**Internship Progress Report**

**Intern Details**

* **Name**: Swarn Ranjan
* **Internship Duration**: 06th May 2025 – 30th June 2025
* **Institution**: Kalinga Institute of Industrial Technology
* **Mentor**: Dr. Tarun Yadav, Scientist-F, SAG, DRDO, New Delhi
* **Reporting Period**: Week 4 (21/05/2025 – 04/06/2025)

**1. Topics Covered**

During this period, I modularized and enhanced the secure chat application by implementing a structured communication protocol and separating cryptographic logic into a utility module. Key progress includes:

* Creation of crypto\_utils.py for reusable encryption/decryption logic using AES-CBC
* Implemented AES-based secure text and file transfer
* Designed a custom protocol using message-type headers (MSG , FILE) and length-prefixed fields
* Developed threaded server-client architecture for real-time communication
* Added direct client-to-client messaging via centralized relay server
* Enabled verbose logging for encryption and decryption processes (server-side)
* Added file encryption/decryption support for safe file transfers between users
* Performed full system tests with multiple users sending messages and files

**2. Resources Utilized**

* PyCryptodome documentation and FIPS-197 standard (AES)
* Python socket and threading documentation
* Community forums (Stack Overflow, GitHub issues)
* Local tests, simulation scripts, and encrypted file outputs

**3. Key Learnings**

* Modular programming: Decoupled cryptographic code into crypto\_utils.py, improving clarity and reuse.
* Structured message protocol: Created a predictable communication pattern with headers (MSG , FILE) and padding.
* File I/O with encryption: Learned to securely encrypt and decrypt large binary files block by block.
* AES internals: Understood the importance of IVs, padding, and ciphertext formatting.
* Threaded client handling: Enhanced concurrency on the server using daemon threads.
* Logging and debugging: Added verbose logs to understand encryption flows in real time.

**4. Project Status**

The secure messaging system now includes:

* End-to-end AES encryption for messages and files
* Client identification using fixed-length username fields
* File sending and receiving via encrypted binary data
* Interactive text-based client interface
* Logging for encryption, decryption, file handling, and errors
* Modular and maintainable code architecture (server.py, client.py, crypto\_utils.py)

**5. Code Snippet**

* AES Encryption:

def encrypt\_message(message, verbose=False):

    iv = get\_random\_bytes(16)

    cipher = AES.new(KEY, AES.MODE\_CBC, iv)

    ct\_bytes = cipher.encrypt(pad(message.encode(), AES.block\_size))

    iv\_enc = base64.b64encode(iv).decode()

    ct\_enc = base64.b64encode(ct\_bytes).decode()

    encrypted\_text = iv\_enc + ":" + ct\_enc

    return encrypted\_text

* AES Decryption:

def decrypt\_message(cipher\_text, verbose=False):

    iv\_str, ct\_str = cipher\_text.split(":", 1)

    iv = base64.b64decode(iv\_str)

    ct = base64.b64decode(ct\_str)

    cipher = AES.new(KEY, AES.MODE\_CBC, iv)

    pt = unpad(cipher.decrypt(ct), AES.block\_size)

    decrypted\_text = pt.decode()

    return decrypted\_text

* File Encryption/Decryption Snippets:

def encrypt\_file(input\_path, output\_path, verbose=False):

    iv = get\_random\_bytes(16)

    cipher = AES.new(KEY, AES.MODE\_CBC, iv)

    with open(input\_path, 'rb') as f:

        plaintext = f.read()

    padded\_data = pad(plaintext, AES.block\_size)

    ciphertext = cipher.encrypt(padded\_data)

    with open(output\_path, 'wb') as f:

        f.write(iv + ciphertext)

* Message Handling in Server

if msg\_type == b"MSG ":

length = int(conn.recv(10).decode())

encrypted = conn.recv(length).decode()

decrypted = decrypt\_message(encrypted, verbose=True)

print(f"[SERVER] Message from {username} to {recipient}: {decrypted}")

* File Sending in Client

sock.send(b"FILE")

    sock.send(recipient.encode().ljust(20))

    sock.send(str(len(filename)).zfill(3).encode())

    sock.send(filename.encode())

    sock.send(str(filesize).zfill(12).encode())

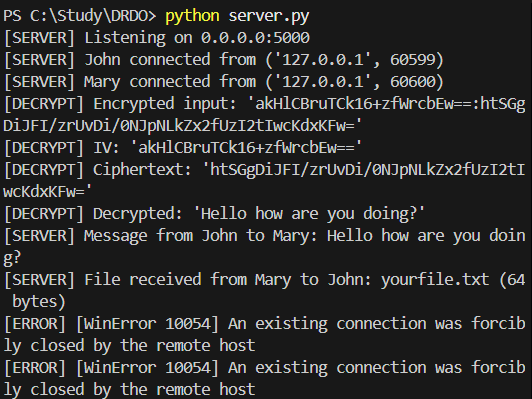
    with open(enc\_file, 'rb') as f:

        while chunk := f.read(4096):

            sock.send(chunk)

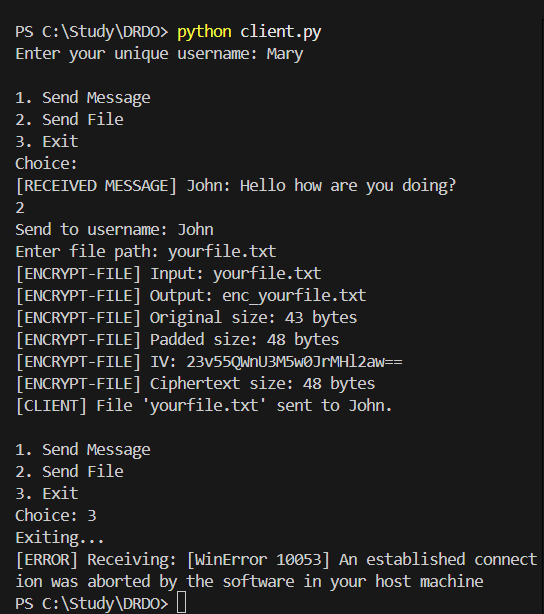
**6. Terminal Outputs**

* Server terminal output

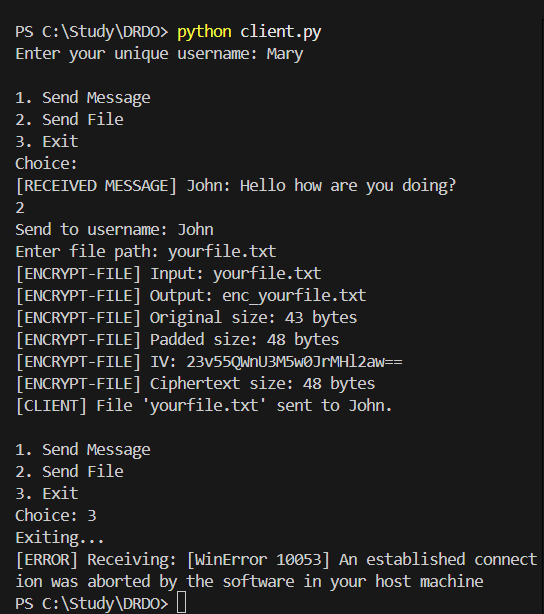


* Client terminal output

Client 1:



Client 2:



**7. Upcoming Objectives (Weeks 5–6)**

* Add message/file integrity checks using HMAC or SHA-256
* Build a GUI frontend using Tkinter or PyQt
* Implement logging to file instead of console only
* Add file type and size validations
* Evaluate scalability and error scenarios under stress
* Document the project structure, flows, and encryption policy

(Swarn Ranjan)