S6 BTech CSE(AIE)

22AIE314 Computer Security

Programming Assignment

Name: **Anuvind M P**

Roll no. : **AM.EN.U4AIE22010**

Q) End-to-End Secure Messaging System using Diffie-Hellman, RSA, and Digital Signatures [CO3]

1. **KEY GENERATION**

RSA Key Generation:

```
import rsa

def generate_keys():
    public_key, private_key = rsa.newkeys(2048)
    return public_key, private_key

alice_pub, alice_priv = generate_keys()
bob_pub, bob_priv = generate_keys()
```

Purpose:

- RSA generates a public-private key pair for secure communication.
- Public key → used to encrypt data.
- Private key → used to decrypt and sign data.
- Enables encryption and digital signing.

Implementation:

- Both Alice and Bob generate 2048-bit RSA key pairs.
- Public keys are exchanged securely.
- Private keys are stored locally.

Diffie-Hellman Key Exchange:

```
import random

p = 37  # agreed large prime
g = 13  # primitive root

a = random.randint(1, p)
A = pow(g, a, p)  # Alice public

b = random.randint(1, p)
B = pow(g, b, p)  # Bob public

shared_key_alice = pow(B, a, p)
shared_key_bob = pow(A, b, p)
```

Purpose:

- Allows both users to derive the same shared secret without directly sending it.
- Used to derive a symmetric key for AES.

Implementation:

- Agreed constants p and g.
- Alice and Bob each generate a secret and a corresponding public value.
- Shared key is calculated using the other's public value.

2. SECURE MESSAGE SENDING (ALICE TO BOB)

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
import hashlib

# Symmetric key from DH
aes_key = hashlib.sha256(str(shared_key_alice).encode()).digest()

message = b"This is a secure message."
signature = rsa.sign(message, alice_priv, 'SHA-256')

cipher = AES.new(aes_key, AES.MODE_CBC)
ciphertext = cipher.encrypt(pad(message, AES.block_size))
iv = cipher.iv

payload = iv + ciphertext
final_data = rsa.encrypt(payload, bob_pub)
```

```
Message Verified and Authentic!
Decrypted Message: This is a secure message.
```

Steps:

- Alice signs the message using her RSA private key.
- Uses AES with shared DH key to encrypt the message.
- AES-encrypted message and IV are encrypted with Bob’s RSA public key.

3. SECURE MESSAGE RECEIVING (BOB)

```
from Crypto.Util.Padding import unpad
decrypted = rsa.decrypt(final_data, bob_priv)
iv, ciphertext = decrypted[:16], decrypted[16:]
```

```
cipher = AES.new(aes_key, AES.MODE_CBC, iv)
message_received = unpad(cipher.decrypt(ciphertext), AES.block_size)

try:
    rsa.verify(message_received, signature, alice_pub)
    print("Message Verified and Authentic!")
except rsa.VerificationError:
    print("Verification Failed!")
```

Explanation:

- Bob decrypts the AES payload using RSA.
- Decrypts the original message using the shared AES key.
- Verifies the signature using Alice’s public key.
- If signature is valid → message is original and untampered.

4. SYMMETRIC ENCRYPTION (AES)

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

aes_key = hashlib.sha256(str(shared_key_alice).encode()).digest()
cipher = AES.new(aes_key, AES.MODE_CBC)
ciphertext = cipher.encrypt(pad(message, AES.block_size))
```

1. Key Setup
    - Hash it to make a 256-bit AES key
2. Encryption
    - Encrypt message with AES
    - Add random IV for security
3. Secure Transfer
    - Encrypt (AES data + IV) with Bob's RSA public key
    - Attach Alice's digital signature

Why AES?

- AES is efficient and fast for encrypting data.
- The shared DH key is hashed into a 256-bit AES key.
- CBC mode ensures message randomness with IV.

```
Message is authentic and verified.
Decrypted Message: Hello Bob, this is Alice!
```