TERMINAL

Subject: Information Security Lab

Submitted To: Mam Ambreen Gul

Submitted By: Aaliya Ansari

Registration Number: Sp24-Bse-016

Date: 15/12/2025

Question 1:

Explanation:

Ssystem uses **ECC (Elliptic Curve Cryptography)** for **encrypting emails** and **DSA (Digital Signature Algorithm)** for **signing emails**. This ensures:

1. **Confidentiality:** Only the intended recipient can read the email.
2. **Integrity:** The email has not been changed.
3. **Authenticity:** The sender is verified.

Main Functions

1. generate_ecc_keys()

- Generates a **private and public ECC key pair** using the P-256 curve.
- **Private key:** Used to decrypt messages.
- **Public key:** Shared with others to encrypt messages sent to you.

2. ecc_encrypt(message, receiver_public_key, sender_private_key)

- Encrypts a message using the **receiver's ECC public key**.
- Uses ECC to generate a **shared secret**, then **AES GCM** encrypts the message.
- Returns the **nonce**, **ciphertext**, and **authentication tag**.

3. ecc_decrypt(nonce, ciphertext, tag, receiver_private_key, sender_public_key)

- Decrypts the message using the **receiver's ECC private key**.
- Recreates the **shared secret**, then uses AES to decrypt.
- Returns the **original plaintext email**.

4. generate_dsa_keys()

- Generates a **DSA key pair** for signing emails.
- **Private key:** Used to create a digital signature.
- **Public key:** Used by recipients to verify the signature.

5. sign_email(message, dsa_private_key)

- Signs an email message using the DSA **private key**.
- Ensures **authenticity** and **integrity** of the message.

6. verify_email(message, signature, dsa_public_key)

- Verifies a message's signature using the **DSA public key**.
- Returns True if the message is valid and unmodified; False if tampered.

main() Function Flow

1. Generate ECC keys for encryption and DSA keys for signing.
2. Compose an email message.
3. **Encrypt** the email using ECC.
4. **Sign** the email using DSA.
5. **Decrypt** the email to check confidentiality.
6. **Verify the signature** to confirm integrity and authenticity.

Summary :

- ECC → hides the message (confidential).
- DSA → proves who sent it and that it wasn't changed.
- AES → actually encrypts the email efficiently using a key derived from ECC.

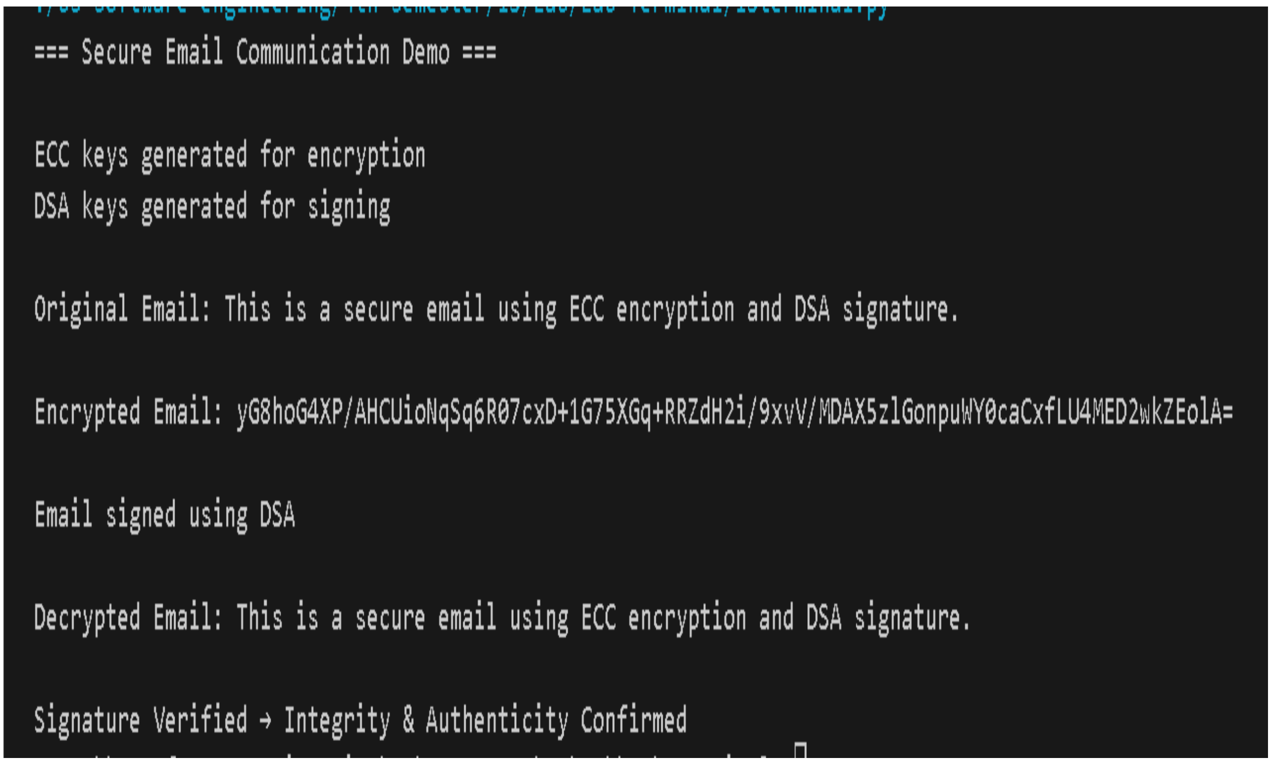
Code:

```python
# ===============================
# Secure Email System: ECC + DSA
# ===============================

from Crypto.PublicKey import ECC, DSA
from Crypto.Signature import DSS
from Crypto.Hash import SHA1
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Protocol.KDF import HKDF
import base64

# ==============================================
# 1️⃣ECC KEY GENERATION (For Encryption/Decryption)
# ==============================================

def generate_ecc_keys():
    """
    Generate ECC key pair using P-256 curve.
    Private key: Used to decrypt messages
    Public key: Used to encrypt messages
    """
    private_key = ECC.generate(curve='P-256')
    public_key = private_key.public_key()
    return private_key, public_key

# ===============================================
# 2️⃣ECC ENCRYPTION & DECRYPTION (Hybrid ECC + AES)
# ===============================================

def ecc_encrypt(message, receiver_public_key, sender_private_key):
    """
    Encrypt message using receiver's ECC public key
    - Uses ECC to derive shared secret
    - AES GCM for actual encryption
    """
    shared_point = receiver_public_key.pointQ * sender_private_key.d
    shared_secret = int(shared_point.x).to_bytes(32, 'big')
    aes_key = HKDF(shared_secret, 32, b'', SHA1)

    cipher = AES.new(aes_key, AES.MODE_GCM)
    ciphertext, tag = cipher.encrypt_and_digest(message.encode())
    return cipher.nonce, ciphertext, tag

def ecc_decrypt(nonce, ciphertext, tag, receiver_private_key,
sender_public_key):
    """
    Decrypt message using ECC private key
    """
    shared_point = sender_public_key.pointQ * receiver_private_key.d
    shared_secret = int(shared_point.x).to_bytes(32, 'big')
    aes_key = HKDF(shared_secret, 32, b'', SHA1)

    cipher = AES.new(aes_key, AES.MODE_GCM, nonce=nonce)
    plaintext = cipher.decrypt_and_verify(ciphertext, tag)
    return plaintext.decode()

# ===============================================
# 3️⃣DSA KEY GENERATION (For Signing/Verification)
# ===============================================

def generate_dsa_keys():
    """
    Generate DSA key pair for signing emails
    """
    private_key = DSA.generate(2048)
    public_key = private_key.publickey()
```

```python
    return private_key, public_key

# ================================================
# 4️⃣ SIGNING & VERIFYING EMAIL (DSA)
# ================================================

def sign_email(message, dsa_private_key):
    """
    Sign the email message using DSA private key
    """
    hash_obj = SHA1.new(message.encode())
    signer = DSS.new(dsa_private_key, 'fips-186-3')
    signature = signer.sign(hash_obj)
    return signature

def verify_email(message, signature, dsa_public_key):
    """
    Verify email signature using DSA public key
    """
    hash_obj = SHA1.new(message.encode())
    verifier = DSS.new(dsa_public_key, 'fips-186-3')
    try:
        verifier.verify(hash_obj, signature)
        return True
    except ValueError:
        return False

# ================================================
# 5️⃣ MAIN FUNCTION: DEMO SECURE EMAIL
# ================================================

def main():
    print("=== Secure Email Communication Demo ===\n")

    # Step 1: Generate Keys
    ecc_private, ecc_public = generate_ecc_keys()
    dsa_private, dsa_public = generate_dsa_keys()
    print("ECC keys generated for encryption")
    print("DSA keys generated for signing\n")

    # Step 2: Compose email
    email_message = "This is a secure email using ECC encryption and DSA
signature."
    print("Original Email:", email_message, "\n")

    # Step 3: Encrypt email
    nonce, ciphertext, tag = ecc_encrypt(email_message, ecc_public,
ecc_private)
```

```python
    print("Encrypted Email:", base64.b64encode(ciphertext).decode(), "\n")

    # Step 4: Sign email
    signature = sign_email(email_message, dsa_private)
    print("Email signed using DSA\n")

    # Step 5: Decrypt email
    decrypted_message = ecc_decrypt(nonce, ciphertext, tag, ecc_private,
ecc_public)
    print("Decrypted Email:", decrypted_message, "\n")

    # Step 6: Verify signature
    if verify_email(decrypted_message, signature, dsa_public):
        print("Signature Verified → Integrity & Authenticity Confirmed")
    else:
        print("Signature Invalid → Message may be tampered")

# Run the program
if __name__ == "__main__":
    main()
```

Output:



```
=== Secure Email Communication Demo ===


ECC keys generated for encryption
DSA keys generated for signing


Original Email: This is a secure email using ECC encryption and DSA signature.


Encrypted Email: yG8hoG4XP/AHCUioNqSq6R07cxD+1G75XGq+RRZdH2i/9xvV/MDAX5zlGonpuWY0caCxfLU4MED2wkZEolA=


Email signed using DSA


Decrypted Email: This is a secure email using ECC encryption and DSA signature.


Signature Verified → Integrity & Authenticity Confirmed
```

Question 2:

1. Justify your security method: Why is it suitable or better than other possible methods for your type of project?

Solution:

Elliptic Curve Cryptography (ECC) is used for encryption:

- ECC provides **high security with smaller key sizes** (e.g., 256-bit ECC ≈ 3072-bit RSA).
- Smaller keys mean **faster encryption/decryption** and **less computational load**, which is ideal for email systems, especially on smartphones or low-power devices.

Digital Signature Algorithm (DSA) is used for signing:

- Ensures **message integrity** and **authenticity**, so recipients can verify that the message was not altered and was sent by the legitimate sender.

Why better than other methods:

- Compared to RSA: ECC is faster and more efficient for mobile/low-resource environments.
- Compared to symmetric-only methods (AES alone): ECC + DSA ensures both **confidentiality** and **authenticity**, not just secrecy.

Conclusion:

The combination of ECC (encryption) + DSA (signing) provides a **balanced and efficient solution** for secure email communication, protecting confidentiality, integrity, and authenticity.

2. Identify one possible vulnerability or weakness in your current system. How could an attacker misuse it?

Answer:

- **Vulnerability:** If the **private keys (ECC or DSA)** are stored insecurely on the sender or receiver device, an attacker could steal them.
- How an attacker could misuse it:
  - **ECC private key theft** → Attacker can decrypt all emails intended for the victim.
  - **DSA private key theft** → Attacker can forge signatures, sending fake emails that appear legitimate.

3. **Suggest one realistic improvement to enhance the security of your project. Briefly explain how it would work**

Answer:

Improvement: Use secure key storage with encryption, such as:

- Encrypt private keys with a **strong passphrase** before saving.
- Use OS-provided secure storage (e.g., **Windows DPAPI**, macOS Keychain, Linux Keyrings).

How it works:

- Even if an attacker gets access to the stored file, without the passphrase or secure storage access, they **cannot use the private key**.
- This significantly reduces the risk of key theft and enhances overall system security.