**Elliptic Curve Cryptography**

**CSE 459: Cryptography & Network Security**

Submitted by

Name: D. Tejaswi

Roll No: AP22110011503

Section: CSE Y

Lab Date: 17/03/2025

Submission Date: 23/03/2025

**A picture containing text

Description automatically generated**

**Department Computer Science and Engineering**

**School of Engineering and Sciences**

**SRM University–AP**

**Amaravati, Andhra Pradesh – 522 240, India**

1. **Question**

**1) Write a program to perform a key exchange using ECC.**

**2) Write a program to encrypt and decrypt the message “Hell0 SRM AP” using Elliptic Curve**

**Cryptography and AES.**

**Use two different curve and report the changes.**

1. **Algorithm Description**
2. **Solution**

**1) Write a program to perform a key exchange using ECC.**

from cryptography.hazmat.primitives.asymmetric import ec

# Use the same curve for both parties (e.g., SECP256R1)

curve = ec.SECP256R1()

# Generate private keys for two parties

private\_key\_A = ec.generate\_private\_key(curve)

private\_key\_B = ec.generate\_private\_key(curve)

# Generate public keys

public\_key\_A = private\_key\_A.public\_key()

public\_key\_B = private\_key\_B.public\_key()

# Exchange keys and derive shared secret

shared\_secret\_A = private\_key\_A.exchange(ec.ECDH(), public\_key\_B)

shared\_secret\_B = private\_key\_B.exchange(ec.ECDH(), public\_key\_A)

# Ensure both shared secrets are the same

assert shared\_secret\_A == shared\_secret\_B, "Key exchange failed!"

print("Shared Secret:", shared\_secret\_A.hex())

**output:**

Shared Secret: 8cf7b5ac948c7a26fc15a0c55a5aa80c91fbe1d1c6e52951358c348ef6b69181

**2) Write a program to encrypt and decrypt the message “Hell0 SRM AP” using Elliptic Curve Cryptography and AES.**

from cryptography.hazmat.primitives.asymmetric import ec

from cryptography.hazmat.primitives.kdf.hkdf import HKDF

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

import os

# Message to encrypt

message = b"Hell0 SRM AP"

# Generate ECC key pair

private\_key = ec.generate\_private\_key(ec.SECP256R1())

public\_key = private\_key.public\_key()

# Encrypt function

def encrypt\_message(public\_key, message):

    ephemeral\_key = ec.generate\_private\_key(ec.SECP256R1())

    shared\_secret = ephemeral\_key.exchange(ec.ECDH(), public\_key)

    # Derive AES key using HKDF

    aes\_key = HKDF(

        algorithm=hashes.SHA256(),

        length=32,

        salt=None,

        info=b"encryption",

    ).derive(shared\_secret)

    iv = os.urandom(16)

    cipher = Cipher(algorithms.AES(aes\_key), modes.CBC(iv))

    encryptor = cipher.encryptor()

    padded\_message = message + b' ' \* (16 - len(message) % 16)  # Padding

    ciphertext = encryptor.update(padded\_message) + encryptor.finalize()

    return ephemeral\_key.public\_key(), iv, ciphertext

# Decrypt function

def decrypt\_message(private\_key, ephemeral\_public\_key, iv, ciphertext):

    shared\_secret = private\_key.exchange(ec.ECDH(), ephemeral\_public\_key)

    aes\_key = HKDF(

        algorithm=hashes.SHA256(),

        length=32,

        salt=None,

        info=b"encryption",

    ).derive(shared\_secret)

    cipher = Cipher(algorithms.AES(aes\_key), modes.CBC(iv))

    decryptor = cipher.decryptor()

    decrypted\_padded = decryptor.update(ciphertext) + decryptor.finalize()

    return decrypted\_padded.strip()

# Encrypt message

ephemeral\_pub, iv, ciphertext = encrypt\_message(public\_key, message)

print("Ciphertext:", ciphertext.hex())

# Decrypt message

decrypted\_message = decrypt\_message(private\_key, ephemeral\_pub, iv, ciphertext)

print("Decrypted Message:", decrypted\_message.decode())

**output:**

Ciphertext: e5524228a3ffbe4e134baed3c222858b

Decrypted Message: Hell0 SRM AP

1. **Code Repository:**