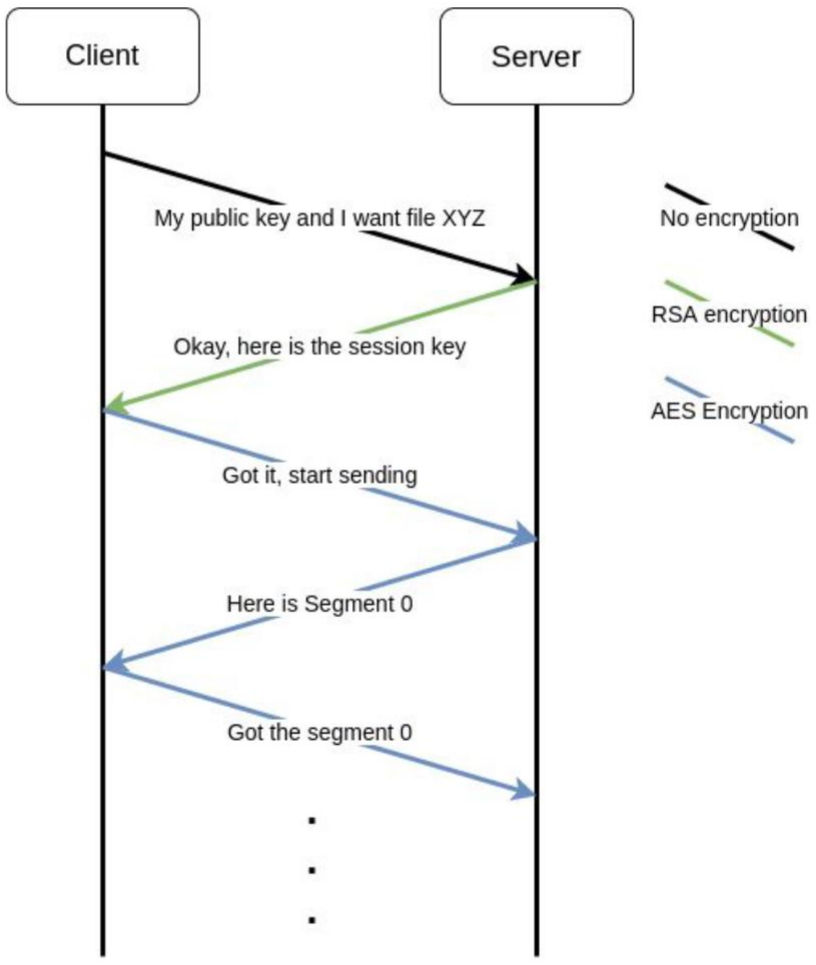
Reliable Data Transfer Protocol using UDP

In this project, you are required to **implement a server** that provides encrypted, reliable transport protocol (similar to TCP) **using UDP on top of an unreliable channel**. You will use an asymmetrical encryption scheme(RSA-1024) to exchange private session keys. After key exchange is complete, symmetrical and more efficient AES-256 will be used. AES is a symmetrical encryption scheme, where both the sender and receiver have **the same key**. In contrast, RSA is an asymmetric encryption scheme, where **encryption and decryption keys are different.** This makes RSA suitable for exchanging session keys on unsecure transmission channels, in which malicious third-parties can inspect/imitate packets. Thus, the client code initially generates a private and public key pair and only shares its public key with the server. The server encrypts the session key(which is generated by the server on the fly) using RSA. Since only the client has the private key, it can safely decode the session key from the server and start using computationally less expensive AES to encrypt and decrypt messages. You will be given a **client code**. You will develop your **server code locally** using this client code.The server must be implemented considering these items:

- All of the packets sent can be lost in the unreliable channel. This includes Handshake packets, ACK packets and Data packets. Packet loss is controlled by changing parameters(errRate) of the **unreliableSend()** function. Packet loss will take values: {1%, 5%, 10%, 20%, 40%}
- Client initiates handshake by sending its public RSA key to the server. Then the server encrypts the session key using this public RSA key. After this step both parties know the session key.
- After exchanging the session key for the AES-256, encryption for all packets will be done using this session key.
- The server will detect timed out packets just like reliable data transfer protocol (rdt3) explained in the class' slides(chapter 3, pages 39, 48-49) .
- The client and the server will use the **go-back-N** scheme to pipeline multiple segments. You must analyze the performance(number of packets, throughput) by using different N values {1, 5, 20, 50, 100}.

Part 1 - Implementing Triple Handshake

In this part, the client should establish the end-to-end connection by sending its public key and wanted file's name to the server. In this homework, file is a simple text file having a word in each line which will be shared along with the client code. The name of the file is arbitrary and is known by the client from the start. When the server receives the client's initiation message, it checks whether the filename is correct. If it is wrong, then server timeouts. If the filename matches, the server embeds the session key into the handshake packet that will be sent to the client. While sending the session key, the server encrypts it using the public key provided by the client via RSA encryption. When the client gets the encrypted packet from the server, it decrypts it using its private key(RSA1024 private key) to acquire the session key. Then, it encrypts another ACK message using the session key generated by the server and sends it to the server. After this step the triple handshake is complete and both the server and the client know the session key that is going to be used when encrypting further packages.



Part 2 - Implementing RDT protocol

In this part, the server starts sending segments of the text file. The segments can get lost in the way. Your server should notice any lost segments when the ACK for that segment timeouts. The timeout value is 0.0001 seconds as in the sample client code. Thus the server needs to properly handle packet timeouts. Finally you must use the Go-back-N method to pipeline your reliable transport protocol.

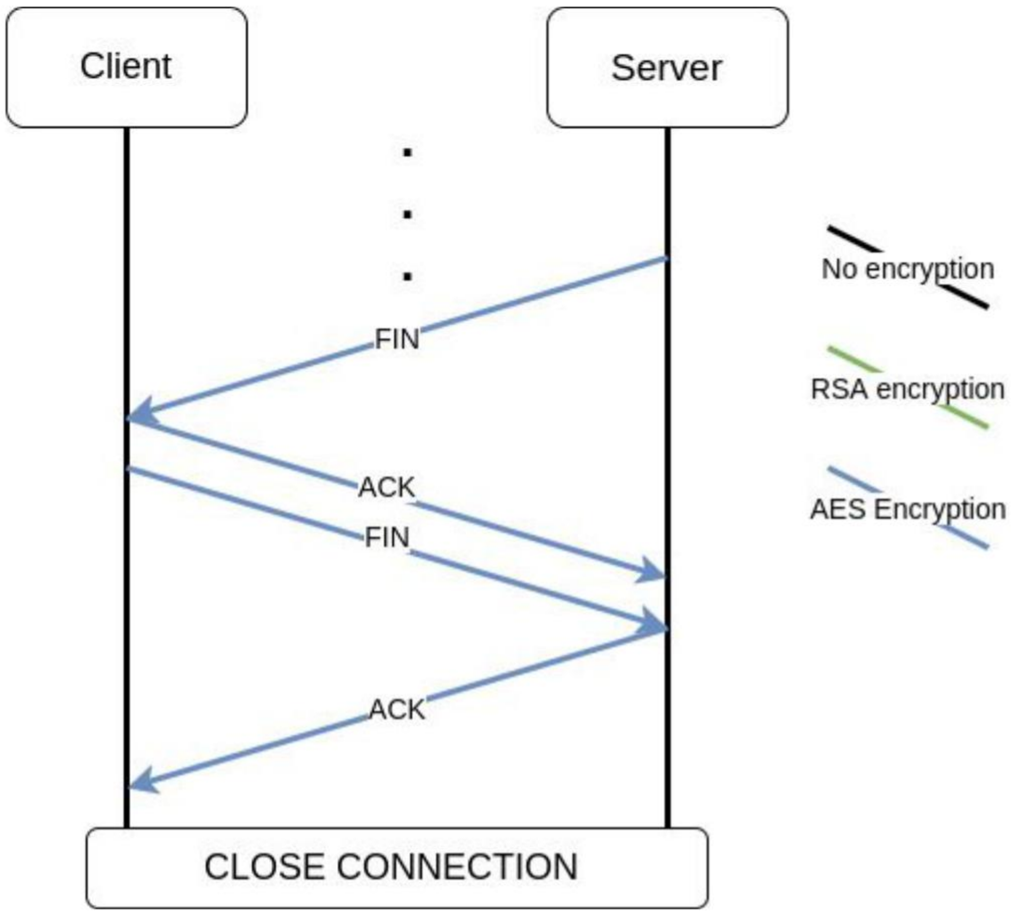


Figure 2: Ending Sequence

When the server receives the ACK for the last segment, it sends the FIN packet to the client. After the client receives the FIN packet it sends ACK(with sequence number = file's sequence number+1) back to the server. Following this the client increments the sequence number and sends its own FIN packet and partially gets closed(if server times out connection gets closed). After receiving the FIN packet, the server sends the ACK packet and closes the connection.

Appendix A: Packet types

First byte of the packet determines the type of the packet. All of the packets are encoded using UTF-8. Payload lengths are given in terms of bytes and max payload length is 255 bytes. Similar to this, sequence numbers are counted with 1 bytes.

Packet type	**Meaning**
0	Handshake
1	ACK
2	DATA
3	FIN

Handshake Packet (Client Side)

This packet is not encrypted. It has the public RSA key of the client which server will use to encrypt the session key.

Bytes	**Content**
0	Packet type = 0
1	Payload Length (Filename + RSA1024 Key)
2-...	Filename + RSA1024 Key

Handshake Packet (Server Side)

This packet is encrypted using the public key provided by the client. The encryption scheme is RSA1024.

Bytes	**Content**
0	Packet type = 0
1	Payload Length (AES256 Key)
2-...	RSA 1024 Encrypted AES256 Key

ACK Packet

This packet is encrypted using the private session key provided by the server. The encryption scheme is AES256.

Bytes	Content
0	Packet type = 1
1	Sequence Number

Data Packet

This packet is encrypted using the private session key provided by the server. The encryption scheme is AES256. It has a word in its payload.

Bytes	Content
0	Packet type = 2
1	Payload Length (Data)
2	Sequence Number
3-...	Data

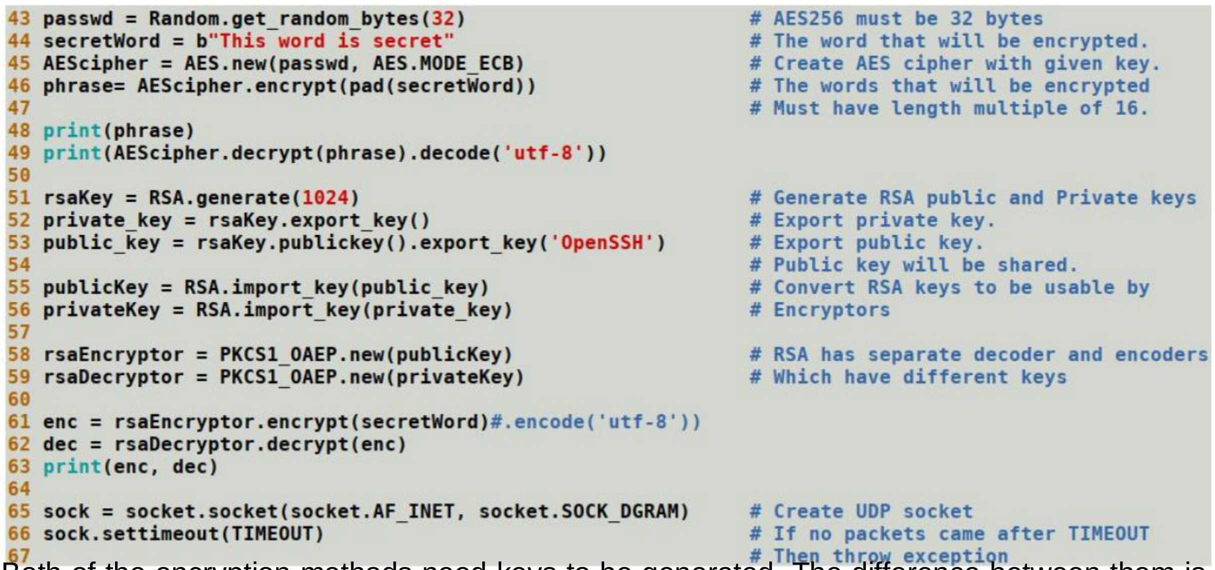
FIN Packet

This packet is encrypted using the private session key provided by the server. The encryption scheme is AES256. This packet is sent by the server to indicate data transfer is complete.

Bytes	Content
0	Packet type = 3
1	Sequence Number

Appendix B: Encryption

In this part RSA/AES encryption and decryption examples will be given. The code piece below is taken from the sample client that will be shared with the project. In this project the **pycryptodome**library is used for encryption.



```
passwd = Random.get_random_bytes(32)                          # AES256 must be 32 bytes
secretWord = b"This word is secret"                           # The word that will be encrypted.
AEScipher = AES.new(passwd, AES.MODE_ECB)                     # Create AES cipher with given key.
phrase= AEScipher.encrypt(pad(secretWord))                    # The words that will be encrypted
                                                              # Must have length multiple of 16.
print(phrase)
print(AEScipher.decrypt(phrase).decode('utf-8'))

rsaKey = RSA.generate(1024)                                   # Generate RSA public and Private keys
private_key = rsaKey.export_key()                             # Export private key.
public_key = rsaKey.publickey().export_key('OpenSSH')         # Export public key.
                                                              # Public key will be shared.
publicKey = RSA.import_key(public_key)                        # Convert RSA keys to be usable by
privateKey = RSA.import_key(private_key)                      # Encryptors

rsaEncryptor = PKCS1_OAEP.new(publicKey)                      # RSA has separate decoder and encoders
rsaDecryptor = PKCS1_OAEP.new(privateKey)                     # Which have different keys

enc = rsaEncryptor.encrypt(secretWord)#.encode('utf-8'))
dec = rsaDecryptor.decrypt(enc)
print(enc, dec)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)       # Create UDP socket
sock.settimeout(TIMEOUT)                                      # If no packets came after TIMEOUT
                                                              # Then throw exception
```

Both of the encryption methods need keys to be generated. The difference between them is where AES-256 one key and RSA-1024 needs two keys. After creating the cipher classes, encrypt() and decrypt() functions simply will do encryption. AES encryption needs blocks of 16-bytes to encrypt. Thus, words must be padded to multiples of 16 and unpadded when decrypted. These functions are also present in the **client.py**.

Submission Guidelines

- You must present parameter analysis on your report. Discuss the **optimal N** which gives **best throughput**for each packet loss scenario.
- Your code must work on Python 3.7 or higher.
- You can not use TCP sockets and **you must use the unreliableSend()** function provided in the project files.