

Paras Gupta  
19110128

**CS 433**  
**Computer Networks**  
**Assignment 1**

**Part I: Network Application using Socket Programming**  
**Q1.**

---

**(a) Source Code:** The working code implementation for the Remote File System (RFS) service has been uploaded to [GitHub](#). One can refer to the readme file for details on code execution.

---

**(b) Design Document:** *Network Application using Socket Programming*

This design document details the following:

1. Modes of Layering
2. Protocol Design
3. Associated Challenges
4. Code Execution Results

**Modes of Layering**

Layering has been done following the same template provided in the question description. The layering principles for each of the above layers can be mapped to specific layers in the stack for the OSI reference model.

The OSI or the Open Systems Interconnection model provides seven layers to facilitate communication across a network. The seven layers, in a top to down order, include *Application Layer*, *Presentation Layer*, *Session Layer*, *Transport Layer*, *Network Layer*, *Data Link Layer*, and *Physical Layer*. The three layers in our RFS system, with the corresponding mappings to the OSI model, are given below:

**1) File Service Layer :: Application Layer**

The File Service Layer of our system forms the application layer of the network. The files **service\_client.py** and **service\_server.py** in the Client and Server directories respectively of the submitted code contain this layer. The files contain utilities that form the data to be transmitted across the client-server link and also allow displaying received or other meaningful information at both ends. The layer makes use of the services of the Crypto Layer, as detailed in the Protocol Design Section.

## **2) Crypto Layer :: Presentation Layer**

The Crypto Layer of our system forms the presentation layer of the network. The files **server\_crypt.py** and **client\_crypt.py** in the Client and Server directories respectively of the submitted code contain this layer. The files allow for manipulation of the data before sending it over the network using encryption and decryption techniques given below:

- Plain Text - Sending the data as it is
- Substitute - Substituting alphanumeric characters with fixed offset before sending
- Transpose - Reversing the contents in a word-by-word manner before sending

The layer makes use of the services of the Network Layer, as detailed in the protocol design section.

## **3) Networking Layer :: Transport Layer**

The Networking Layer of our system forms the transport layer of the network. The files **server.py** and **client.py** in the Client and Server directories of the submitted code contain this layer. The files contain the source code to establish a non-persistent connection between server and client using the socket package in python. The layer sends and transmits the data received from the upper layer, across the network.

## **Protocol Design**

### **a) Client**

Upon starting the client (see [README.md](#) on GitHub), the **File Service Layer** asks the user for the encryption mode and the service that is to be requested from the server. The input is then checked to see if the service is the upload or “upd” service, which requires reading and sending the contents of a file present in the client along with the service information. The File service layer interacts with OS to read the file contents.

The service information includes service command and other input metadata like file or directory path. The File Service Layer appends the input encryption mode as a **header** to the service information. The complete unit or protocol data unit (PDU) is then passed down to the **Crypto Layer** as its service data unit (SDU).

The Crypto Layer file contains utilities that allow the encryption and decryption of data. Since its SDU needs to be sent across the network, the Crypto Layer encrypts the SDU based on the encryption mode header present in the SDU, forms the PDU of the Crypto Layer, and then passes it down to the third layer, the **Networking Layer**.

The Networking Layer, upon receiving the data unit from the Crypto Layer, establishes a socket connection the running client and the server using **IPv4** (Internet Protocol Version 4) over **TCP** (Transmission Control Protocol), and sends the data unit to the server.

### b) Server

The **Networking Layer** of the server receives the data units from the Networking Layer of the client. The data is then passed up to the **Crypto Layer** of the server where it is decrypted using the encryption mode information available in the form of a header. The decrypted data is next passed to the **File Service Layer** of the server as its SDU, where the requested service is extracted and separated from the mode header. The service request is then processed at the server, and the corresponding **service response is** gets ready to be transmitted through the network.

The service response follows a flow (top to down) across the three server-side network layers, similar to the one described above in the client protocol design. The response is transmitted to the client, and the data flows down to the top, to the File Service Layer of the Client which displays the response to the client user.

## Associated Challenges

The major challenge was to structure the whole code to form separate layers that used each other's services. Challenges and difficulties faced while implementing those layers are given below:

### 1) *File Service Layer :: Application Layer*

Parsing the input data, appending header information, checking for certain services and processing them, and displaying information to the user, were some of the major tasks performed in this layer.

Initially, the header containing the encryption mode was sent through a separate send operation to the server. Although it wouldn't have caused many problems in the submitted python code, if a packet loss were to occur when the header was being sent, it would have become difficult to decrypt any data being sent next.

The download and upload services required different processing from the other services and were prone to various errors. The thought process involved for these services was longer and it was more difficult to code them.

### 2) *Crypto Layer :: Presentation Layer*

Encryption and Decryption using substitute and transpose posed some challenges. The substitute and transpose methods were initially implemented using the split operation in python that did not account for line breaks. The code was changed to account for line breaks.

### 3) Networking Layer :: Transport Layer

The socket package in python was used for the first time, and so it was difficult to debug whenever errors were encountered. The final submitted code establishes a non-persistent connection between the client and the server. It was difficult to think of the steps to establish a persistent connection.

## Code Execution Results

The initial directory structure and the responses from the server (left terminal) and the client (right terminal) are shown below:

The screenshot shows a terminal window with the following content:

```
(base) C:\Users\Paras_Gupta\CS433_Assignment_1>cd Client
(base) C:\Users\Paras_Gupta\CS433_Assignment_1>python client.py
The following services can be requested from the server:
1. CWD - Retrieve the path of the current working directory for the user
2. LS - List the files/folders present in the current working directory
3. CD <dir> - Change the directory to <dir> as specified by the client
4. DWD <file> - Download the <file> specified by the user on server to client
5. UPD <file> - Upload the <file> on client to the remote server in CWD

The following encryption modes are available:
1 - Plain Text
2 - Substitute
3 - Transpose

Enter encryption mode: [ ]
```

The code execution for the five service commands is as follows:

### 1. CWD

The screenshot shows a terminal window with the following content:

```
(base) C:\Users\Paras_Gupta\CS433_Assignment_1>cd Server
(base) C:\Users\Paras_Gupta\CS433_Assignment_1>python server.py
Server listening...
Connection established with client having address: ('192.168.81.1', 49845)
[ ]

The following services can be requested from the server:
1. CWD - Retrieve the path of the current working directory for the user
2. LS - List the files/folders present in the current working directory
3. CD <dir> - Change the directory to <dir> as specified by the client
4. DWD <file> - Download the <file> specified by the user on server to client
5. UPD <file> - Upload the <file> on client to the remote server in CWD

The following encryption modes are available:
1 - Plain Text
2 - Substitute
3 - Transpose

Enter encryption mode: 2
Enter service: CWD

Establish connection with Server..
Connection established.

-----
Response from server: C:\Users\Paras_Gupta\CS433_Assignment_1\Server
-----
Service response received. Connection is now closed
```

The current working directory of the server is returned as a response to the client.

## 2. LS

The screenshot shows a terminal window within a code editor interface. The terminal tab is active, displaying the command `(base) C:\Users\Paras\_Gupta\CS433\_Assignment\_1\Client>python client.py` followed by the server's response. The server lists its own directory contents, which include `server.py`, `server\_crypt.py`, `server\_test.txt`, `service\_server.py`, and `\_\_pycache\_\_`. The terminal also shows the user entering encryption mode (2) and service (LS), establishing a connection, and receiving a list of files from the server. The session concludes with a service response received message.

```
(base) C:\Users\Paras_Gupta\CS433_Assignment_1\Client>python client.py
The following services can be requested from the server:
1. CWD - Retrieve the path of the current working directory for the user
2. LS - List the files/folders present in the current working directory
3. CD <dir> - Change the directory to <dir> as specified by the client
4. DWD <file> - Download the <file> specified by the user on server to client
5. UPD <file> - Upload the <file> on client to the remote server in CWD

The following encryption modes are available:
1 - Plain Text
2 - Substitute
3 - Transpose

Enter encryption mode: 2
Enter service: LS

Establish connection with Server..
Connection established.

-----
Response from server: List of directories
server.py
server_crypt.py
server_test.txt
service_server.py
__pycache__

-----
Service response received. Connection is now closed
```

The list of files and folders present in the server are listed out.

## 3. CD <dir>

The screenshot shows a terminal window within a code editor interface. The terminal tab is active, displaying the command `(base) C:\Users\Paras\_Gupta\CS433\_Assignment\_1\Client>python client.py` followed by the server's response. The server indicates that the directory was successfully changed. The terminal also shows the user entering encryption mode (2) and service (CD ..), establishing a connection, and receiving a status OK message from the server. The session concludes with a service response received message.

```
(base) C:\Users\Paras_Gupta\CS433_Assignment_1\Client>python client.py
The following services can be requested from the server:
1. CWD - Retrieve the path of the current working directory for the user
2. LS - List the files/folders present in the current working directory
3. CD <dir> - Change the directory to <dir> as specified by the client
4. DWD <file> - Download the <file> specified by the user on server to client
5. UPD <file> - Upload the <file> on client to the remote server in CWD

The following encryption modes are available:
1 - Plain Text
2 - Substitute
3 - Transpose

Enter encryption mode: 2
Enter service: CD ..

Establish connection with Server..
Connection established.

-----
Response from server: Status:OK
-----

Service response received. Connection is now closed

(base) C:\Users\Paras_Gupta\CS433_Assignment_1\Client>python client.py
```

The current working directory of the server is requested to be changed. Response status is shown to be OK which means that the task has been completed successfully. This can be verified by the following CWD service request as shown below.

```
CS433_ASSIGNMENT_1
> _pycache_
  > Client
    > _pycache_
      > client_crypt.py M
      & client_test.txt M
      & client.py
      & service_client.py
  < Server
    > _pycache_
      & server_crypt.py M
      & server_test.txt M
      & server.py
      & service_server.py
  & .gitignore U
  & README.md

(base) C:\Users\Paras_Gupta\CS433_Assignment_1>cd Server
(base) C:\Users\Paras_Gupta\CS433_Assignment_1>python server.py
Server listening....
Connection established with client having address: ('192.168.81.1', 49845)
Connection established with client having address: ('192.168.81.1', 49859)
Connection established with client having address: ('192.168.81.1', 49868)
Connection established with client having address: ('192.168.81.1', 49871)

Enter service: CD ..
Establish connection with Server..
Connection established.

-----
Response from server: Status:OK
-----

Service response received. Connection is now closed

(base) C:\Users\Paras_Gupta\CS433_Assignment_1>python client.py

The following services can be requested from the server:
  1. CWD - Retrieve the path of the current working directory for the user
  2. LS - List the files/folders present in the current working directory
  3. CD <dir> - Change the directory to <dir> as specified by the client
  4. DWD <file> - Download the <file> specified by the user on server to client
  5. UPD <file> - Upload the <file> on client to the remote server in CWD

The following encryption modes are available:
  1 - Plain Text
  2 - Substitute
  3 - Transpose

Enter encryption mode: 2
Enter service: CWD

Establish connection with Server..
Connection established.

-----
Response from server: C:\Users\Paras_Gupta\CS433_Assignment_1
-----

Service response received. Connection is now closed
```

We can see the current working directory has indeed been changed.

#### 4. DWD <file>

The screenshot shows a Visual Studio Code interface with two panes. The left pane displays a file tree for a project named 'CS433\_Assignment'. It includes files for both 'Client' and 'Server' components, such as 'client\_crypt.py', 'client\_test.txt', 'server\_crypt.py', and 'server\_test.txt'. The right pane shows two terminal windows. The top terminal window, titled 'downloaded\_file.txt M', shows the contents of a file received from the server. The bottom terminal window, titled 'server\_test.txt M', shows the contents of a file sent to the client. Both files contain encrypted binary data. Below the terminals is a command palette with options like 'cmd', '+', 'cmd', and 'cmd'. The status bar at the bottom indicates the service response received.

```
(base) C:\Users\Paras_Gupta\CS433_Assignment_1>python server.py
Server listening....
Connection established with client having address: ('192.168.81.1', 49904)
Connection established with client having address: ('192.168.81.1', 49911)

The following encryption modes are available:
1 - Plain Text
2 - Substitute
3 - Transpose

Enter encryption mode: 2
Enter service: DWD server_test.txt

Establish connection with Server..
Connection established.

-----
Response from server: Status:OK
File downloaded succesfully into downloaded_file.txt
-----

Service response received. Connection is now closed
```

A file present in the server directory is requested for download. Response status is shown to be OK which means that the task has been completed successfully.

We can see in the editor files open that the file has indeed been downloaded to a file named downloaded\_file.txt. Note that the file was not present earlier.

## 5. UPD <file>

The screenshot shows the VS Code interface with two editor panes. The left pane contains the client code, and the right pane contains the server code. Both files show identical code for handling UPD requests.

```

Client > client_test.txt
1 Client
2 ****&73739739gubhj3c 3gf3 f3nch offlawflawflufguw By@yrg9 3r98r Br Br 2gtg2g
3
4
5
6 #$$$$$*)((**$@#$%^&*))(
7
8
9
10 hjj@

Server > uploaded_file.txt
1 Client
2 ****&73739739gubhj3c 3gf3 f3nch offlawflawflufguw By@yrg9 3r98r Br Br 2gtg2g
3
4
5
6
7 #$$$$$*)((**$@#$%^&*))(
8
9
10 hjj@

```

The terminal below the editors shows the execution of the server script and the interaction with the client:

```

The following encryption modes are available:
1 - Plain Text
2 - Substitute
3 - Transpose

Enter encryption mode: 2
Enter service: UDP client_test.txt

Establish connection with Server..
Connection established.

-----
Response from server: Status:OK
File uploaded successfully to server
-----

Service response received. Connection is now closed

```

A file in the client directory is requested to be uploaded to the server. Response status is shown to be OK which means that the task has been completed successfully.

We can see in the editor files open that the file has indeed been uploaded to a file named uploaded\_file.txt. Note that the file was not present earlier.

### (c) Wireshark dump and Analysis

The Wireshark dump and analysis indicating that data was correctly encrypted before requesting a service (client) and before sending the response (server) has been done for all the three encryption modes as below:

#### 1. Plain Text

```

(kali㉿kali)-[~/Downloads/CS433_Assignment_1-main/Server]
$ python server.py
Connection received. Connection is now closed!
Server listening....
Connection established with client having address: ('127.0.0.1', 51814)

Encrypted Service Response
Response: /home/kali/Downloads/CS433_Assignment_1-main/Server
Hex Response 2f686f6d652f6b616c692f446f776e6c6f6164732f43533433335f41737369676e6d656e745f312d6d61696e2f536572766572

```

**(1a) Server Response Encryption**

```
(kali㉿kali)-[~/Downloads/CS433_Assignment_1-main/Client]$ python client.py
The following services can be requested from the server:
 1. CWD - Retrieve the path of the current working directory for the user
 2. LS - List the files/folders present in the current working directory
 3. CD <dir> - Change the directory to <dir> as specified by the client
 4. DWD <file> - Download the <file> specified by the user on server to client
 5. UPD <file> - Upload the <file> on client to the remote server in CWD

The following encryption modes are available:
 1 - Plain Text
 2 - Substitute
 3 - Transpose

Enter encryption mode: 1
Enter service: CWD

Encrypted Service Request
Request: 1 CWD
Hex Request: 3120435744

Establish connection with Server..
Connection established.

Response from server: /home/kali/Downloads/CS433_Assignment_1-main/Server

Service response received. Connection is now closed
```

### **(1b) Client Response Encryption**

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.1.1	TCP	76	51814 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=427638739 TSecr=0 WS=128
2	0.000016869	127.0.1.1	127.0.0.1	TCP	76	5000 → 51814 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3677911017 TSecr=427638739 WS=128
3	0.000033127	127.0.0.1	127.0.1.1	TCP	68	51814 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=427638739 TSecr=3677911017
4	0.000236322	127.0.1.1	127.0.1.1	TCP	73	51814 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=5 TSval=427638739 TSecr=3677911017
5	0.000249123	127.0.1.1	127.0.0.1	TCP	68	5000 → 51814 [ACK] Seq=1 Ack=6 Win=65536 Len=0 TSval=3677911017 TSecr=427638739
6	0.000661666	127.0.1.1	127.0.0.1	TCP	119	5000 → 51814 [PSH, ACK] Seq=1 Ack=6 Win=65536 Len=51 TSval=3677911017 TSecr=427638739
7	0.000667260	127.0.1.1	127.0.1.1	TCP	68	51814 → 5000 [ACK] Seq=6 Ack=52 Win=65536 Len=0 TSval=427638739 TSecr=3677911017
8	0.000853058	127.0.1.1	127.0.1.1	TCP	68	51814 → 5000 [FIN, ACK] Seq=6 Ack=52 Win=65536 Len=0 TSval=427638739 TSecr=3677911017
9	0.042789874	127.0.1.1	127.0.1.1	TCP	68	5000 → 51814 [ACK] Seq=52 Ack=7 Win=65536 Len=0 TSval=3677911059 TSecr=427638739

### **(1c) Wireshark Request Packet Capture**

No.	Time	Source	Destination	Protocol	Length	Info
1	00:00:00:00:00:00	127.0.0.1	127.0.0.1	TCP	76	51814 - 5000 [SYN] Seq=0 Win=65536 Len=0 MSS=65536 SACK_PERM=1 Tsvl=427638739 TSscr=0 WS=128
2	0.0000100000000000	127.0.0.1	127.0.0.1	TCP	76	50898 - 51814 [SYN, ACK] Seq=0 Win=65536 Len=0 MSS=65536 SACK_PERM=1 Tsvl=427638739 TSscr=0 WS=128
3	0.0000033127	127.0.0.1	127.0.0.1	TCP	68	51814 - 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 Tsvl=427638739 TSscr=3677911017
4	0.0000236322	127.0.0.1	127.0.0.1	TCP	73	51814 - 5000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=5 Tsvl=427638739 TSscr=3677911017
5	0.0000249123	127.0.0.1	127.0.0.1	TCP	68	5000 - 51814 [ACK] Seq=1 Ack=6 Win=65536 Len=0 Tsvl=3677911017 TSscr=427638739
6	0.0000161566	127.0.0.1	127.0.0.1	TCP	119	5000 - 51814 [PSH, ACK] Seq=1 Ack=6 Win=65536 Len=51 Tsvl=3677911017 TSscr=427638739
7	0.0000676266	127.0.0.1	127.0.0.1	TCP	68	51814 - 5000 [ACK] Seq=6 Ack=52 Win=65536 Len=0 Tsvl=427638739 TSscr=3677911017
8	0.0000853858	127.0.0.1	127.0.0.1	TCP	68	51814 - 5000 [FIN, ACK] Seq=6 Ack=52 Win=65536 Len=0 Tsvl=427638739 TSscr=3677911017
9	0.042768874	127.0.0.1	127.0.0.1	TCP	68	5000 - 51814 [ACK] Seq=52 Ack=7 Win=65536 Len=0 Tsvl=3677911059 TSscr=427638739

### **(1d) Wireshark Response Packet Capture**

From the **(1a)** and **(1b)** screenshots above, we can see how the request and response messages were encrypted using **Plain Text encryption mode**, that is data sent as it is, before sending and receiving them across the network.

From the **(1c)** and **(1d)** screenshots above, we can see how the request and response messages were transmitted in the network, as captured by Wireshark.

It can be observed that encrypted request and response messages are the same when transmitted in the network as packets, indicating that the data was correctly encrypted.

## 2. Substitute

#### **(2a) Server Response Encryption**

```
(kali㉿kali)-[~/Downloads/CS433_Assignment_1-main/Client]
$ python client.py

The following services can be requested from the server:1.1
 1. CWD - Retrieve the path of the current working directory for the user
 2. LS - List the files/folders present in the current working directory
 3. CD <dir> - Change the directory to <dir> as specified by the client
 4. DWD <file> - Download the <file> specified by the user on server to client
 5. UPD <file> - Upload the <file> on client to the remote server in CWD

The following encryption modes are available: 127.0.0.1
 1 - Plain Text
 2 - Substitute
 3 - Transpose

Enter encryption mode: 2
Enter service: LS

Encrypted Service Request
Request: 2 NU
Hex Request: 32204e55 06 00 00 00 00 00 00 00 00 08 00
Establish connection with Server...
Connection established.

Response from server: List of directories
service_server.py
server_crypt.py
server_test.txt
__pycache__
uploaded_file.txt
server.py

Service response received. Connection is now closed
```

### **(2b) Client Response Encryption**

## **(2c) Wireshark Request Packet Capture**

No.	Time	Source	Destination	Protocol	Length Info
1	10:00:00.000000	fe80::d901:4aeb:e2f...	ff02::1	ICMPv6	64 Router Solicitation
2	10:00:00.000000	127.0.0.1	127.0.0.1	TCP	70 45692 - 5000 [SYN] Seq=0 Win=65536 Len=0 MSS=65495 SACK_PERM=1 Tsvl=427946405 Tsecr=0 WS=128
3	10:00:00.000000	127.0.0.1	127.0.0.1	TCP	76 5000 45692 [SYN, Ack] Seq=0 Ack=1 Win=65493 Len=0 MSS=65495 SACK_PERM=1 Tsvl=3678218683 Tsecr=427946405 WS=128
4	10:00:00.000000	127.0.0.1	127.0.0.1	TCP	68 45692 - 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 Tsvl=427946405 Tsecr=3678218683
5	10:00:00.000000	127.0.0.1	127.0.0.1	TCP	72 45692 - 5000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 Tsvl=427946405 Tsecr=3678218683
6	10:00:00.000000	127.0.0.1	127.0.0.1	TCP	68 5000 - 45692 [ACK] Seq=1 Ack=5 Win=65536 Len=0 Tsvl=3678218683 Tsecr=427946405
7	10:00:00.000000	127.0.0.1	127.0.0.1	TCP	179 5000 - 45692 [PSH, ACK] Seq=1 Ack=5 Win=65536 Len=11 Tsvl=3678218684 Tsecr=427946405
8	10:00:00.000000	127.0.0.1	127.0.0.1	TCP	68 45692 - 5000 [ACK] Seq=5 Ack=112 Win=65536 Len=0 Tsvl=427946406 Tsecr=3678218684
9	10:00:00.000000	127.0.0.1	127.0.0.1	TCP	68 45692 - 5000 [FIN, ACK] Seq=5 Ack=112 Win=65536 Len=0 Tsvl=427946407 Tsecr=3678218684
10	10:00:00.000000	127.0.0.1	127.0.0.1	TCP	68 5000 - 45692 [ACK] Seq=112 Ack=6 Win=65536 Len=0 Tsvl=3678218727 Tsecr=427946407
Frame 7: 179 bytes on wire (1432 bits), 179 bytes captured (1432 bits) on interface any, id 0					
Linux cooked capture v1					
Internet Protocol Version 4, Src: 127.0.1.1, Dst: 127.0.0.1					
Transmission Control Protocol, Src Port: 5000, Dst Port: 45692, Seq: 1, Ack: 5, Len: 111					
Data (111 bytes)					
data = 4e6b7c5/620716820665b/46765/67174606/75208/567/4766b55/57567/747867742e					
[Length: 111]					
0000	00:00:03:04:00:00	00:00:00:00:00:00	00:00:00:00:00:00	---	---
0001	45 00 00 03 89 2e	40 00 49 00 61 b1 78	00 01 01	E .. . @ @ P ..	-----
0002	7f 00 00 01 13 88	b2 7c 86 48 5f bb cd d8 07 5d	-----	H .. . ]	-----
0003	18 00 02 ff	97 00 00 00 01 01 08 0a dd 3d 29 bc	-----	= )	-----
0004	19 81 f1 a5 4e	6b 75 76 29 71 68 66 74 67	-----	Nkuv qh fktg	-----
0005	65 76 71 74 6b	75 76 20 75 67 74 78 6b 65	-----	evqtku -ugtkxeg	-----
0006	74 6b 76 75 76	76 75 76 75 76 75 76 75 76	-----	ugtkxeg -ugtkxeg	-----
0007	74 5f 65 74 71	72 76 76 72 61 78 67 74 68	-----	t_stvwv -rv_dvqj	-----
0008	74 5f 67 75 76 76	76 74 6b 76 75 76 75 76 75	-----	t_vquv v_zv -rae	-----
0009	63 65 66 67 5f 5b	80 77 72 66 71 63 66 67 66 5f	-----	cejg -w_rnqcf	-----
000a	68 6b 6e 67 2e 76	78 76 76 75 67 74 78 67 74 2e	-----	hkng_vzv -ugtxgt.	-----
000b	72 61 6a	ra.	-----		

## **(2d) Wireshark Response Packet Capture**

From the **(2a)** and **(2b)** screenshots above, we can see how the request and response messages were encrypted using **Substitute encryption mode**, that is data sent with alphanumeric characters substituted with a fixed offset, before sending and receiving them across the network.

From the **(2c)** and **(2d)** screenshots above, we can see how the request and response messages were transmitted in the network, as captured by Wireshark.

It can be observed that encrypted request and response messages are the same when transmitted in the network as packets, indicating that the data was correctly encrypted.

### 3. Transpose

```
(kali㉿kali)-[~/Downloads/CS433_Assignment_1-main/Server]
└─$ python server.py
server listening...
connection established with client having address: ('127.0.0.1', 49582)
Encrypted Service Response
Response: K0sutalS revr05
c3jhbg9379376**** * fg3 honf uwgfuiwlifiawiffo 9ry9by8 r89r3 r8 r8 g2gtg2

)))(+*%$@@@$*((")$@%#"

ajjh
Hex Response: 4bf4fa3737574617453207265767265530a0a63336a686275673933373933373337262a2a2a203366733208686f6e3366207577667569667716c696677616966666f2039727939387938207238397233207238207238206732077467320a0a0a292929282a265e2524404040
25e2a28289562426254230a0a616a68

[!] No new connections in progress.
[!] Packets: B: Discarded: 0 (0.00%)
```

### **(3a) Server Response Encryption**

```
└─(kali㉿kali)-[~/Downloads/CS433_Assignment_1-main/Client]
└─$ python client.py
```

The following services can be requested from the server:

1. CWD - Retrieve the path of the current working directory for the user
2. LS - List the files/folders present in the current working directory
3. CD <dir> - Change the directory to <dir> as specified by the client
4. DWD <file> - Download the <file> specified by the user on server to client
5. UPD <file> - Upload the <file> on client to the remote server in CWD

The following encryption modes are available:

- 1 - Plain Text
- 2 - Substitute
- 3 - Transpose

Enter encryption mode: 3

Enter service: DWD server\_test.txt

Encrypted Service Request

Request: 3 DWD txt.tset\_revres

Hex Request: 3320445744207478742e747365745f726576726573

Establish connection with Server..

Connection established.

---

Response from server: Status:OK  
File downloaded successfully into downloaded\_file.txt

---

Service response received. Connection is now closed

### (3b) Client Response Encryption

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000000	127.0.0.1	127.0.1.1	TCP	76	49582 - 5800 [SYN] Seq=0 Win=65405 MSS=65405 SACK_PERM=1 Tsv1=4282716767 Tscr=0 WS=128
2	0.000026594	127.0.1.1	127.0.0.1	TCP	76	50000 - 49582 [SYN, ACK] Seq=8 Ack=1 Win=65483 Len=0 MSS=65405 SACK_PERM=1 Tsv1=3678543956 Tscr=428271678 WS=128
3	0.000048264	127.0.0.1	127.0.1.1	TCP	68	49582 - 5800 [ACK] Seq=1 Ack=1 Win=65536 Len=0 Tsv1=428271678 Tscr=3678543956
4	0.000282298	127.0.0.1	127.0.1.1	TCP	89	49582 - 5800 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=21 Tsv1=428271678 Tscr=3678543956
5	0.000293658	127.0.1.1	127.0.0.1	TCP	68	50000 - 49582 [ACK] Seq=22 Win=65536 Len=0 Tsv1=3678543956 Tscr=428271678
6	0.001053446	127.0.1.1	127.0.0.1	TCP	199	50000 - 49582 [PSH, ACK] Seq=22 Ack=22 Win=65536 Len=181 Tsv1=3678543957 Tscr=428271678
7	0.001104552	127.0.0.1	127.0.1.1	TCP	68	49582 - 5800 [ACK] Seq=22 Ack=132 Win=65408 Len=0 Tsv1=428271678 Tscr=3678543957
8	0.002197602	127.0.0.1	127.0.1.1	TCP	68	49582 - 5800 [FIN, ACK] Seq=22 Ack=132 Win=65536 Len=0 Tsv1=428271680 Tscr=3678543957
9	0.002444109	127.0.1.1	127.0.0.1	TCP	68	50000 - 49582 [ACK] Seq=23 Ack=23 Win=65536 Len=0 Tsv1=3678543999 Tscr=428271680
10	0.002444109	127.0.0.1	127.0.1.1	ARP	68	Who has 192.168.81.2? Tell 192.168.81.1
						[Length: 21]
						Data: 3320445744207478742e747365745f726576726573
						[Length: 21]
0000	00 00 03 04 00 00 00 00 00 00 00 00 00 00 00 00	E .. I .. @ Hw ..				
0001	00 00 00 49 f3 35 40 00 00 00 49 77 ff 00 00 01	F ..				
0002	7f 00 01 01 c1 ae 13 08 46 68 f6 ba b7 aa 00 1e	..... = ..... >				
0003	80 18 02 00 ff 3d 00 00 01 00 08 0a 19 86 e8 3e	B T3 0W D txt.ts				
0040	db 42 20 54 33 20 44 57 44 20 74 78 74 26 74 73	et_revre s				
0050	65 74 57 72 65 76 72 65 73					

### (3c) Wireshark Request Packet Capture

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000000	127.0.0.1	127.0.1.1	TCP	76	49580 - 5000 [SYN] Seq=0 Win=65495 Len=8 MSS=65495 SACK_PERM=1 TSval=428271678 TSerr=0 WS=128
2	0.00000265594	127.0.1.1	127.0.0.1	TCP	76	50000 - 49582 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=9 MSS=65495 SACK_PERM=1 TSval=3678543956 TSerr=428271678 WS=128
3	0.00000448264	127.0.0.1	127.0.1.1	TCP	68	49582 - 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=9 TSval=428271678 TSerr=3678543956
4	0.000022298	127.0.0.1	127.0.1.1	TCP	89	49582 - 5000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=21 TSval=428271678 TSerr=3678543956
5	0.0000235658	127.0.1.1	127.0.0.1	TCP	68	50000 - 49582 [ACK] Seq=2 Ack=22 Win=65536 Len=0 TSval=3678543956 TSerr=428271678
6	0.0000353440	127.0.1.1	127.0.0.1	TCP	199	50000 - 49502 [PSH, ACK] Seq=1 Ack=22 Win=65536 Len=131 TSval=3678543957 TSerr=428271678
7	0.000114552	127.0.0.1	127.0.1.1	TCP	68	49582 - 5000 [ACK] Seq=22 Ack=132 Win=65408 Len=0 TSval=428271678 TSerr=3678543957
8	0.0002197661	127.0.0.1	127.0.1.1	TCP	68	49582 - 5000 [FIN, ACK] Seq=22 Ack=132 Win=65536 Len=0 TSval=428271680 TSerr=3678543957
9	0.003444109	127.0.1.1	127.0.0.1	TCP	68	50000 - 49582 [ACK] Seq=132 Ack=23 Win=65536 Len=0 TSval=3678543999 TSerr=428271680
10	65.177538798	Vmware c0:00:08		ARP	62	Who has 192.168.81.2? Tell 192.168.81.1

### **(3d) Wireshark Response Packet Capture**

From the **(3a)** and **(3b)** screenshots above, we can see how the request and response messages were encrypted using **Transpose encryption mode**, that is reversing the contents in a word-by-word manner, before sending and receiving them across the network.

From the (3c) and (3d) screenshots above, we can see how the request and response messages were transmitted in the network, as captured by Wireshark.

It can be observed that encrypted request and response messages are the same when transmitted in the network as packets, indicating that the data was correctly encrypted.

## Part II: Networking Problems

Q2.

CS 433 : Computer Networks

PARAS GUPTA

19110128

Assignment - 01

Part II : Networking Problems

$$2. (a) \text{Packet size, } p = 100 \text{ kB} + 100 \text{ B} \quad (\text{meta data for 1 packet}) \\ = 100 \times 1000 \times 8 \text{ bits}$$

1 packet

$$\text{Delivery time} = \text{delay} = \frac{p}{R_1} + \frac{p}{R_2} + \frac{p}{R_3} \quad \begin{cases} R_1 \rightarrow \text{Link 1} \\ R_2 \rightarrow \text{Link 2} \\ R_3 \rightarrow \text{Link 3} \end{cases} \\ = p \left( \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} \right) \\ = 100 \times 1000 \times 8 \left( \frac{1}{400 \times 10^6} + \frac{1}{1000 \times 10^6} + \frac{1}{200 \times 10^6} \right) \\ = 1000 \times 8 \times 10^{-6} \left( \frac{1}{400} + \frac{1}{1000} + \frac{1}{200} \right) \text{ seconds} \\ = 1000 \times 8 \times 10^{-6} \times 10^{-8} \text{ seconds} \\ = 1401400 \times 10^{-8} \text{ seconds} \\ = 14.014 \text{ ms} \quad \boxed{14.014 \text{ ms}}$$

$$(b) \text{Packet size, } p = \frac{1}{10} \times 100 \text{ kB} + 100 \text{ B} = 80800 \text{ bits}$$

10 packets

Through each link ( $L_1, L_2, L_3$ ), one packet can move at a time. Since,  $L_2$  has the least transmission rate, every  $(n+1)^{\text{th}}$  packet shall wait for  $t_2$  time (time taken to move across Link  $L_2$ ) relative to the  $n^{\text{th}}$  packet in order to move from A to B.

$$\text{Delay for 1st packet} = \frac{p}{R_1} + \frac{p}{R_3} + \frac{p}{R_2} = t_1 + t_3 + t_2$$

$$\text{Delay for 2nd packet} = (\text{Delay for first packet}) + t_2$$

$$\text{Delay for } n^{\text{th}} \text{ packet} = (\text{Delay for } (n-1)^{\text{th}} \text{ packet}) + t_2$$

$$= (t_1 + t_3 + (n-1)t_2) + t_2 = t_1 + t_3 + nt_2$$

Delivery time = Delay for lost packet

$$= t_1 + t_2 + n t_3$$

Here,  $n = 10$

$$\therefore \text{Delivery time} = \frac{P}{R_1} + \frac{P}{R_2} + 10 \times \frac{P}{R_3}$$

$$= 808 \times 10^{-6} \left( \frac{1}{144 \times 10^6} + \frac{1}{244 \times 10^6} + 10 \times \frac{1}{124 \times 10^6} \right)$$

$$= 808 \times 10^{-6} \times \left( \frac{1}{4} + \frac{1}{2} + 10 \right)$$

$$= 808 \times 10^{-6} \times \frac{202}{43}$$

$$= 808 \times 10^{-6} = 8.686 \text{ ms}$$

$\boxed{8.686 \text{ ms}}$

(c) 50 packets

$$\text{Packet size, } P = \frac{1}{50} \times \frac{2}{100} \text{ kB} + 100 \text{ B} = 16800 \text{ bits}$$

$$\text{Delivery time} = t_1 + t_2 + 50 \times t_3$$

$$= \frac{P}{R_1} + \frac{P}{R_2} + \frac{P}{R_3} \times 50$$

$$= 16800 \left( \frac{1}{4} + \frac{1}{2} + 50 \right) \times 10^{-6} \times \frac{1}{144}$$

$$= 16.8 \times \frac{203}{4} \times 10^{-3} = 8.52 \text{ ms}$$

(d) 100 packets

$$\text{Packet size, } P = \frac{1}{100} \times 100 \text{ kB} + 100 \text{ B} = 8800 \text{ bits}$$

$$\text{Delivery time} = \frac{P}{R_1} + \frac{P}{R_2} + 100 \times \frac{P}{R_3}$$

$$= 8800 \times 10^{-6} \times \frac{1}{144} \left( \frac{1}{4} + \frac{1}{2} + 100 \right)$$

$$= 88 \times 10^{-6} \times \frac{403}{4}$$

$$= 8.866 \text{ ms}$$

$\Rightarrow$  Lowest delivery time is for  $\boxed{50 \text{ packets}}$   $\rightarrow$  answer

Q3. & Q4.

3.

$$(a) \text{Propagation delay} = \frac{\text{length}}{\text{speed}} = \frac{10 \times 1000 \text{ m}}{2/3 \times 10^8 \text{ m/s}} = 50 \times 10^{-6} \text{ s} = 50 \text{ us}$$

(b) Maximum number of bits that can be sent between R<sub>1</sub> and R<sub>2</sub> in the time taken by the first bit to reach R<sub>2</sub> (or 50 us)

$$\begin{aligned} &= (\text{Bandwidth}) \times (\text{time taken by first bit}) \\ &= (8 \times 10^9 \text{ bits/s}) \times (50 \times 10^{-6} \text{ s}) \\ &= 5000 \times 10^3 \text{ bits} \\ &= 5 \times 10^6 \text{ bits} = 5 \text{ Mb} \end{aligned}$$

$$\begin{aligned} (c) \text{Bit width} &= \frac{(\text{length})}{(\text{maximum number of bits})} \\ &= \frac{10 \times 1000 \text{ m}}{5 \times 10^6 \text{ bits}} = 2 \times 10^{-3} \text{ m/bit} \\ &= 2 \text{ mm/bit} \end{aligned}$$

4.

$$\text{RTT} = 10 \text{ ms} \quad (\text{round trip})$$

(a) Non-persistent

Let the file transmission time be 't' seconds for each KB of data.

$$\text{Page load time} = [\text{Request time}] + [\text{Response time}]$$

$$= \underbrace{[\# \text{requests}] \times (\text{RTT})}_{\text{non-persistence}} + \underbrace{[\# \text{requests}] \times (\text{RTT})}_{\text{time to send each request}} + \underbrace{[t + 10 \times 100]}_{\substack{\text{Response time for webpage} \\ \text{Response time for all objects}}} \times t$$

$$= ((\alpha+1) \times 50 \times 10^{-3} + (\alpha+1) \times 10 \times 10^{-3} + (\alpha + 100t))$$

$$= \frac{11}{100} + \frac{11}{100} + 1001t.$$

$$= (0.22 + 1001t)$$

↳ (# requests  
= 1 webpage  
+ 10 objects)  
= 11

### (b) Persistent

Again, let file transmission rate =  $\alpha/10^3$  t seconds for 1 kB

$$\text{Page load time} = [\text{Request time}] + [\text{Response time}]$$

$$= [(\# \text{ requests}) \times (\text{RTT}) + \text{RTT}] + [\alpha + 10 \times 100t]$$

$\downarrow$   
Persistent connection  
requiring connection  
to be set only once

$$= [12 \times 10 \times 10^{-3}] + [100t]$$

$$= (0.12 + 100t)$$

### (c)

Let file transmission time =  $\alpha t$  seconds for 1 kB of data

$$\text{Page load time} = [\text{Initial connection time}] + [\text{Request + Response time for web-page}]$$

+ [Request + Response time for object]

$$= [\text{RTT}] + [\text{RTT} + \cancel{\text{RTT}}] + [\text{RTT} + 10 \times 100 \times \cancel{10^{-3}}]$$

$$= 3 \times \text{RTT} + \alpha t + 1000t$$

$$= 3 \times 10 \times 10^{-3} + 1001t$$

$$= (0.03 + 1001t)$$

---

---

## Part III: Network Tools

Q5.

---

(a)

1. **ICMPv6** - It is the implementation of the Internet Control Manager Protocol for IPv6. It performs several functions when data is transmitted from one end to the other including router discovery, error reporting, diagnostics, etc. It is used by routers, hosts, and other intermediary devices to other devices.

Associated RFC Number: **4443**

2. **mDNS** - mDNS or Multicast DNS is an alternative to the DNS protocol for resolving hostnames to IP addresses in smaller networks. It uses the same interfaces, formats, and semantics as DNS.

Associated RFC Number: **6762**

3. **OCSP** - OCSP or Online Certificate Status Protocol is used to determine the revocation status of an identified X.509 digital certificate in real time. It is useful in time-sensitive requests such as bank transactions.

Associated RFC Number: **6960**

4. **SSDP** - SSDP or Simple Service Discovery Protocol is used in small networks such as home networks to advertise and discover network services without configuration mechanisms such as DNS. It is a simple text-based protocol that uses UDP as the transport protocol.

Associated RFC Number: **NO RFC**

5. **TLSv1.2** - TLS or Transport Layer Security is a protocol used to provide secure communications across a network. It is majorly used to secure HTTPS, but also has applications in services such as email, voice over IP, and messaging.

Associated RFC Number: **5246**

---

(b)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000000	192.168.81.129	192.168.81.2	DNS	71	Standard query 0xF348 A yahoo.com
2	0.000182561	192.168.81.129	192.168.81.2	DNS	71	Standard query 0xba56 AAAA yahoo.com
3	0.005065982	192.168.81.2	192.168.81.129	DNS	449	Standard query response 0xf348 A yahoo.com A 74.6.231.20 A 74.6.231.2
4	0.005066461	192.168.81.2	192.168.81.129	DNS	521	Standard query response 0xba56 AAAA yahoo.com AAAA 2001:4998:44:3507:
5	0.005989090	192.168.81.129	74.6.231.20	ICMP	100	Echo (ping) request id=0xc3d3, seq=1/256, ttl=64 (reply in 6)
6	0.360639046	74.6.231.20	192.168.81.129	ICMP	100	Echo (ping) reply id=0xc3d3, seq=1/256, ttl=128 (request in 5)
7	0.361472761	192.168.81.129	192.168.81.2	DNS	86	Standard query 0x18b2 PTR 20.231.6.74.in-addr.arpa
8	0.454753312	192.168.81.2	192.168.81.129	DNS	428	Standard query response 0x18b2 PTR 20.231.6.74.in-addr.arpa PTR media

Let us estimate the RTT for the connection between request and reply packets with numbers **5** and **6** respectively in the above screenshot.

$$\begin{aligned}RTT &= (\text{Time for 6}) - (\text{Time for 5}) = (0.361 \text{ seconds}) - (0.006 \text{ seconds}) \\&= 0.355 \text{ seconds} \\&= 355 \text{ milliseconds}\end{aligned}$$

---

(c)

No cookies are found when we only visit the webpage <https://ims.iitgn.ac.in/>.

The screenshot shows the Network tab of the Chrome DevTools interface. The main pane displays a screenshot of the IMS Institute Management System homepage. The sidebar on the right is titled 'Application' and includes sections for Manifest, Service Workers, Storage, Cache, and Background Services. Under 'Storage', the 'Cookies' section is expanded, showing a table with one entry for the domain https://ims.iitgn.ac.in. The table has columns for Name, Value, Domain, P..., E..., S..., H..., S..., S..., Parti..., and F. A message at the bottom right of the sidebar says 'Select a cookie to preview its value'.

The following cookie "RequestToken" is found when I logged in to the IMS Student Portal.

The screenshot shows the Chrome DevTools Application tab. On the left, there's a sidebar with sections for Application (Manifest, Service Workers, Storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies), Cache (Cache Storage, Back/forward cache), and Background Services (Background Fetch, Background Sync, Notifications, Payment Handler, Periodic Background, Push Messaging, Reporting API). The main area is titled 'Cookies' and shows a single entry for 'https://ims.iitgn.ac.in'. The cookie details are as follows:

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	SameParty	Partition Key	Priority
RequestToken	bp11q45qdwgw225raxjyanhd	ims.iitgn.ac.in	/	Session	36	✓		Lax			Medium

Below the table, it says 'Cookie Value' and shows 'bp11q45qdwgw225raxjyanhd'. There are also checkboxes for 'Show URL decoded' and 'Show URL encoded'.

The following are the characteristics of the “RequestToken” cookie:

- The cookie is created as soon as the person logs in and expires when we log out. It is a **Session** cookie.
  - The cookie's path is ‘\’ which makes it available to the entire web pages.
  - The size of the cookie is 36 bytes.
  - The cookie is HttpOnly, which does not allow client-side scripts to access the cookie's data.
  - The SameSite attribute does not allow the browser to send cookies' data to cross-site requests. The lax value of the attribute allows a balance of security and usability.
- 
-