Hybrid Encryption

This lab shows the foundational mechanism behind secure communication:

- RSA -> protects small secrets (like AES keys).
- AES -> protects the actual data.
- Sockets -> move encrypted bytes across the network.
- Pickle -> bundles everything so both sides interpret the data correctly.

It proves one thing:

Secure communication = correct cryptography + correct data transfer.

Key Concepts (My Version)

1. Hybrid Encryption

It combines both

- AES (symmetric) - encrypts the actual message
- RSA (Asymmetric) - encrypts the AES key (secure exchange) RSA is secure but slow, while AES is fast but it requires a shared key

2. AES Key

Randomly generated for each message must remain secret.

3. IV (Initialization Vector)

Random for each encryption Ensures to produce different ciphertexts for repeated messages Not secret, must be shared with the receiver

4. Public/Private Keys

- Public Key - encrypt AES key (anyone can use it)
- Private key - decrypt AES key (must be kept secret)

5. Socket Basics

Data is sent as bytes may need chunking for large messages

- socket.bind() -> where server listens

- socket.listen() -> starts listening for connections
- socket.accept() -> accepts incoming connections
- socket.connect() -> client connects to the server Networking basics are simple: server listens, client connects and data flows Both sides (sender & receiver) must use UTF-8 or byte encoding.

6. Pickle/Serialization

Python objects can't be sent raw over a network.

- pickle.dumps() -> serialize tuple (encrypted_key, iv, encrypted_message)
- pickle.loads() -> deserialize on the other side Always package all parts together, receiver unpacks them

7. AES CFB Mode (Cipher Feedback Mode)

Purpose: It makes AES work like a stream cipher (can encrypt data of any length).

Working:

- It encrypts the previous ciphertext block (or IV for the first one) to create a keystream
- The keystream is then XORed with the (first or next block in the sequence of) plaintext to produce ciphertext.
- Each cipher text block depends on the one before it

Advantages: No padding needed, errors affect only one block, repeated messages look different (because of IV and chaining)

Result: Secure, flexible, and produces unique ciphertexts even for identical plaintexts.

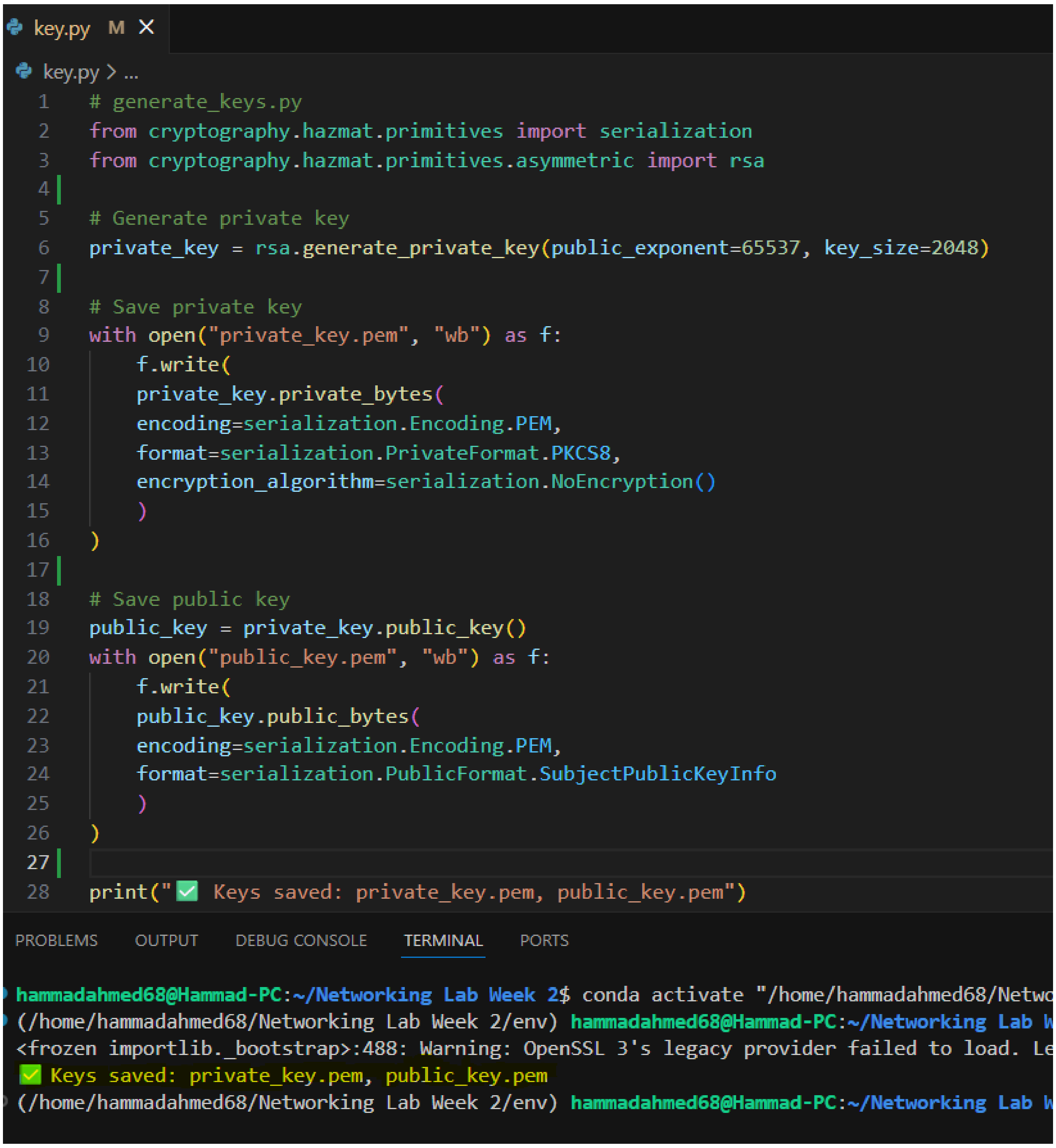
Workflow of Hybrid Encryption

Step 1: Key Generation:

Generate RSA key pair:

- Public Key -> public_key.pem
- Private Key -> private_key.pem

Done once per user/server



```python
# generate_keys.py
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa

# Generate private key
private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)

# Save private key
with open("private_key.pem", "wb") as f:
    f.write(
    private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.PKCS8,
    encryption_algorithm=serialization.NoEncryption()
    )
)

# Save public key
public_key = private_key.public_key()
with open("public_key.pem", "wb") as f:
    f.write(
    public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
    )
)

print("✅ Keys saved: private_key.pem, public_key.pem")
```

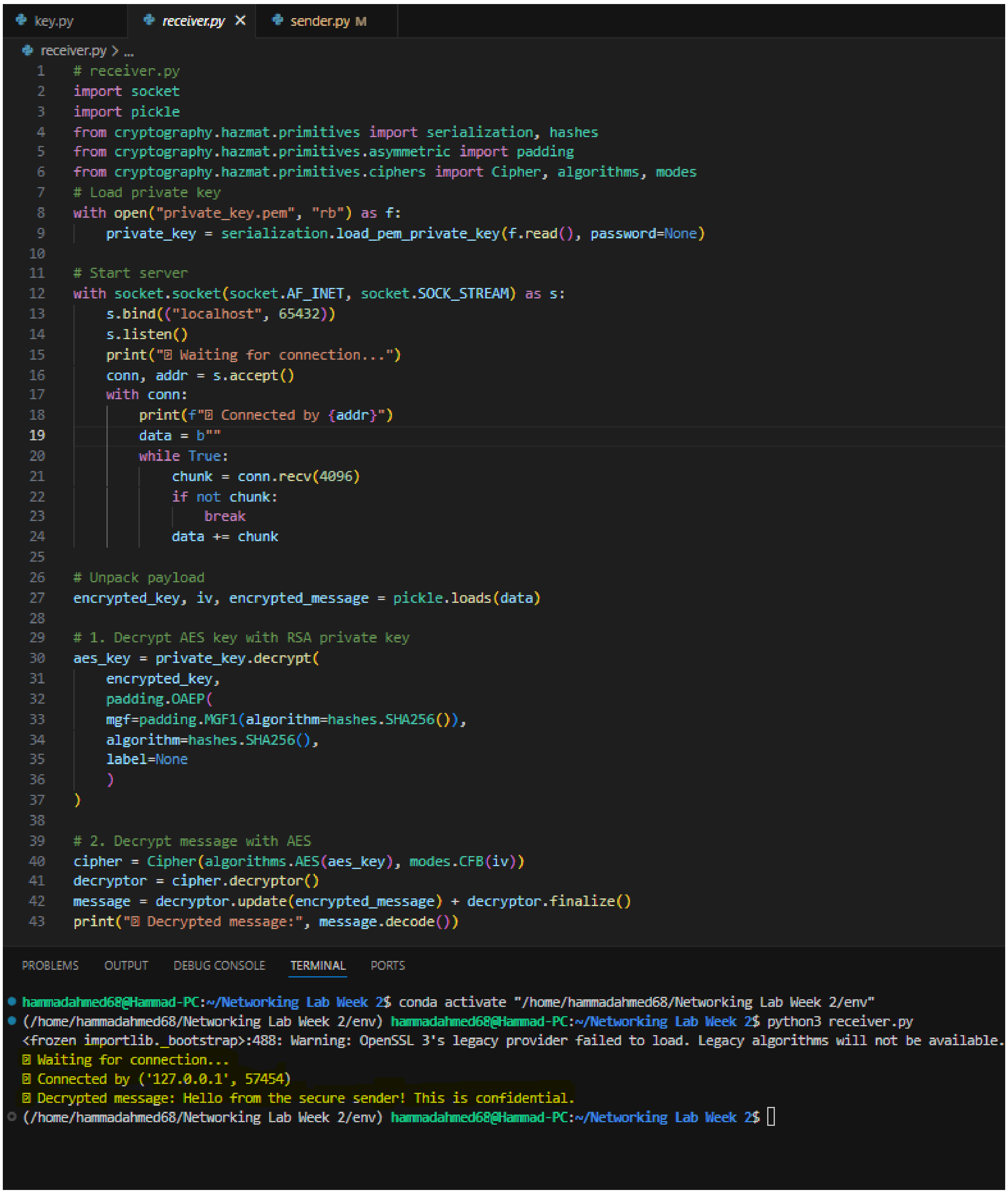
```
hammadahmed68@Hammad-PC:~/Networking Lab Week 2$ conda activate "/home/hammadahmed68/Netwo
(/home/hammadahmed68/Networking Lab Week 2/env) hammadahmed68@Hammad-PC:~/Networking Lab W
<frozen importlib._bootstrap>:488: Warning: OpenSSL 3's legacy provider failed to load. Le
✅ Keys saved: private_key.pem, public_key.pem
(/home/hammadahmed68/Networking Lab Week 2/env) hammadahmed68@Hammad-PC:~/Networking Lab W
```

Step 2 – Receiver (Server)

What the server does logically:

- Loads its private key.
- Starts a TCP server on port 65432.
- Waits for connection.
- Receives the full encrypted payload.
- Extracts (encrypted_key, iv, encrypted_message) using pickle.
- Uses RSA private key → decrypt AES key.
- Uses AES-CFB with IV → decrypts message.
- Prints the plaintext.

The important security idea: Only the holder of the private key can decrypt the AES key, so only they can read the message.



```python
# receiver.py
import socket
import pickle
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
# Load private key
with open("private_key.pem", "rb") as f:
    private_key = serialization.load_pem_private_key(f.read(), password=None)

# Start server
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind(("localhost", 65432))
    s.listen()
    print("⍰ Waiting for connection...")
    conn, addr = s.accept()
    with conn:
        print(f"⍰ Connected by {addr}")
        data = b""
        while True:
            chunk = conn.recv(4096)
            if not chunk:
                break
            data += chunk

# Unpack payload
encrypted_key, iv, encrypted_message = pickle.loads(data)

# 1. Decrypt AES key with RSA private key
aes_key = private_key.decrypt(
    encrypted_key,
    padding.OAEP(
    mgf=padding.MGF1(algorithm=hashes.SHA256()),
    algorithm=hashes.SHA256(),
    label=None
    )
)

# 2. Decrypt message with AES
cipher = Cipher(algorithms.AES(aes_key), modes.CFB(iv))
decryptor = cipher.decryptor()
message = decryptor.update(encrypted_message) + decryptor.finalize()
print("⍰ Decrypted message:", message.decode())
```

```
hammadahmed68@Hammad-PC:~/Networking Lab Week 2$ conda activate "/home/hammadahmed68/Networking Lab Week 2/env"
(/home/hammadahmed68/Networking Lab Week 2/env) hammadahmed68@Hammad-PC:~/Networking Lab Week 2$ python3 receiver.py
<frozen importlib._bootstrap>:488: Warning: OpenSSL 3's legacy provider failed to load. Legacy algorithms will not be available.
⍰ Waiting for connection...
⍰ Connected by ('127.0.0.1', 57454)
⍰ Decrypted message: Hello from the secure sender! This is confidential.
(/home/hammadahmed68/Networking Lab Week 2/env) hammadahmed68@Hammad-PC:~/Networking Lab Week 2$
```

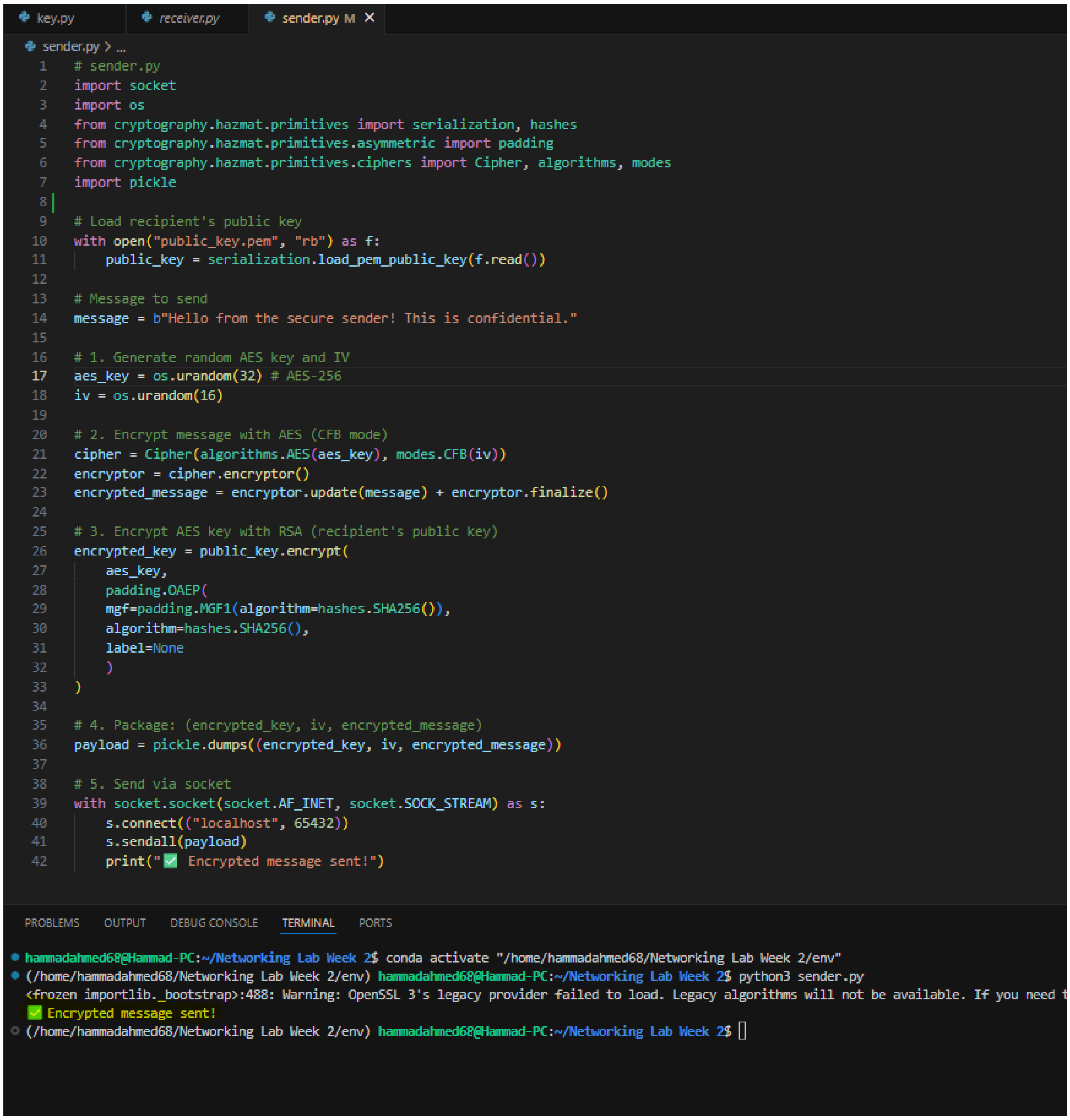
Step 3 – Sender (Client)

What the sender does logically:

- Loads receiver's public key.
- Prepares plaintext → converts to bytes.
- Generates fresh AES key + IV.
- AES-encrypts the message in CFB mode.

- RSA-encrypted the AES key.
- Packages everything via pickle.
- Sends to the server over TCP.

Key Point: The sender DOES NOT need the private key. They only need the recipient's public key.



```python
# sender.py
import socket
import os
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
import pickle

# Load recipient's public key
with open("public_key.pem", "rb") as f:
    public_key = serialization.load_pem_public_key(f.read())

# Message to send
message = b"Hello from the secure sender! This is confidential."

# 1. Generate random AES key and IV
aes_key = os.urandom(32) # AES-256
iv = os.urandom(16)

# 2. Encrypt message with AES (CFB mode)
cipher = Cipher(algorithms.AES(aes_key), modes.CFB(iv))
encryptor = cipher.encryptor()
encrypted_message = encryptor.update(message) + encryptor.finalize()

# 3. Encrypt AES key with RSA (recipient's public key)
encrypted_key = public_key.encrypt(
    aes_key,
    padding.OAEP(
    mgf=padding.MGF1(algorithm=hashes.SHA256()),
    algorithm=hashes.SHA256(),
    label=None
    )
)

# 4. Package: (encrypted_key, iv, encrypted_message)
payload = pickle.dumps((encrypted_key, iv, encrypted_message))

# 5. Send via socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect(("localhost", 65432))
    s.sendall(payload)
    print("✅ Encrypted message sent!")
```

```
hammadahmed68@Hammad-PC:~/Networking Lab Week 2$ conda activate "/home/hammadahmed68/Networking Lab Week 2/env"
(/home/hammadahmed68/Networking Lab Week 2/env) hammadahmed68@Hammad-PC:~/Networking Lab Week 2$ python3 sender.py
<frozen importlib._bootstrap>:488: Warning: OpenSSL 3's legacy provider failed to load. Legacy algorithms will not be available. If you need t
✅ Encrypted message sent!
(/home/hammadahmed68/Networking Lab Week 2/env) hammadahmed68@Hammad-PC:~/Networking Lab Week 2$ 
```

Reflection

Practical Summary (My Condensed Version)

Sender Path

Plaintext

- AES-encrypt with AES key + IV
- ciphertext
- package (RSA(AES key), IV, ciphertext)
- send via socket

Receiver path

Receive package

- RSA-decrypt AES key
- AES-decrypt ciphertext using AES key + IV
- recover plaintext

This is literally how secure messaging works (Signal, WhatsApp, TLS handshake, etc.).

Conclusion

This lab is a minimal version of modern secure communication pipelines.

- RSA only protects the key.
- AES does the real encryption.
- IV prevents pattern leaks.
- Pickle ensures structured transfer.
- Sockets push bytes, nothing else.
- CFB removes padding problems and keeps ciphertext unique.