

# BEFORE ENCRYPTION

**Server**

```
(kali@kali)~$ python server.py
Server listening on 10.0.0.217:65432
New connection from (10.0.0.195, 37890)
User bobb authenticated successfully
Number of Online Users: 1
New connection from (10.0.0.195, 32778)
User alice authenticated successfully
Number of Online Users: 2
alice: Hie
bobb: Hey Alice
alice: Is Our Chat Encrypted?
bobb: I Hope so
```

**Client 1**

```
(kali@kali)~$ python3 tcp_client.py
Connected to server at 10.0.0.217:65432
Enter your username: bobb
Enter your password: 123
Welcome bobb
You:
alice: Hie
You: Hey Alice
You:
alice: Is Our Chat Encrypted?
You: I Hope so
You:
```

**Client 2**

```
(kali@kali)~$ python3 tcp_client.py
Connected to server at 10.0.0.217:65432
Enter your username: alice
Enter your password: 123
Welcome alice
You: Hie
You:
bobb: Hey Alice
You: Is Our Chat Encrypted?
You:
bobb: I Hope so
You:
```

**Wireshark: Follow TCP Stream (tcp.stream eq 0) - eth0**

```
bobb
Enter password:
123
AUTHENTICATED:alice: Hie
Hey Alice
alice: Is Our Chat Encrypted?
I Hope so
```

**Readable plain text**

# AFTER ENCRYPTION

**Client 1**

```
(encrypted_client1)kali@kali:~/Desktop/encrypted_chat
File Actions Edit View Help
Using cached pycparser-2.22-py3-none-any.whl.metadata (943 bytes)
Using cached cryptography-45.0.6-cp311-abi3-manylinux_2_34_x86_64.whl (4.5 M
9)
Using cached cffi-1.17.1-cp312-cp312-manylinux_2_17_x86_64.whl (479 kB)
Installing collected packages: pycparser, cffi, cryptography
Successfully installed cffi-1.17.1 cryptography-45.0.6 pycparser-2.22
(encrypted_client1)kali@kali:~/Desktop/encrypted_chat
$ python3 encrypted_client.py
Connected to 10.0.0.217:65432
[DEBUG] Derived AES key: 08f8722eae5f28f364bf28f2cb29c59adf74f2a4278477cb3
7721af7c319c
Enter your username: bobb
Enter your password: 123
Welcome bobb!
alice: Hie Bobb
You:
alice: Is Our Chat Encrypted?
You: I hope so!!
```

**Server**

```
(encrypted_server)kali@kali:~/Desktop/Encrypted chat
File Actions Edit View Help
$ python3 encrypted_server.py
Server listening on 10.0.0.217:65432
[DEBUG] Derived AES key: e24530438da5a51782c76a7a73ba8af3864fcd8e4326a8c878f58
af4d157f8
User alice connected from (10.0.0.192, 52728)
[DEBUG] Derived AES key: 08f8722eae5f28f364bf28f2cb29c59adf74f2a4278477cb3
29c59adf74f2a4278477cb37721af7c319c
User bobb connected from (10.0.0.195, 44268)
alice: Hie Bobb
alice: Is our Chat Encrypted?
bobb: i hope so!!
```

**Client 2**

```
(encrypted_client2)kali@kali:~/Desktop/encrypted_chat
File Actions Edit View Help
Computer Fully charged
(encrypted_client2)kali@kali:~/Desktop/encrypted_chat
$ python3 encrypted_client.py
Connected to 10.0.0.217:65432
[DEBUG] Derived AES key: e24530438da5a51782c76a7a73ba8af3864fcd8e4326a8c878f58
af4d157f8
Enter your username: alice
Enter your password: 123
Welcome alice!
Hie Bobb
Is our Chat Encrypted?
bobb: I hope so!!
You:
```

**Wireshark: Follow TCP Stream (tcp.stream eq 0) - eth0**

```
v+18X1vGd1Q5oLA9LY14i5cJHPGax9rLMDMM6PBweo3J2LWHEsaj2n0Y3xByCyX
PpIawDuunsjc2YCR888+6dMscdgvpgLfwSVU75mQtVgpeCDB4/NIE7SAPIGI440
BBuUsj4LE3PMAT7zbFqRfMo56812LvesecM071Jox5KBZGT8FPS1/J4rjccCAQID
ggEFAAKCAQBNhrE01TRc/xUfxfvztX/8YAKxOU1J4MuoxKQVRWgSFFKbQdQjXQ
FwbbJMAUp/ehk5yzR1gZ8WYjQDwX48NW-iDp+Pg/UG6fKQZANX92ZCZv0b2
0TA29SWmcMwKQHPsueqr8nIVgfrq9v1bA68Vno01-wJYtGdQMUXrZec
Vvyhng/93HgrNM5tped5JorKrnW2YrvtVyhstNVNehABazKyhHELExrEM96
/TTLaMNA4gW4hVdLGFoFCYCIg745gPg2L+5eNbcRMf51Ghd0In08CKf5GKvb6
vF1cxTgblrAZ6eP1dmOn3A0edPn5
-----END PUBLIC KEY-----
.....2.6.....
.....h.....B.YT.....Q.M
.....e0B_2f.....a.D.7k.....
.....3.i.g.....
.....j.s.s>e.7YX.n.8.....g7e.n./C.c.z.W.....(.....A.A.q.3.....S.A.z.K.
.....J.M.....R.....YI.w.....ZBBh.q.s.j.p.W.X0.
-----
```

# Steps To Achieving Encryption

## Phase 1: Cryptographic Setup

### 1: Import Required Libraries

```
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
```

### 2: Generate Diffie-Hellman Parameters

```
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

```
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

```
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
        )
    )

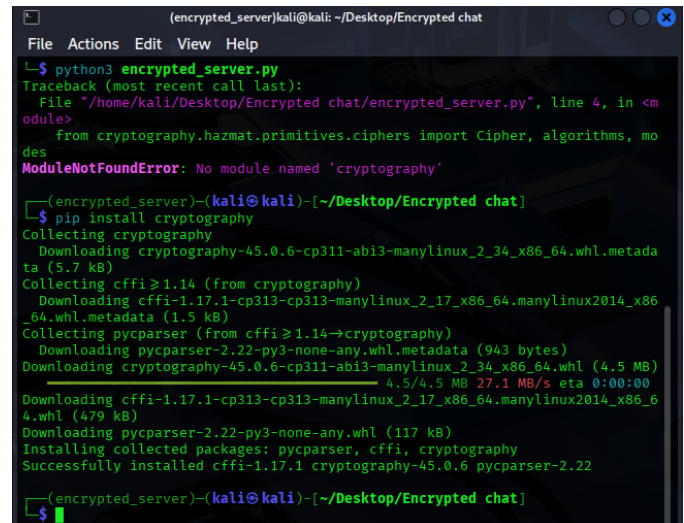
    # Step 2: Server generates its private key
    server_private_key = self.dh_parameters.generate_private_key()

    # 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)
```



The screenshot shows a terminal window titled "(encrypted\_server)kali@kali: ~/Desktop/Encrypted chat". It displays a Python traceback for a `ModuleNotFoundError: No module named 'cryptography'`. The user then runs `pip install cryptography`, which shows the process of downloading and installing `cryptography-45.0.6`, `cfssl-1.17.1`, and `pycparser-2.22`.

## Phase 3 Add Encryption and Decryption Function

```
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
```

ENCRYPTION

```
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()
```

DECRYPTION

# Application Flow

**CLIENT 1**



**SERVER**



**ALL CLIENTS**

