Project Report

Title: Secure Communication System

Group 2: Cryptography & Encryption

Project completed by the following members:

- ***Chukwuma Fumnanya Divine***
- ***Apena Dare David***
- ***David Oko Odu***

Project Objective:

The goal of this project is to design and implement a Python-based secure communication system that ensures data confidentiality, integrity, and secure key exchange between two clients.

Technologies & Libraries Used (In Kali Linux)

- Python 3.13.5
- PyCryptodome library (for AES, RSA encryption, and hashing)
- hashlib (for SHA-256 hashing)
- base64 (for safe encoding/decoding of encrypted data)

System Components

The project is built using five to six (5-6) Python modules, each handling a different cryptographic process:

1. *"crypto_utils.py"*

Handles AES (Advanced Encryption Standard) encryption and decryption of text messages using EAX mode, which provides both confidentiality and authentication.

Code:

```
from Crypto.Cipher import AES, PKCS1_OAEP

from Crypto.PublicKey import RSA

from Crypto.Random import get_random_bytes

from Crypto.Hash import SHA256

import base64
```

```
# ---------- AES ENCRYPTION / DECRYPTION ----------
def encrypt_aes(plaintext, key):
    """
    Encrypts a plaintext message using AES (EAX mode).
    Returns a base64-encoded string containing nonce + tag + ciphertext.
    """
    cipher = AES.new(key, AES.MODE_EAX)
    ciphertext, tag = cipher.encrypt_and_digest(plaintext.encode())
    combined = cipher.nonce + tag + ciphertext
    return base64.b64encode(combined).decode()

def decrypt_aes(encoded_ciphertext, key):
    """
    Decrypts a base64-encoded ciphertext using AES (EAX mode).
    Returns the original plaintext if the integrity check passes.
    """
    raw = base64.b64decode(encoded_ciphertext)
    nonce = raw[:16]
    tag = raw[16:32]
    ciphertext = raw[32:]
    cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
    plaintext = cipher.decrypt_and_verify(ciphertext, tag)
return plaintext.decode()

# ---------- RSA ENCRYPTION / DECRYPTION ----------
```

```python
def encrypt_rsa(data, public_key_pem):
    """
    Encrypts data (bytes) using an RSA public key.
    """
    public_key = RSA.import_key(public_key_pem)
    cipher_rsa = PKCS1_OAEP.new(public_key)
    encrypted_data = cipher_rsa.encrypt(data)
    return base64.b64encode(encrypted_data).decode()


def decrypt_rsa(encoded_data, private_key_pem):
    """
    Decrypts base64-encoded data using an RSA private key.
    """
    private_key = RSA.import_key(private_key_pem)
    cipher_rsa = PKCS1_OAEP.new(private_key)
    decrypted_data = cipher_rsa.decrypt(base64.b64decode(encoded_data))
    return decrypted_data


# ---------- SHA-256 HASHING ----------
def generate_hash(message):
    """
    Generates a SHA-256 hash for a given message.
"""
    h = SHA256.new(message.encode())
    return h.hexdigest()
```

Output:

```
┌──(venv)─(wickedjamezz㉿starboy10)-[~/cryptography_secure_messaging_project]
└─$ python3
Python 3.13.5 (main, Jun 25 2025, 18:55:22) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from crypto_utils import encrypt_aes, decrypt_aes
... from Crypto.Random import get_random_bytes
...
... key = get_random_bytes(16)
... cipher = encrypt_aes("Hello Mama!", key)
... print(cipher[:80] + " ... ")
... print(decrypt_aes(cipher, key))
...
TREpQUspYHM2lnfu/ZvhavTBC/PQI907NPBu2Ifd+/rBDxP1lCnzjpGdKw= ...
Hello Mama!
>>> exit
```

2. ***"generate_keys.py"***

Generates a 2048-bit RSA key pair (public and private keys).

- Public key >> used to encrypt the AES session key.
- Private key >> used to decrypt the AES session key.

Code:

```
from Crypto.PublicKey import RSA


def generate_keys():
    # Generate 2048-bit RSA key pair
    key = RSA.generate(2048)


    private_key = key.export_key()
    public_key = key.publickey().export_key()


    # Save the private key
    with open("private.pem", "wb") as priv_file:
        priv_file.write(private_key)
```

```
    # Save the public key
    with open("public.pem", "wb") as pub_file:
        pub_file.write(public_key)

    print("✅ RSA key pair generated successfully!")
    print("Private key saved as private.pem")
    print("Public key saved as public.pem")

if __name__ == "__main__":
    generate_keys()
```

Output:

```
┌──(venv)─(wickedjamezz㉿starboy10)-[~/cryptography_secure_messaging_project]
└─$ python3 generate_keys.py
✅ RSA key pair generated successfully!
Private key saved as private.pem
Public key saved as public.pem
```

3. ***"rsa_encrypt_decrypt.py"***

AES is symmetric, meaning both sides need the same key but then, how do we share the **AES** session key securely between two clients?

That's where **RSA (Asymmetric encryption)** comes in:

- Each person has a public key and a private key.
- The public key can be publicly shared freely because it encrypts data.
- The private key is kept hidden and a secret because it decrypts data.

So when one user wants to send a secure message:

- They generate a random **AES** key (session key).
- They encrypt the **AES** key using the recipient's **RSA** public key.
- Only the recipient can decrypt it using their private key.

Code:

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import base64

def encrypt_session_key(session_key, public_key_path="public.pem"):
    # Load the recipient's public key
    with open(public_key_path, "rb") as pub_file:
        recipient_key = RSA.import_key(pub_file.read())
        cipher_rsa = PKCS1_OAEP.new(recipient_key)

    # Encrypt the AES session key
    encrypted_session_key = cipher_rsa.encrypt(session_key)
    return base64.b64encode(encrypted_session_key).decode('utf-8')


def decrypt_session_key(encrypted_session_key_b64, private_key_path="private.pem"):
    # Load the private key
    with open(private_key_path, "rb") as priv_file:
        private_key = RSA.import_key(priv_file.read())
        cipher_rsa = PKCS1_OAEP.new(private_key)

    # Decode and decrypt the AES session key
    encrypted_session_key = base64.b64decode(encrypted_session_key_b64)
```

```python
    decrypted_session_key = cipher_rsa.decrypt(encrypted_session_key)
    return decrypted_session_key



if __name__ == "__main__":
# Example usage
    from Crypto.Random import get_random_bytes
    session_key = get_random_bytes(16)  # 128-bit AES key

    print("Original AES Session Key:", session_key)

    encrypted = encrypt_session_key(session_key)
    print("\n Encrypted Session Key (Base64):", encrypted)

    decrypted = decrypt_session_key(encrypted)
    print("\n Decrypted AES Session Key:", decrypted)

    if decrypted == session_key:
        print("\n✅RSA encryption & decryption successful!")
    else:
        print("\n❌Something went wrong.")
```

Output:

4. ***"hash_utils.py"***

Implements **SHA-256** hashing to verify message integrity.

The hash ensures that the message or file received has not been altered during transmission.

Code:

```
import hashlib


def calculate_sha256(data):
    """
    Calculate SHA-256 hash of the given data.
    """
    if isinstance(data, str):
        data = data.encode('utf-8')
    hash_object = hashlib.sha256(data)
    return hash_object.hexdigest()


if __name__ == "__main__":
    # Example usage
    message = "This project is done in fulfillment of the GetBundi Cyber security training."
    hash_value = calculate_sha256(message)
    print("Message:", message)
```

```
    print("SHA-256 Hash:", hash_value)
```

Output:

```
┌──(venv)-(wickedjamezz㉿starboy10)-[~/cryptography_secure_messaging_project]
└─$ nano hash_utils.py

┌──(venv)-(wickedjamezz㉿starboy10)-[~/cryptography_secure_messaging_project]
└─$ python3 hash_utils.py
Message: This project is done in fulfillment of the GetBundi Cyber security training.
SHA-256 Hash: 0c6864e4554c34d2eb95535dbcdebc7a79db84517dd7a26203bae58cb6c2ba02
```

4. ***"secure_messaging_app.py"***

Integrates **AES**, **RSA**, and **SHA-256** into one secure messaging workflow:

- **AES** encrypts the actual message.
- **RSA** encrypts the **AES** session key.
- **SHA-256** generates a hash to verify integrity.

Code:

```
from Crypto.Cipher import AES, PKCS1_OAEP

from Crypto.PublicKey import RSA

from Crypto.Random import get_random_bytes

from base64 import b64encode, b64decode

import hashlib


# --- AES Encryption & Decryption ---

def encrypt_message(message, aes_key):

    cipher = AES.new(aes_key, AES.MODE_EAX)

    ciphertext, tag = cipher.encrypt_and_digest(message.encode('utf-8'))

    return {

        'ciphertext': b64encode(ciphertext).decode('utf-8'),

        'nonce': b64encode(cipher.nonce).decode('utf-8'),

        'tag': b64encode(tag).decode('utf-8')

    }
```

```
def decrypt_message(encrypted_data, aes_key):
    nonce = b64decode(encrypted_data['nonce'])
    tag = b64decode(encrypted_data['tag'])
    ciphertext = b64decode(encrypted_data['ciphertext'])
    cipher = AES.new(aes_key, AES.MODE_EAX, nonce=nonce)
    message = cipher.decrypt_and_verify(ciphertext, tag)
    return message.decode('utf-8')


# --- RSA Encryption & Decryption of AES Key ---
def encrypt_session_key(session_key, public_key_path="public.pem"):
    with open(public_key_path, "rb") as pub_file:
        recipient_key = RSA.import_key(pub_file.read())
       cipher_rsa = PKCS1_OAEP.new(recipient_key)
    encrypted_key = cipher_rsa.encrypt(session_key)
    return b64encode(encrypted_key).decode('utf-8')


def decrypt_session_key(encrypted_session_key_b64, private_key_path="private.pem"):
    with open(private_key_path, "rb") as priv_file:
        private_key = RSA.import_key(priv_file.read())
        cipher_rsa = PKCS1_OAEP.new(private_key)
    encrypted_key = b64decode(encrypted_session_key_b64)
    session_key = cipher_rsa.decrypt(encrypted_key)
    return session_key
```

```
# --- SHA-256 Hashing ---

def calculate_sha256(data):

    if isinstance(data, str):

        data = data.encode('utf-8')

    return hashlib.sha256(data).hexdigest()


# --- Demonstration ---

if __name__ == "__main__":

    # Sender generates AES session key

    aes_key = get_random_bytes(16)

    print("Generated AES Session Key:", aes_key)


    # Sender encrypts a message

    message = "Hello Mr Adebayo Azeez, this message is from group 2 for this
Cybersecurity training. We need the IP address of your device for practice purposes, LOL."

    encrypted_data = encrypt_message(message, aes_key)

    print("\n Encrypted Message:", encrypted_data)

    # Sender computes SHA-256 hash for integrity

    hash_value = calculate_sha256(message)

    print("\n Message Hash (SHA-256):", hash_value)


    # Sender encrypts the AES session key with receiver's RSA public key

    encrypted_session_key = encrypt_session_key(aes_key)

    print("\n Encrypted AES Session Key:", encrypted_session_key)


    # Receiver decrypts AES session key with their private RSA key
```

```
    decrypted_session_key = decrypt_session_key(encrypted_session_key)
    print("\n Decrypted AES Session Key:", decrypted_session_key)

    # Receiver decrypts message
    decrypted_message = decrypt_message(encrypted_data, decrypted_session_key)
    print("\n Decrypted Message:", decrypted_message)

    # Receiver recalculates hash and verifies integrity
    received_hash = calculate_sha256(decrypted_message)
    if received_hash == hash_value:
        print("\n✅Integrity Verified: Message not tampered!")
    else:
        print("\n❌Integrity Check Failed: Message was altered!")
```

Output:

Generated AES Session Key: b'\xbb\xd7A\x84K\xee9\xb4\xa4q\xa3\n\x9b\n&1'

Encrypted Message: {'ciphertext': 'y3UeMwaBSchgQ2a8Po835ZdP25pYt8NFVv3ktCh/hFVbqJlJ/Ph65CTyjQtPxnvh3ocwG+G6cufLrA+SK7LMVVm/5hqRTVIHR4H9cx3Mkn07nDn0hhk5dsl/WfRq/A+3QimhS/GIWmQl4ZBXZBayh9QbcK33MvzdTjOTHW34Zw1ADGEGlW/IfP9N3vE2kSfMpmkRN0NOLQ==', 'nonce': 'oXAFkCWQ0akiJhvs8qO7og==', 'tag': 'OPwpqBEaBHYSGn9zxZlWkQ=='}

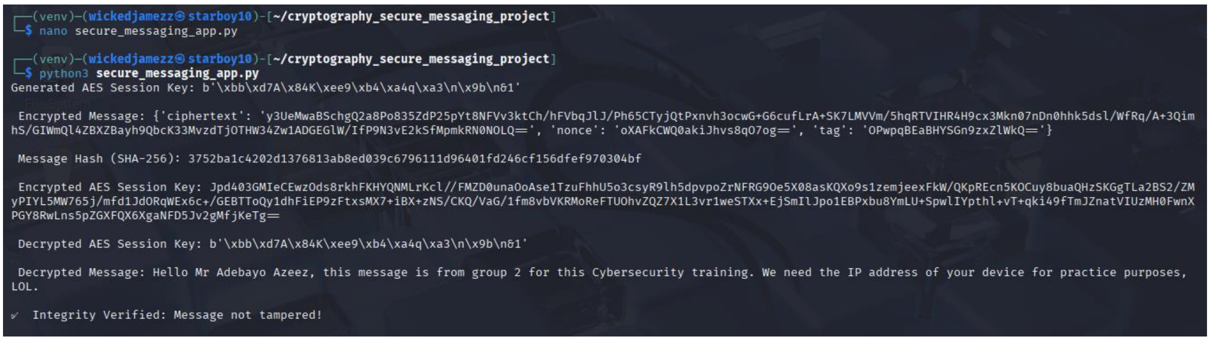
Message Hash (SHA-256): 3752ba1c4202d1376813ab8ed039c6796111d96401fd246cf156dfef970304bf

Encrypted AES Session Key:
Jpd403GMIeCEwzOds8rkhFKHYQNMLrKcl//FMZD0unaOoAse1TzuFhhU5o3csyR9lh5dpvpoZrNFRG9Oe5X08asKQXo9s1zemjeexFkW/QKpREcn5KOCuy8buaQHzSKGgTLa2BS2/ZMyPIYL5MW765j/mfd1JdORqWEx6c+/GEBTToQy1dhFiEP9zFtxsMX7+iBX+zNS/CKQ/VaG/1fm8vbVKRMoReFTUOhvZQZ7X1L3vr1weSTXx+EjSmIlJpo1EBPxbu8YmLU+SpwlIYpthl+vT+qki49fTmJZnatVIUzMH0FwnXPGY8RwLns5pZGXFQX6XgaNFD5Jv2gMfjKeTg==

Decrypted AES Session Key: b'\xbb\xd7A\x84K\xee9\xb4\xa4q\xa3\n\x9b\n&1'

Decrypted Message: Hello Mr Adebayo Azeez, this message is from group 2 for this Cybersecurity training. We need the IP address of your device for practice purposes, LOL.

✅*Integrity Verified: Message not tampered!*



5. ***"secure_file_or_folder_transfer.py"***

Simulates a secure file exchange between two clients.

- The file is encrypted using **AES**.
- The **AES** key is encrypted with RSA.
- Both sender and receiver verify file integrity using **SHA-256** hash comparison.

Code:

from Crypto.Cipher import AES, PKCS1_OAEP

from Crypto.PublicKey import RSA

from Crypto.Random import get_random_bytes

from base64 import b64encode, b64decode

```python
import hashlib


# --- AES Encryption & Decryption ---
def encrypt_file(file_path, aes_key):
    with open(file_path, 'rb') as f:
        data = f.read()

    cipher = AES.new(aes_key, AES.MODE_EAX)
    ciphertext, tag = cipher.encrypt_and_digest(data)

    encrypted_file = {
        'ciphertext': b64encode(ciphertext).decode('utf-8'),
        'nonce': b64encode(cipher.nonce).decode('utf-8'),
        'tag': b64encode(tag).decode('utf-8')
    }
    return encrypted_file

def decrypt_file(encrypted_file, aes_key, output_path):
    nonce = b64decode(encrypted_file['nonce'])
    tag = b64decode(encrypted_file['tag'])
    ciphertext = b64decode(encrypted_file['ciphertext'])
    cipher = AES.new(aes_key, AES.MODE_EAX, nonce=nonce)
    data = cipher.decrypt_and_verify(ciphertext, tag)
with open(output_path, 'wb') as f:
        f.write(data)
```

```
# --- RSA Encryption & Decryption ---
def encrypt_session_key(session_key, public_key_path="public.pem"):
    with open(public_key_path, "rb") as pub_file:
        recipient_key = RSA.import_key(pub_file.read())
        cipher_rsa = PKCS1_OAEP.new(recipient_key)
    encrypted_key = cipher_rsa.encrypt(session_key)
    return b64encode(encrypted_key).decode('utf-8')


def decrypt_session_key(encrypted_session_key_b64, private_key_path="private.pem"):
    with open(private_key_path, "rb") as priv_file:
        private_key = RSA.import_key(priv_file.read())
        cipher_rsa = PKCS1_OAEP.new(private_key)
    encrypted_key = b64decode(encrypted_session_key_b64)
    session_key = cipher_rsa.decrypt(encrypted_key)
    return session_key


# --- SHA-256 Hashing ---
def calculate_sha256(file_path):
    sha256 = hashlib.sha256()
    with open(file_path, 'rb') as f:
        for chunk in iter(lambda: f.read(4096), b""):
            sha256.update(chunk)
    return sha256.hexdigest()
# --- Demonstration ---
```

```python
if __name__ == "__main__":
    # Create a sample file to send
    test_file = "message.txt"
    with open(test_file, 'w') as f:
        f.write("This is a top-secret document for GetBundi DLC being securely transferred!")

    # Sender generates AES key
    aes_key = get_random_bytes(16)

    # Sender encrypts file and hashes it
    encrypted_file = encrypt_file(test_file, aes_key)
    hash_before = calculate_sha256(test_file)

    # Sender encrypts AES key with receiver's RSA public key
    encrypted_aes_key = encrypt_session_key(aes_key)

    # Receiver decrypts AES key
    decrypted_aes_key = decrypt_session_key(encrypted_aes_key)

    # Receiver decrypts file
    output_file = "received_message.txt"
    decrypt_file(encrypted_file, decrypted_aes_key, output_file)

    # Receiver verifies file integrity
    hash_after = calculate_sha256(output_file)
```

```
print("\n File Transfer Summary:")
print("Original Hash:", hash_before)
print("Received Hash:", hash_after)
if hash_before == hash_after:
    print("✅File integrity verified successfully!")
else:
    print("❌Integrity check failed!")
```

Output:

File Transfer Summary:

Original Hash: ef78dc366e263059ea1a85d9cefbc96654c27e9db6e0cf9876d011bcad58ec20

Received Hash: ef78dc366e263059ea1a85d9cefbc96654c27e9db6e0cf9876d011bcad58ec20

✅File integrity verified successfully!

How the System Works (Step-by-Step)

- **Sender Side**

A random AES key (session key) is generated. The sender encrypts the message or file using AES encryption. A SHA-256 hash of the original message/file is created for integrity verification. The AES session key is encrypted using the receiver's RSA public key.

- **Receiver Side**

The receiver decrypts the AES session key using their private RSA key. The encrypted message or file is decrypted using AES. The receiver recalculates the SHA-256 hash of the received message/file. The two hashes are compared to verify integrity.

Key features

- AES encryption ensures **data confidentiality**
- RSA encryption ensures **secure key exchange**
- SHA-256 hashing ensures **data integrity**
- File transfer simulation demonstrates **real-world application**

Real-world Relevance

This hybrid encryption system mirrors the structure used in:

- **WhatsApp's end-to-end encryption**
- **Secure banking and cloud storage systems**
- **Digital signature and secure file transmission protocols**

Conclusion

The Secure communication system successfully meets all project requirements:

- It uses **AES** for symmetric encryption
- It used **RSA** for asymmetric encryption of the **AES** session key
- It uses **SHA-256** for message integrity
- Lastly, it demonstrates secure file transfer between two clients.

This ensures that data remains confidential, authentic and tamper-proof during transmission.