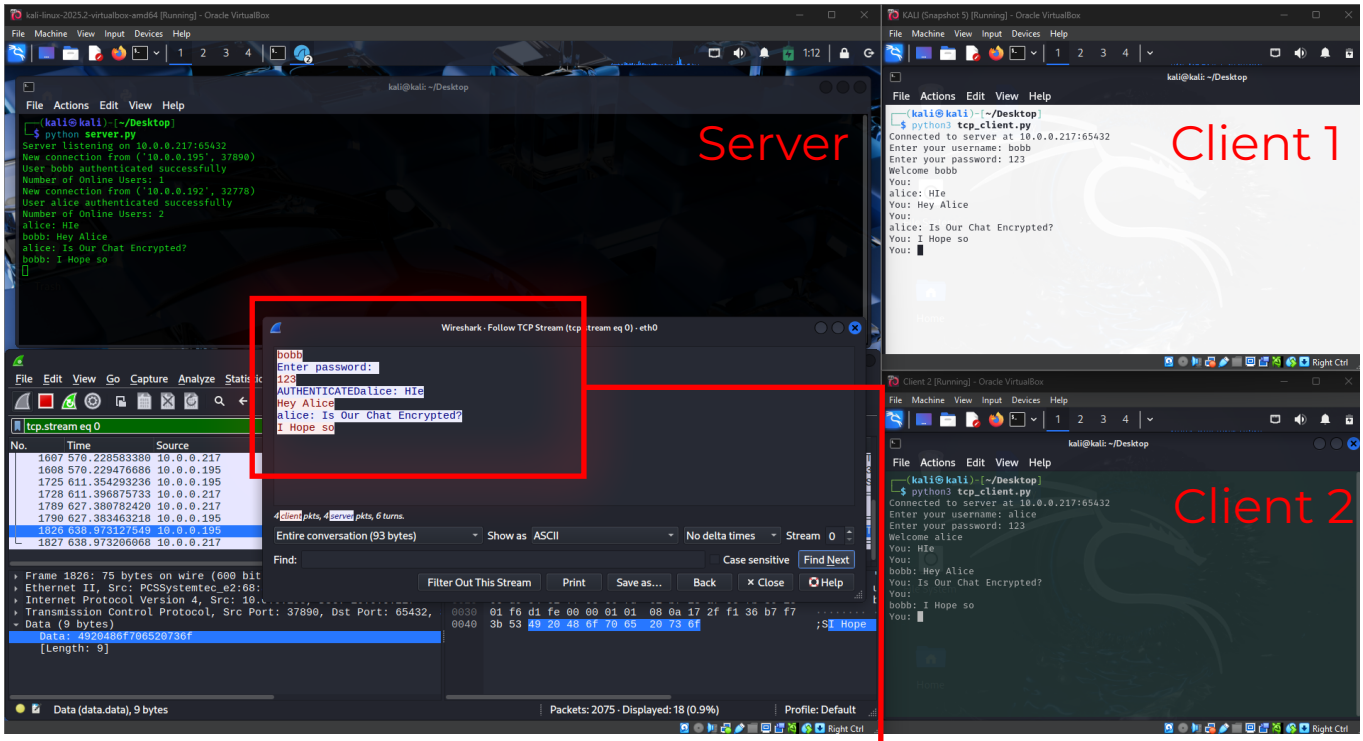
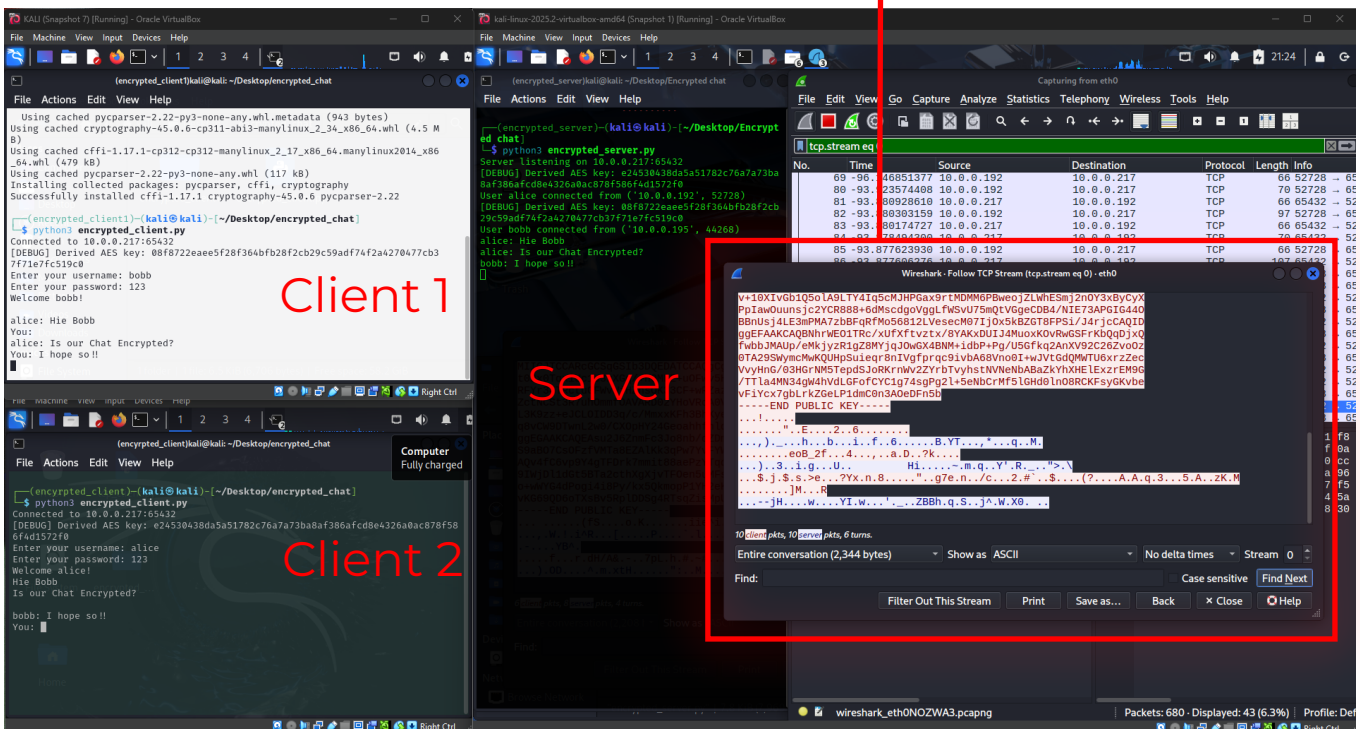
BEFORE ENCRYPTION



AFTER ENCRYPTION



Steps

Phase 1:
Cryptographic Setup

1 Import Required Libraries

```python
import socket
import threading
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives.asymmetric import dh
from cryptography.hazmat.primitives.serialization import *
import os
```

```
(encrypted_server)kali@kali: ~/Desktop/Encrypted chat
File  Actions  Edit  View  Help
└─$ python3 encrypted_server.py
Traceback (most recent call last):
  File "/home/kali/Desktop/Encrypted chat/encrypted_server.py", line 4, in <m
odule>
    from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, mo
des
ModuleNotFoundError: No module named 'cryptography'

┌──(encrypted_server)─(kali㉿kali)-[~/Desktop/Encrypted chat]
└─$ pip install cryptography
Collecting cryptography
  Downloading cryptography-45.0.6-cp311-abi3-manylinux_2_34_x86_64.whl.metada
ta (5.7 kB)
Collecting cffi ≥ 1.14 (from cryptography)
  Downloading cffi-1.17.1-cp313-cp313-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (36 kB)
Collecting pycparser (from cffi ≥ 1.14 → cryptography)
  Downloading pycparser-2.22-py3-none-any.whl.metadata (943 bytes)
Downloading cryptography-45.0.6-cp311-abi3-manylinux_2_34_x86_64.whl (4.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.5/4.5 MB 27.1 MB/s eta 0:00:00
Downloading cffi-1.17.1-cp313-cp313-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (467 kB)

Downloading pycparser-2.22-py3-none-any.whl (117 kB)
Installed packages: pycparser, cffi, cryptography
Successfully installed cffi-1.17.1 cryptography-45.0.6 pycparser-2.22

┌──(encrypted_server)─(kali㉿kali)-[~/Desktop/Encrypted chat]
└─$
```

2 Generate Diffie-Hellman Parameters

```python
class ChatServer:
    def __init__(self):
        self.online_users = {}  # {conn: username}
        self.users = {"bobb": "123", "alice": "123"}
        self.lock = threading.Lock()

        # Generate DH parameters (shared between server and clients)
        self.dh_parameters = dh.generate_parameters(generator=2, key_size=2048)
```

3 Add Key Derivation Function

```python
def derive_aes_key(self, shared_secret):
    # Derive a 32-byte (256-bit) AES key
    return HKDF(
        algorithm=hashes.SHA256(),
        length=32,
        salt=None,
        info=b'chat-app-key',
    ).derive(shared_secret)
```

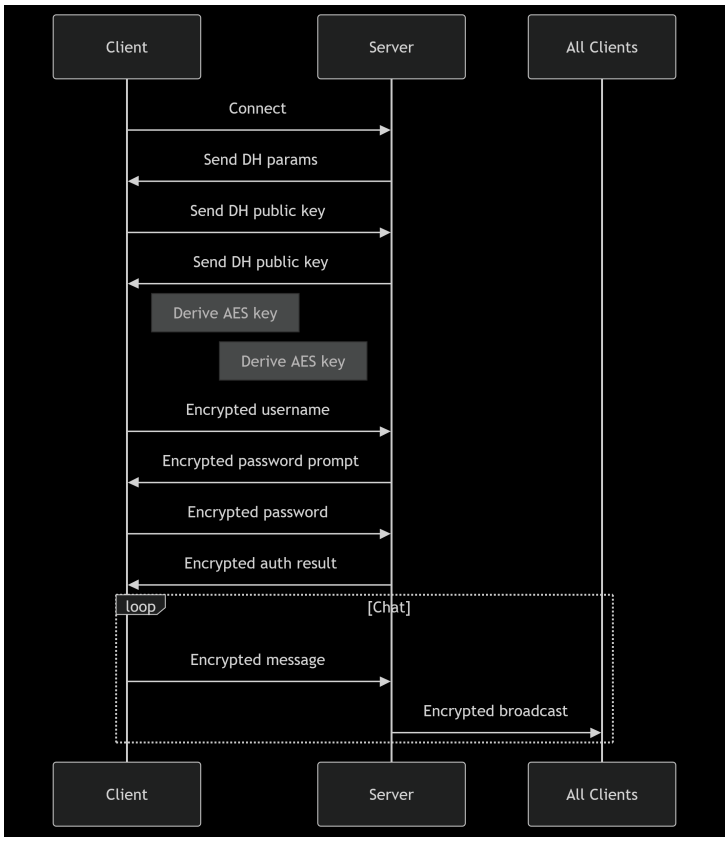
Phase 2:
Secure Key Exchange

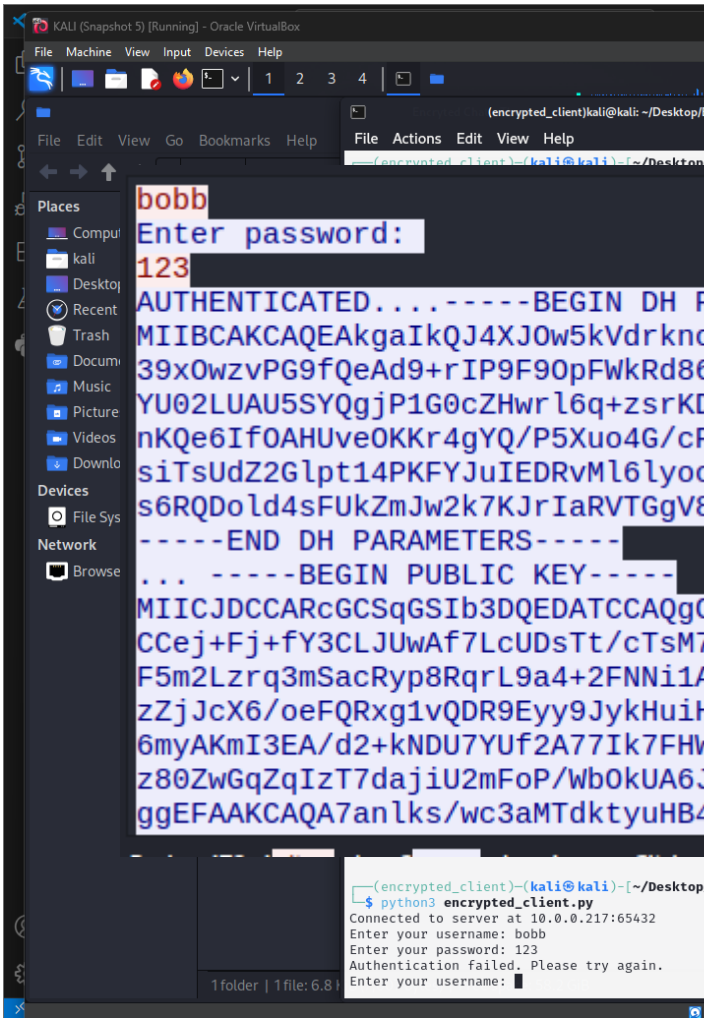
```python
if password == self.users[username]:
    # Perform Diffie-Hellman key exchange
    # Step 1: Server sends DH parameters to client
    conn.sendall(
        self.dh_parameters.parameter_bytes(
            Encoding.PEM,
            ParameterFormat.PKCS3
        )
    )

    # Step 2: Server generates its private key
    server_private_key = self.dh_parameters.generate_private_key()

    # Step 3: Server sends its public key to client
    server_public_key = server_private_key.public_key()
    conn.sendall(
        server_public_key.public_bytes(
            Encoding.PEM,
            PublicFormat.SubjectPublicKeyInfo
        )
    )

    # Step 4: Receive client's public key
    client_public_key_bytes = conn.recv(2048)
    client_public_key = load_pem_public_key(client_public_key_bytes)

    # Step 5: Generate shared secret
    shared_secret = server_private_key.exchange(client_public_key)

    # Step 6: Derive AES key
    aes_key = self.derive_aes_key(shared_secret)
```

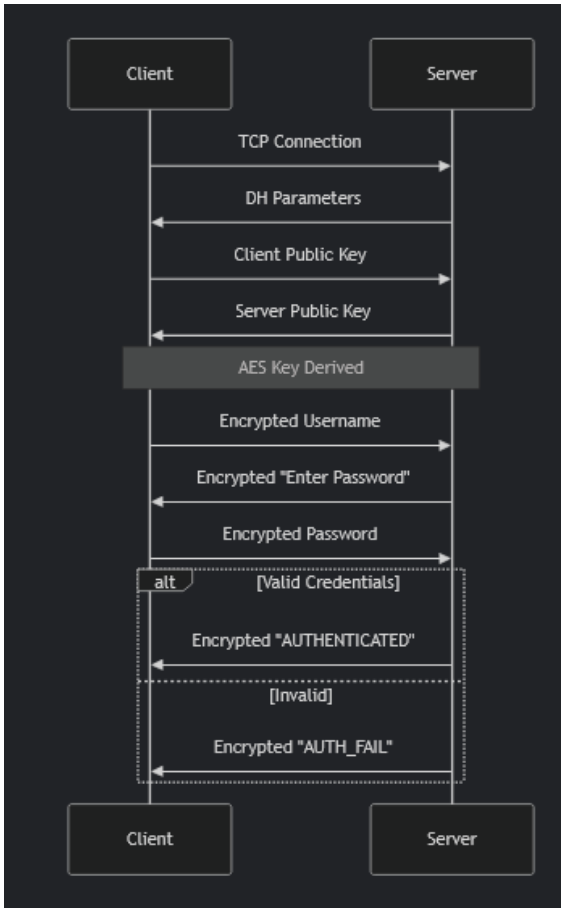
Phase 3
Add Encryption and Decryption Function

```python
def encrypt_message(self, key, message):
    # Generate a random 12-byte nonce for GCM
    nonce = os.urandom(12)

    # Encrypt the message
    cipher = Cipher(
        algorithms.AES(key),
        modes.GCM(nonce),
    )
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(message.encode()) + encryptor.finalize()

    # Return nonce + ciphertext + tag
    return nonce + ciphertext + encryptor.tag
```

```python
def decrypt_message(self, key, encrypted_data):
    # Split the data into nonce, ciphertext, and tag
    nonce = encrypted_data[:12]
    ciphertext = encrypted_data[12:-16]
    tag = encrypted_data[-16:]

    # Decrypt the message
    cipher = Cipher(
        algorithms.AES(key),
        modes.GCM(nonce, tag),
    )
    decryptor = cipher.decryptor()
    plaintext = decryptor.update(ciphertext) + decryptor.finalize()

    return plaintext.decode()
```


Client
Server
All Clients
Connect
Send DH params
Send DH public key
Send DH public key
Derive AES key
Derive AES key
Encrypted username
Encrypted password prompt
Encrypted password
Encrypted auth result
loop
[Chat]
Encrypted message
Encrypted broadcast
Client
Server
All Clients


KALI (Snapshot 5) [Running] - Oracle VirtualBox
File Machine View Input Devices Help
File Edit View Go Bookmarks Help
File Actions Edit View Help
Places
Devices
Network
bobb
Enter password:
123
AUTHENTICATED....-----BEGIN DH
-----END DH PARAMETERS-----
... -----BEGIN PUBLIC KEY-----
python3 encrypted_client.py
Connected to server at 10.0.0.217:65432
Enter your username: bobb
Enter your password: 123
Authentication failed. Please try again.
Enter your username:


Client
Server
TCP Connection
DH Parameters
Client Public Key
Server Public Key
AES Key Derived
Encrypted Username
Encrypted "Enter Password"
Encrypted Password
alt
[Valid Credentials]
Encrypted "AUTHENTICATED"
[Invalid]
Encrypted "AUTH_FAIL"
Client
Server