COSC 3371 Cybersecurity — Assignment 2

Encrypted File Transfer via Sockets

Course	Points	Language	Due Date
COSC 3371 Cybersecurity	100 pts	Python 3.10+ | cryptography library required	March 11, 2026

1. Overview:

Secure communication relies on two main components: (1) asymmetric (public-key) cryptography for safely exchanging a secret, and (2) symmetric ciphers to encrypt large amounts of data efficiently. In this assignment, you will implement both within a client-server file transfer program.

2. Learning Objectives

- Understand TCP client-server socket programming in Python.
- Use RSA public-key encryption to share a symmetric key securely.
- Use AES in CFB mode to encrypt a file for confidential transfer.
- Observe the difference between encrypted and plain text traffic using Wireshark.

3. Prerequisites

Before starting, ensure the following are in place:

Software	Knowledge
Python 3.10+ (pip install cryptography)	TCP sockets and the OSI model
Wireshark (wireshark.org/download.html)	RSA public-key encryption
PyCharm or VS Code IDE	Block ciphers and AES
	Modular arithmetic basics

4. Background

In this assignment, you will implement a hybrid encryption scheme that mimics real-world protocols like TLS. RSA is used once to securely exchange a random AES key; AES then encrypts the file content quickly.

Protocol Flow (Tasks 1 – 5)

Protocol Flow

1. Client connects to the server (port 6000).
2. Server sends its RSA public key.
3. Client generates a random 256-bit AES key.
4. Client encrypts the AES key with the server's RSA public key and sends it.
5. Client sends the filename (4-byte length prefix + filename bytes).
6. Client encrypts the file content with AES-CFB and sends it in 4096-byte chunks.
7. Server decrypts the AES key using its RSA private key, then decrypts the file.
8. Connection closes.

Cryptography Used

Algorithm	Purpose	Key Fact
RSA-2048	Key exchange	It can only encrypt ~190 bytes; therefore, it is used only for the AES key.
AES-256-CFB	File encryption	Stream-like mode; a unique random IV is prepended to the ciphertext.

5. Files Provided

To make the work easier for students who are not familiar with client-server computing, we are providing the following program skeleton.

- server.py — skeleton for the server (fill in the ## TODO sections)
- client.py — skeleton for the client (fill in the ## TODO sections)

6. Tasks

Task 1 — Complete server.py

Fill in every **## TODO** block in server.py. The comments explain what each block should do. Do **NOT** change the existing helper functions or the surrounding socket logic.

Checklist:

- Generate the RSA key pair at startup.
- Serialize and send the public key to the connecting client.
- Receive and RSA-decrypt the AES symmetric key.
- Receive the length-prefixed filename.
- Receive the encrypted file content in 4096-byte chunks.
- AES-decrypt the content and write it to ./uploads/.

Task 2 — Complete client.py

Fill in every **## TODO** block in client.py.

- Receive and deserialize the server's public key.
- Generate a 32-byte AES key, RSA-encrypt it, and send it.
- AES-encrypt the file content.
- Send the encrypted content in 4096-byte chunks.

Task 3 — Run and Observe the Transfer

1. Create MyFile.txt. Line 1: your full name. Line 2: your student ID.
2. Start Wireshark. Capture on the Loopback interface (Windows: Loopback; macOS/Linux: lo0).

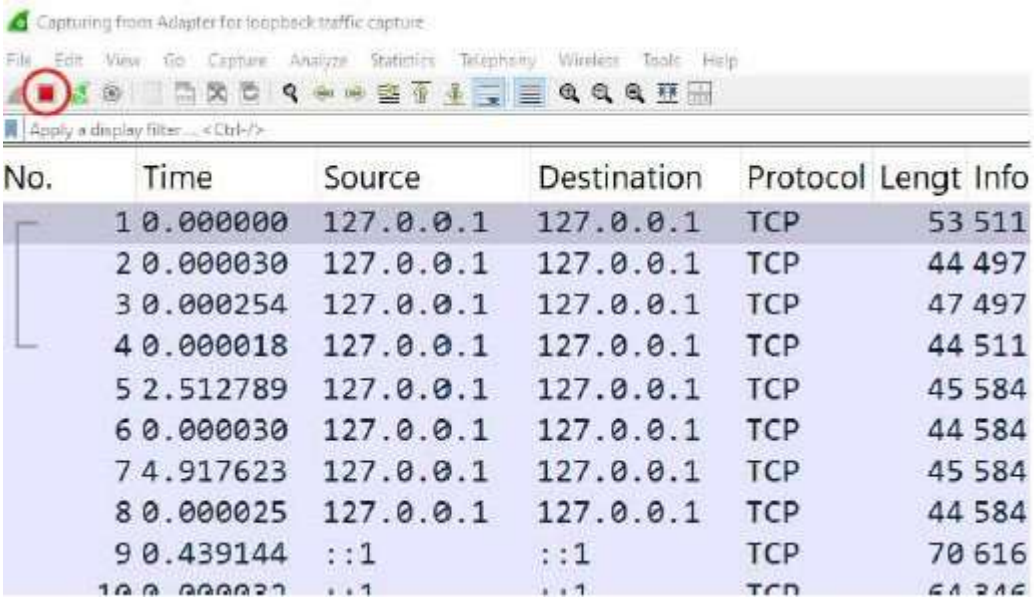
3. In one terminal run: python server.py

```
[+] RSA-2048 key pair generated.
[+] Server listening on 0.0.0.0:6000 ...
```

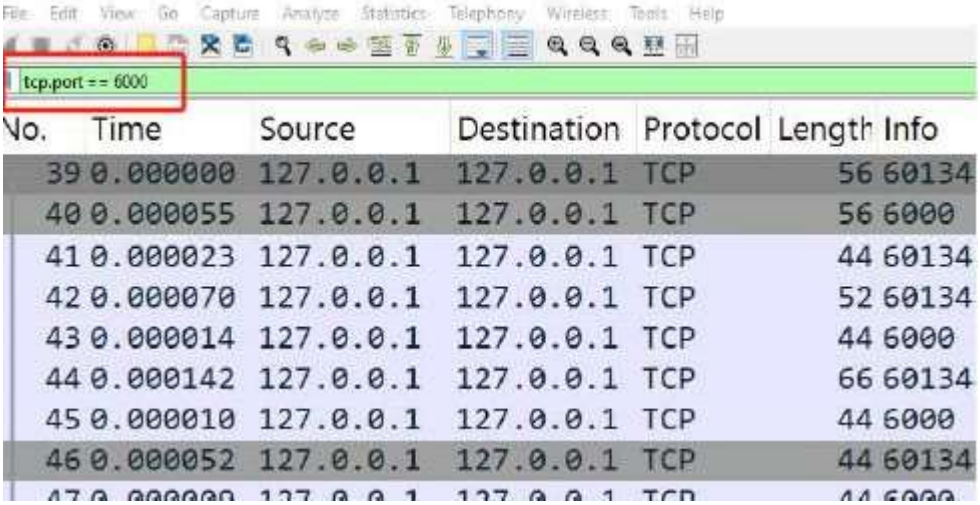
4. In another terminal, run: Python client.py

```
Enter the file name to send: myfile.txt
[+] Connected to 127.0.0.1:6000
[+] Received and deserialized server public key.
[+] Sent RSA-encrypted AES key to server.
[+] Sent filename: 'myfile.txt'
[+] File 'myfile.txt' sent successfully (29 bytes plain → 45 bytes encrypted).
```

5. After the transfer is completed, stop the Wireshark capture.



6. Filter with tcp.port == 6000. Examine every packet that has a TCP payload.

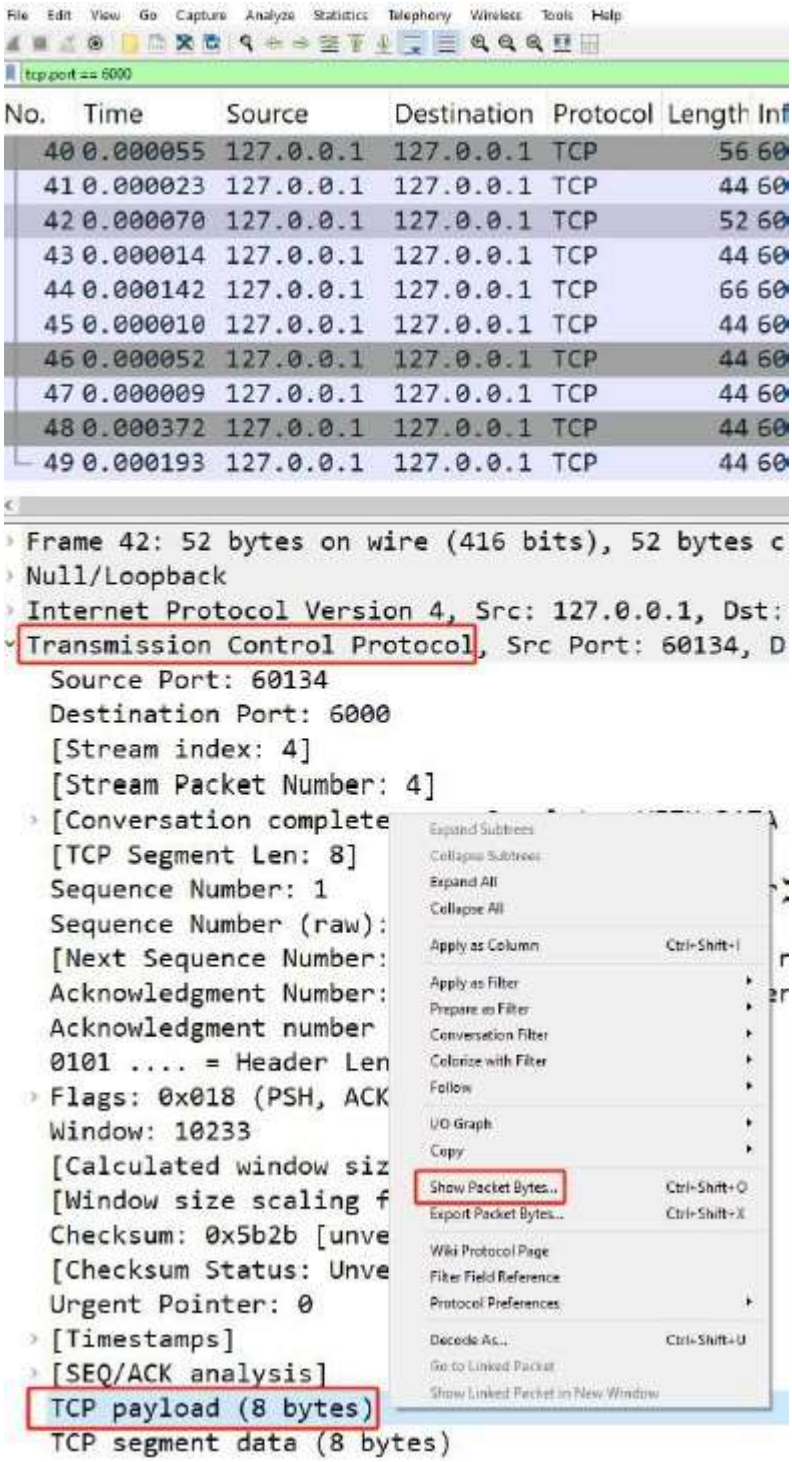


Task 4 — Wireshark Screenshots

Right-click the TCP payload field of each relevant packet and select **Show Packet Bytes**. Take a screenshot of every payload window you observe. Include a brief caption for each:

Caption examples:

- 'RSA public key' — the server's PEM-encoded public key
- 'Encrypted AES key' — 256 bytes of RSA-encrypted binary data
- 'Encrypted file content' — AES ciphertext (random-looking binary)



7. Deliverables (100 pts total)

Item	Points
client.py — all TODO blocks correctly implemented	40 pts
server.py — all TODO blocks correctly implemented	40 pts

Successful end-to-end file transfer (demonstrated terminal output)	10 pts
Wireshark screenshots with captions (all payload types)	10 pts

8. Hints & Quick Reference

Installing the cryptography library

```
pip install cryptography
```

Byte-Size Quick Guide

Field	Size	Notes
AES-256 key	32 bytes	os.urandom(32) or SHA-256 of DH shared secret
AES IV	16 bytes	Prepended to ciphertext automatically
RSA-2048 ciphertext	256 bytes	Encrypted AES key
Filename length prefix	4 bytes	Big-endian unsigned int
File content length prefix	8 bytes	Big-endian unsigned long long

9. Submission

Submit via the Canvas course portal — one ZIP file named:

`assignment2_<YourLastName_YourID>.zip`

Academic Integrity: All code must be your own. You may discuss concepts, but you must not share or copy code. Violations will result in a grade of zero.

Good luck — and remember, every secure connection you make on the web uses concepts you are implementing here!