

# Lab 2

This lab demonstrates how **secure communication works in real networks** using a combination of:

- **RSA (asymmetric encryption)** for secure key exchange
- **AES (symmetric encryption)** for fast data encryption
- **Python sockets** for sending encrypted data between two computers

The lab simulates how secure systems such as **HTTPS** and **VPNs** protect data in real life.

Generation of public and private keys and saving of the keys

```
# Open private_key.pem, read
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())

```

[3] Python

... ● Waiting for connection...  
🔗 Connected by ('127.0.0.1', 59529)  
👉 Decrypted message: Hello from the secure sender! This is confidential.

Starting the server, unpack the payload and decrypt AES key with RSA private key and decrypt message with AES

# Lab 2

private\_key.pem

```
1 -----BEGIN PRIVATE KEY-----
2 MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggsjAgEAAoIBAQDAj/LHgo+4GPDo
3 QswvEXULznZtQv74ZIwWjHDA7Zvp+Lns9AlHcztBj5lSE9wkQthwKiOLi1xh7rw
4 xl8obNs+D4sj/u8usQNXD6RFYAQJtlYenxVoczqYMD2lxz5kM0rSA3ArnwAIk7W
5 i2/Iss57jq4idI0LgiZqo0l5x4KapC78nxCgxVj0kF0Iv3tnBdjsBCE6UrPsYZ6a
6 uXF501TtZotWfmYay4jIPsE6YGouiJeji7xogmsQna1sMWZ7voHFpL7MwAnsqusQ
7 U1g0sgVfcJQVTbpFp2F1l9yr7HY96Exun0A7JYrUhDMPjVaR1GpJOMXku4Ln+gv+
8 SJ9A8IjpAgMBAAECggEAXBJ4Gz0qSvRh/Cw0Jw2x0Xte2N1AaadppwT5Vi2oRasY
9 q4EmvRiKq6qYmwUANuqjbpIYBi0+RtKHaCjxxvqFuh7kgMmcw86PskgwquWQq6U
10 bv/R1KeEFNC8KU9JXscF0vQY31C0qeG9oIf7YxNb9g5iu//oMksZVmbSuDEA3w3
11 LTYJ4S3BXf3m2+ZYyyp6W4JL3N8uYvlk9F9YgFNL9avTD79G3YEzn916UxSpz+5
12 JbyxTOntKyyNT85adh458lpVYa+zq/bQEz9wfKmRcUhJw0FVTSL3TzFlp0nqoJD
13 wZGpcpq8KO/cJITZL8kjP1WEUoiGNktwmBm5mJOFOqKBgQD/k55uXbP6qxHtaMZZ
14 ersBxkpicytchCMKMIJL4P3Fxas8Cjxk96YHKzdM7VyaIV5qt8598Mwo2oqkKi9J+
15 YH8Tdn/163hGaM5Jqhz6awUvXd32hyvx1kmo0fenChHvk+Pg7GSP6PitZxDEIhno
16 1ioQDON3boV46hN74Jgeea+z2wKBgQDA4Zt4T0rcPtGYQv6Ey2Q4BYhGQas1GhCY
17 q+qiArpMh4+OCyZ95pX6PUisivH0f7nAqFXBHMFinb5ADYQ/CD3fgScP7q5900Mwo
18 e+A+k6G7VmCPvolJpkjdWJ7QXjAg0lrrPPxZvQ9P9nj02KaLD5mPjBeX5oQuNK9UF
19 N40yZcPziwKBgEAhjabKEoh69s0o/+SiHFzkofPKCS0vLAo3fkmlwue0o1osg10em
20 2W/HWBLS2pNp0CGIOWPT9uMgy+Qek9Isapa6rH9L9+FLGUB41E6ttbs+BZ1/6gnh
21 Z1g004WJzFHYi1z8VHucDRKOAzIEEJZRbzqZ0VY9wHZH312+TM2Q4VrzAoGBAJrZ
22 xZ60UjSGPnE2f+40+H09hU2aT+iRd3eKtgHBnN3qC4dnTkzPveK8KMFHeLCS1fxD
23 X/PXTamKm50K5MLQ9lxcAvrfKzN6gr2kTkC7eMnVLArbVo618XwOs7HnnE3m6m7u
24 YE2Nr/kyfEQqoyERq2nNB3glIggBg8XfLH2kk+/HAoGAazHsmRvQdbsToxs63Te3
25 neeoLu4f1LzajXpwhH74XbwovBa8H95oc2nuVFmVZK4VfbSJHmrNT/2BbZ4psN6k
26 xBve2g0PGPsQBIoG+vr1VpMVjn0w45y50w5kWP1jExKQvNVOv35uALX5e5PhXYq1
27 z1NlV0ytKJ0yXSD3JAKq2CA=
28 -----END PRIVATE KEY-----
29
```

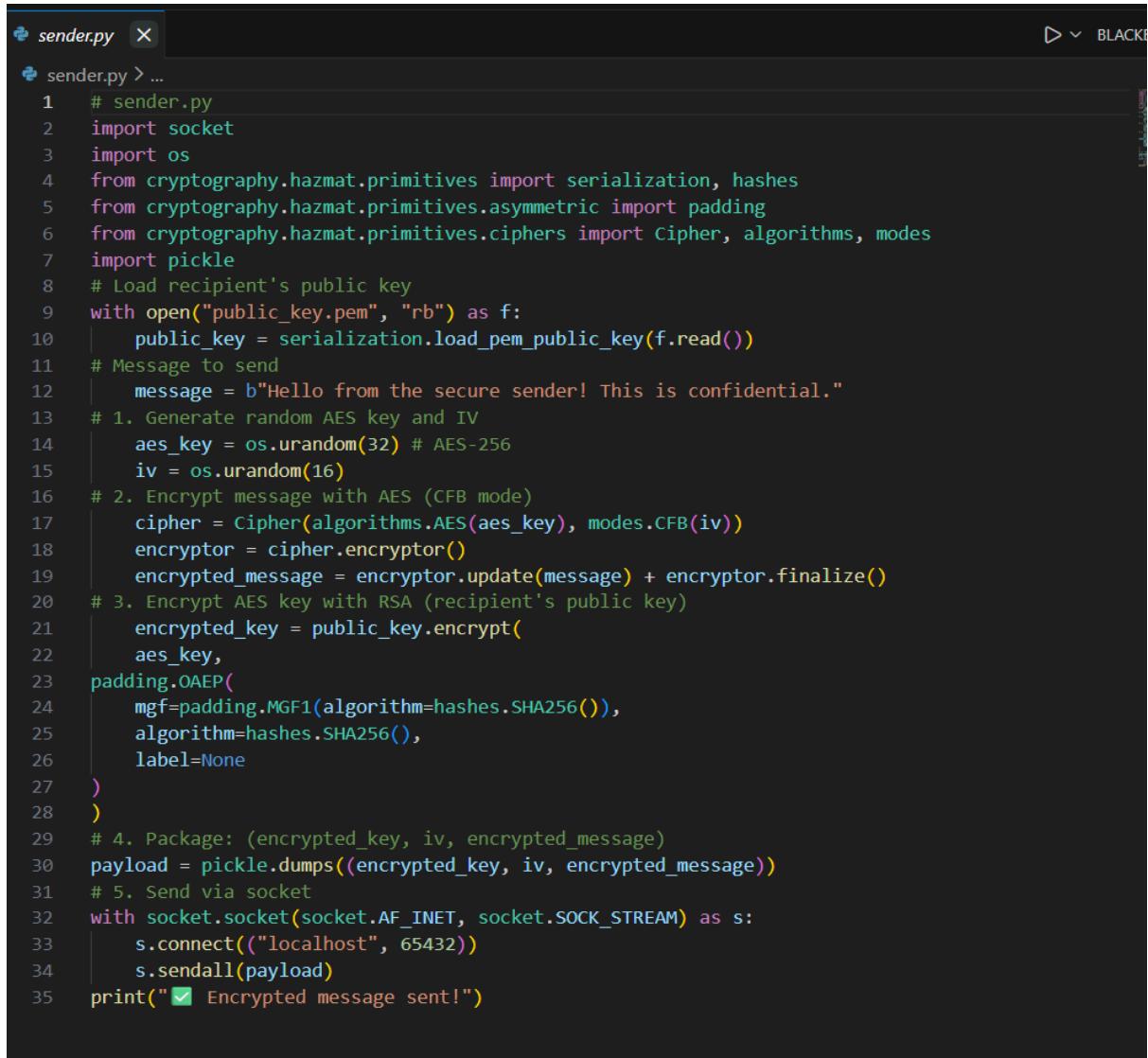
Begin and end of the generation of the public key

public\_key.pem

BLACKBOX ...

```
1 -----BEGIN PUBLIC KEY-----
2 MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAWI/yx4KPUbjw6ELL8BF1
3 C852dbUL++GSMFoxww02b6fi57PQR3M7QY+ZUhPcJELYccoji4tcYe68MZFKGzb
4 Pg+LI/7vLrEDWw+kRWAECbZwHp8VaHM6mDA9pcV2eZDNK0gNwK58ACJ01otvyLL0
5 e46uInSNC4ImaqDpeceCmqQu/J8QoMVYzpBdCL97zwXY7Aqh0lKz7GGemrlxeTtU
6 7WaLvN5mGsuIyD7B0mBqlOix04u8aIJrEJ2tbDFme76BxaS+zMAJ7K1EkFNYNLIF
7 X3CUFU26RadhdZfcq+x2PehF7pzg0yWK1IQzD41WkdRqSTjf5LuC5/oL/kifQPC1
8 6QIDAQAB
9 -----END PUBLIC KEY-----
10
```

# Lab 2



The screenshot shows a code editor window with a dark theme. The title bar says "sender.py". The code is a Python script for sending a secure message. It uses the cryptography library to handle encryption and RSA keys. The code includes importing socket, os, and cryptography modules, loading a public key from a file, generating a random AES key and IV, encrypting the message with AES in CFB mode, encrypting the AES key with RSA using OAEP padding, and finally sending the package (encrypted key, IV, encrypted message) via a socket to localhost port 65432.

```
# sender.py > ...
1 # sender.py
2 import socket
3 import os
4 from cryptography.hazmat.primitives import serialization, hashes
5 from cryptography.hazmat.primitives.asymmetric import padding
6 from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
7 import pickle
8 # Load recipient's public key
9 with open("public_key.pem", "rb") as f:
10     public_key = serialization.load_pem_public_key(f.read())
11 # Message to send
12     message = b"Hello from the secure sender! This is confidential."
13 # 1. Generate random AES key and IV
14     aes_key = os.urandom(32) # AES-256
15     iv = os.urandom(16)
16 # 2. Encrypt message with AES (CFB mode)
17     cipher = Cipher(algorithms.AES(aes_key), modes.CFB(iv))
18     encryptor = cipher.encryptor()
19     encrypted_message = encryptor.update(message) + encryptor.finalize()
20 # 3. Encrypt AES key with RSA (recipient's public key)
21     encrypted_key = public_key.encrypt(
22         aes_key,
23         padding.OAEP(
24             mgf=padding.MGF1(algorithm=hashes.SHA256()),
25             algorithm=hashes.SHA256(),
26             label=None
27         )
28     )
29 # 4. Package: (encrypted_key, iv, encrypted_message)
30     payload = pickle.dumps((encrypted_key, iv, encrypted_message))
31 # 5. Send via socket
32     with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
33         s.connect(("localhost", 65432))
34         s.sendall(payload)
35     print("✓ Encrypted message sent!")
```

Full code