EXP 11: ELLIPTIC CURVE CRYPTOGRAPHY

PROGRAM:
```
# Simple ECC Key Exchange without any external libraries (not secure, educational only)

# Elliptic Curve: y^2 = x^3 + ax + b over finite field F_p
a = 2
b = 3
p = 97  # Prime number for finite field
G = (3, 6)  # Generator point
n = 5  # Private key space (very small for demo)

# Modular inverse
def inverse_mod(k, p):
    return pow(k, -1, p)

# Point addition
def point_add(P, Q):
    if P == (None, None):
        return Q
    if Q == (None, None):
        return P

    if P == Q:
        # Point doubling
        l = (3 * P[0] * P[0] + a) * inverse_mod(2 * P[1], p) % p
    else:
        # Point addition
        l = (Q[1] - P[1]) * inverse_mod(Q[0] - P[0], p) % p

    x = (l * l - P[0] - Q[0]) % p
    y = (l * (P[0] - x) - P[1]) % p
    return (x, y)

# Scalar multiplication (repeated addition)
def scalar_mult(k, point):
    R = (None, None)  # Point at infinity
    for _ in range(k):
        R = point_add(R, point)
    return R

# Alice's keys
alice_private = 2
alice_public = scalar_mult(alice_private, G)

# Bob's keys
bob_private = 3
bob_public = scalar_mult(bob_private, G)

# Shared secret
alice_secret = scalar_mult(alice_private, bob_public)
bob_secret = scalar_mult(bob_private, alice_public)

# Output
print("Curve: y^2 = x^3 + {}x + {} over F_{}".format(a, b, p))
print("\n=== Alice ===")
print("Private key:", alice_private)
print("Public key:", alice_public)
```

```
print("\n=== Bob ===")
print("Private key:", bob_private)
print("Public key:", bob_public)

print("\n=== Shared Secret ===")
print("Alice computes:", alice_secret)
print("Bob computes:  ", bob_secret)
print("Match:", alice_secret == bob_secret)
```

OUTPUT:

Output

```
Curve: y^2 = x^3 + 2x + 3 over F_97

=== Alice ===
Private key: 2
Public key: (80, 10)

=== Bob ===
Private key: 3
Public key: (80, 87)

=== Shared Secret ===
Alice computes: (3, 6)
Bob computes:   (3, 6)
Match: True

=== Code Execution Successful ===
```