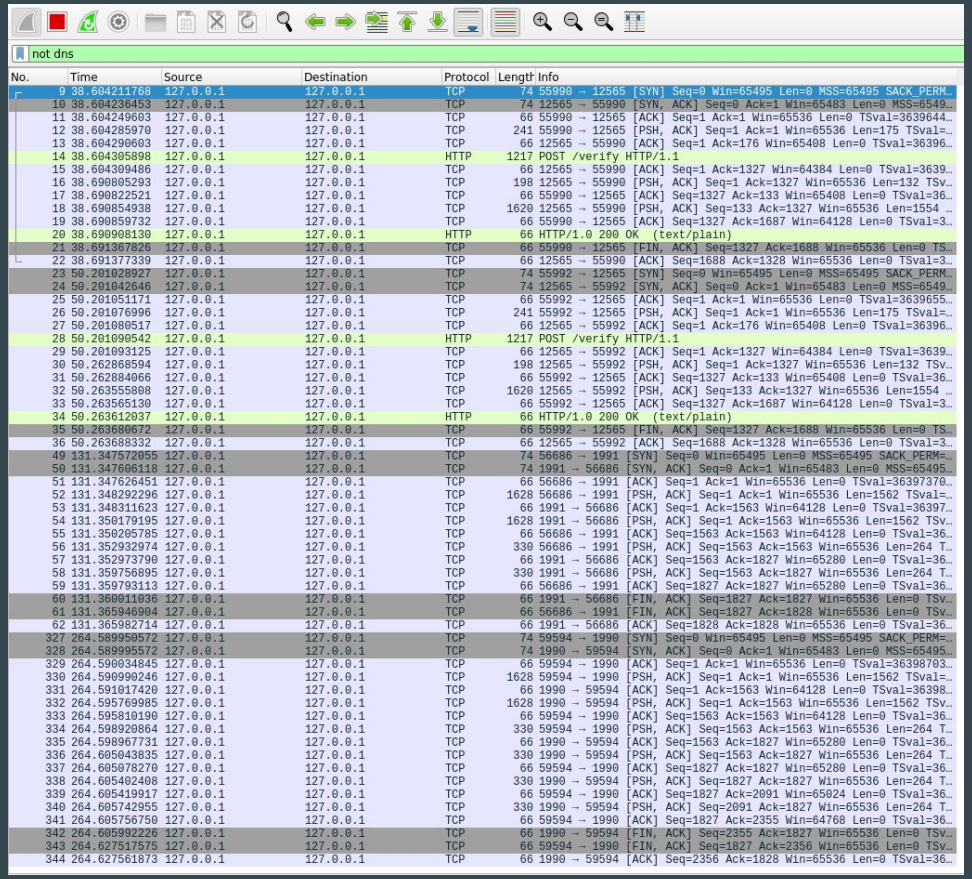


Working Demonstration



Wireshark Captures

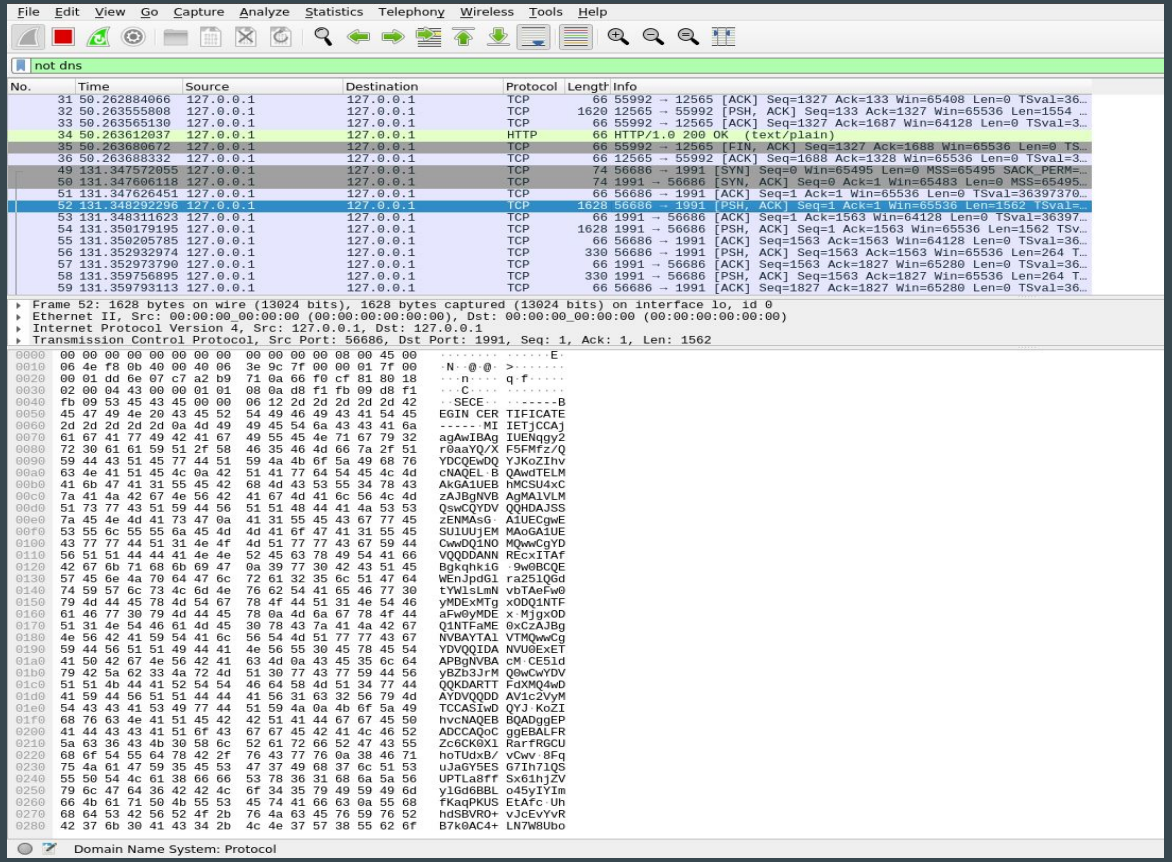
DATA EXCHANGES



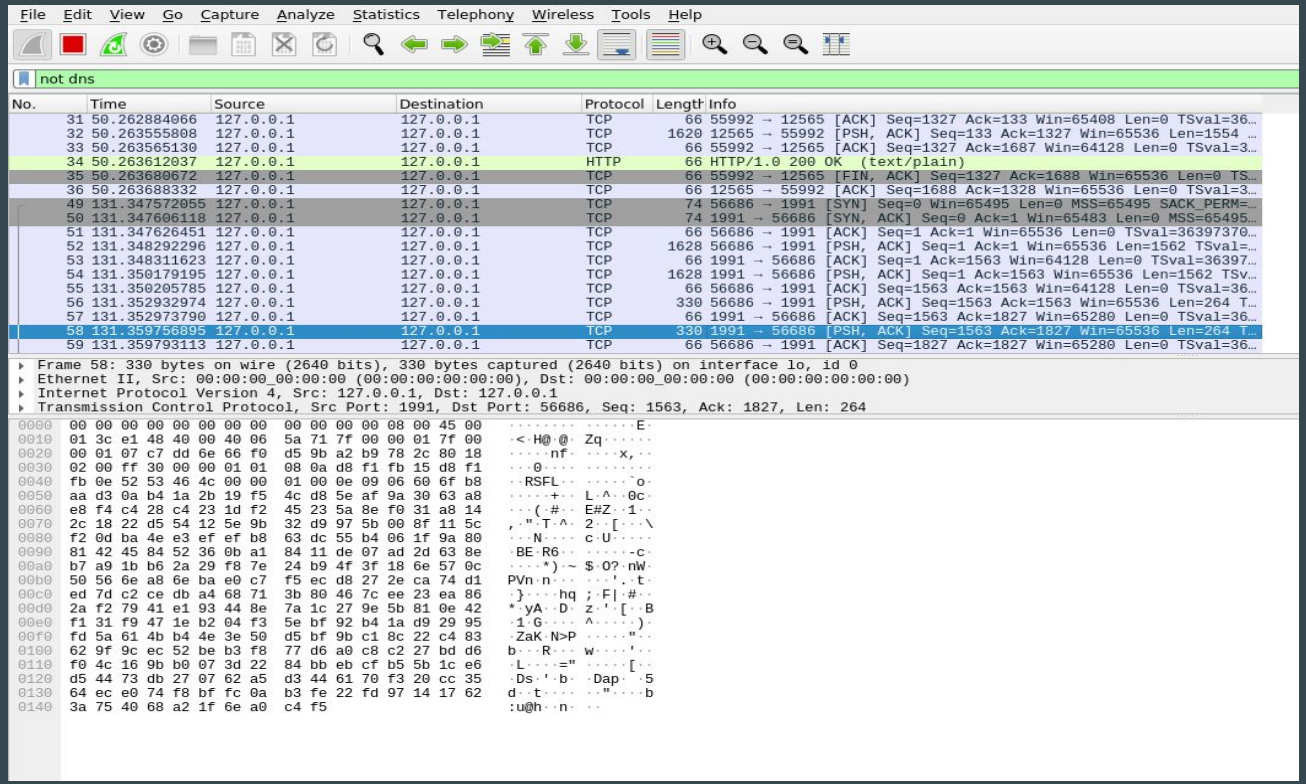
not dns

No.	Time	Source	Destination	Protocol	Length	Info
9	38.604211768	127.0.0.1	127.0.0.1	TCP	74	55990 → 12565 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM…
10	38.604236453	127.0.0.1	127.0.0.1	TCP	74	12565 → 55990 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=6549…
11	38.604249603	127.0.0.1	127.0.0.1	TCP	66	55990 → 12565 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3639644…
12	38.604285970	127.0.0.1	127.0.0.1	TCP	241	55990 → 12565 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=175 TSval=…
13	38.604290603	127.0.0.1	127.0.0.1	TCP	66	12565 → 55990 [ACK] Seq=1 Ack=176 Win=65408 Len=0 TSval=36396…
14	38.604305898	127.0.0.1	127.0.0.1	HTTP	1217	POST /verify HTTP/1.1
15	38.604309486	127.0.0.1	127.0.0.1	TCP	66	12565 → 55990 [ACK] Seq=1 Ack=1327 Win=64384 Len=0 TSval=3639…
16	38.690805293	127.0.0.1	127.0.0.1	TCP	198	12565 → 55990 [PSH, ACK] Seq=1 Ack=1327 Win=65536 Len=132 TSv…
17	38.690822521	127.0.0.1	127.0.0.1	TCP	66	55990 → 12565 [ACK] Seq=1327 Ack=133 Win=65408 Len=0 TSval=36…
18	38.690854938	127.0.0.1	127.0.0.1	TCP	1620	12565 → 55990 [PSH, ACK] Seq=133 Ack=1327 Win=65536 Len=1554 …
19	38.690859732	127.0.0.1	127.0.0.1	TCP	66	55990 → 12565 [ACK] Seq=1327 Ack=1687 Win=64128 Len=0 TSval=3…
20	38.690908130	127.0.0.1	127.0.0.1	HTTP	66	HTTP/1.0 200 OK (text/plain)
21	38.691367826	127.0.0.1	127.0.0.1	TCP	66	55990 → 12565 [FIN, ACK] Seq=1327 Ack=1688 Win=65536 Len=0 TS…
22	38.691377339	127.0.0.1	127.0.0.1	TCP	66	12565 → 55990 [ACK] Seq=1688 Ack=1328 Win=65536 Len=0 TSval=3…
23	50.201028927	127.0.0.1	127.0.0.1	TCP	74	55992 → 12565 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM…
24	50.201042646	127.0.0.1	127.0.0.1	TCP	74	12565 → 55992 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=6549…
25	50.201051171	127.0.0.1	127.0.0.1	TCP	66	55992 → 12565 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3639655…
26	50.201076996	127.0.0.1	127.0.0.1	TCP	241	55992 → 12565 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=175 TSval=…
27	50.201080517	127.0.0.1	127.0.0.1	TCP	66	12565 → 55992 [ACK] Seq=1 Ack=176 Win=65408 Len=0 TSval=36396…
28	50.201090542	127.0.0.1	127.0.0.1	HTTP	1217	POST /verify HTTP/1.1
29	50.201093125	127.0.0.1	127.0.0.1	TCP	66	12565 → 55992 [ACK] Seq=1 Ack=1327 Win=64384 Len=0 TSval=3639…
30	50.262868594	127.0.0.1	127.0.0.1	TCP	198	12565 → 55992 [PSH, ACK] Seq=1 Ack=1327 Win=65536 Len=132 TSv…
31	50.262884066	127.0.0.1	127.0.0.1	TCP	66	55992 → 12565 [ACK] Seq=1327 Ack=133 Win=65408 Len=0 TSval=36…
32	50.263555808	127.0.0.1	127.0.0.1	TCP	1620	12565 → 55992 [PSH, ACK] Seq=133 Ack=1327 Win=65536 Len=1554 …
33	50.263565130	127.0.0.1	127.0.0.1	TCP	66	55992 → 12565 [ACK] Seq=1327 Ack=1687 Win=64128 Len=0 TSval=3…
34	50.263612037	127.0.0.1	127.0.0.1	HTTP	66	HTTP/1.0 200 OK (text/plain)
35	50.263680672	127.0.0.1	127.0.0.1	TCP	66	55992 → 12565 [FIN, ACK] Seq=1327 Ack=1688 Win=65536 Len=0 TS…
36	50.263688332	127.0.0.1	127.0.0.1	TCP	66	12565 → 55992 [ACK] Seq=1688 Ack=1328 Win=65536 Len=0 TSval=3…
49	131.347572655	127.0.0.1	127.0.0.1	TCP	74	56686 → 1991 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=…
50	131.347606118	127.0.0.1	127.0.0.1	TCP	74	1991 → 56686 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495…
51	131.347626451	127.0.0.1	127.0.0.1	TCP	66	56686 → 1991 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=36397370…
52	131.348292296	127.0.0.1	127.0.0.1	TCP	1628	56686 → 1991 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=1562 TSval=…
53	131.348311623	127.0.0.1	127.0.0.1	TCP	66	1991 → 56686 [ACK] Seq=1 Ack=1563 Win=64128 Len=0 TSval=36397…
54	131.350179195	127.0.0.1	127.0.0.1	TCP	1628	1991 → 56686 [PSH, ACK] Seq=1 Ack=1563 Win=65536 Len=1562 TSv…
55	131.350205785	127.0.0.1	127.0.0.1	TCP	66	56686 → 1991 [ACK] Seq=1563 Ack=1563 Win=64128 Len=0 TSval=36…
56	131.352932974	127.0.0.1	127.0.0.1	TCP	330	56686 → 1991 [PSH, ACK] Seq=1563 Ack=1563 Win=65536 Len=264 T…
57	131.352973790	127.0.0.1	127.0.0.1	TCP	66	1991 → 56686 [ACK] Seq=1563 Ack=1827 Win=65280 Len=0 TSval=36…
58	131.359756895	127.0.0.1	127.0.0.1	TCP	330	1991 → 56686 [PSH, ACK] Seq=1563 Ack=1827 Win=65536 Len=264 T…
59	131.359793113	127.0.0.1	127.0.0.1	TCP	66	56686 → 1991 [ACK] Seq=1827 Ack=1827 Win=65280 Len=0 TSval=36…
60	131.360011036	127.0.0.1	127.0.0.1	TCP	66	1991 → 56686 [FIN, ACK] Seq=1827 Ack=1827 Win=65536 Len=0 TSv…
61	131.365946904	127.0.0.1	127.0.0.1	TCP	66	56686 → 1991 [FIN, ACK] Seq=1827 Ack=1828 Win=65536 Len=0 TSv…
62	131.365982714	127.0.0.1	127.0.0.1	TCP	66	1991 → 56686 [ACK] Seq=1828 Ack=1828 Win=65536 Len=0 TSval=36…
327	264.589950572	127.0.0.1	127.0.0.1	TCP	74	59594 → 1990 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=…
328	264.589995572	127.0.0.1	127.0.0.1	TCP	74	1990 → 59594 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495…
329	264.590034845	127.0.0.1	127.0.0.1	TCP	66	59594 → 1990 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=36398703…
330	264.590990246	127.0.0.1	127.0.0.1	TCP	1628	59594 → 1990 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=1562 TSval=…
331	264.591017420	127.0.0.1	127.0.0.1	TCP	66	1990 → 59594 [ACK] Seq=1 Ack=1563 Win=64128 Len=0 TSval=36398…
332	264.595769985	127.0.0.1	127.0.0.1	TCP	1628	1990 → 59594 [PSH, ACK] Seq=1 Ack=1563 Win=65536 Len=1562 TSv…
333	264.595810190	127.0.0.1	127.0.0.1	TCP	66	59594 → 1990 [ACK] Seq=1563 Ack=1563 Win=64128 Len=0 TSval=36…
334	264.598920864	127.0.0.1	127.0.0.1	TCP	330	59594 → 1990 [PSH, ACK] Seq=1563 Ack=1563 Win=65536 Len=264 T…
335	264.598967731	127.0.0.1	127.0.0.1	TCP	66	1990 → 59594 [ACK] Seq=1563 Ack=1827 Win=65280 Len=0 TSval=36…
336	264.605043835	127.0.0.1	127.0.0.1	TCP	330	1990 → 59594 [PSH, ACK] Seq=1563 Ack=1827 Win=65536 Len=264 T…
337	264.605078270	127.0.0.1	127.0.0.1	TCP	66	59594 → 1990 [ACK] Seq=1827 Ack=1827 Win=65280 Len=0 TSval=36…
338	264.605402408	127.0.0.1	127.0.0.1	TCP	330	1990 → 59594 [PSH, ACK] Seq=1827 Ack=1827 Win=65536 Len=264 T…
339	264.605419917	127.0.0.1	127.0.0.1	TCP	66	59594 → 1990 [ACK] Seq=1827 Ack=2091 Win=65024 Len=0 TSval=36…
340	264.605742955	127.0.0.1	127.0.0.1	TCP	330	1990 → 59594 [PSH, ACK] Seq=2091 Ack=1827 Win=65536 Len=264 T…
341	264.605756750	127.0.0.1	127.0.0.1	TCP	66	59594 → 1990 [ACK] Seq=1827 Ack=2355 Win=64768 Len=0 TSval=36…
342	264.605992226	127.0.0.1	127.0.0.1	TCP	66	1990 → 59594 [FIN, ACK] Seq=2355 Ack=1827 Win=65536 Len=0 TSv…
343	264.627517575	127.0.0.1	127.0.0.1	TCP	66	59594 → 1990 [FIN, ACK] Seq=1827 Ack=2356 Win=65536 Len=0 TSv…
344	264.627561873	127.0.0.1	127.0.0.1	TCP	66	1990 → 59594 [ACK] Seq=2356 Ack=1828 Win=65536 Len=0 TSval=36…

Unencrypted certificate



Encrypted data



Contributions

1. Ritik Kumar - Contributed to the actual coding the implementation together with looking at the available libraries to **handle encryption and key management**. Contributed to the **terminal based demonstration** for the project.
2. Kaustubh Trivedi - Contributed in the actual code, specifically **connections.py and peer.py**, where I handled the encryption and sending of data as bytes chunks from file. Contributed to **make the tool interactive** and demonstrate its POC.
3. Bhavye Jain - Created the **overall design** of the system encompassing the **various entities, their roles and interactions**. Evaluated the various options of signing certificates and decided on the hierarchy suitable for the project.
4. Saurabh Singh - Worked on **securing file transfer** using Sockets and developing an **architecture that works similar to SSL/TLS** security on the internet to ensure that communication is encrypted.
5. Ajay Dayma - Worked on security issues of File transfer and how we can improve the existing solution. **Explored different ways of securing communication**, specifically JWT.
6. Sai Krishna Abhiram - Packet Capture and Analysis using Wireshark
7. Shiva Reddy - Worked on what **assumptions to take for our System**. Explored different ways of securing communication, specifically **Symmetric Cryptography**.

Thank You