```
# RSA & Elgamal Signature Scheme

userinput = input("Welcome to the RSA & Elgamal Signature Scheme!"
                  "\n-----------------------------------------"
                  "\nWould you like RSA or Elgamal? ")

print("You Choose To Use ", userinput)


# RSA Scheme
# Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep  5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on
win32
if userinput.upper() == 'RSA Scheme' or userinput.upper() == 'RSA' or userinput.upper() ==
'R':
    P = int(input("-------------------------"
                  "\nWhat is Alice's P key?"
                  "\n(Default = 17):") or "17")
    Q = int(input("-------------------------"
                  "\nWhat is Alice's Q Key?"
                  "\n(Default = 47):") or "47")
    E = int(input("---------------------------------"
                  "\nWhat is the E (Chosen by Alice)?"
                  "\n(Default = 11):") or "11")
    M = int(input("-------------------------------"
                  "\nWhat is the message (m =?)?"
                  "\n(Default = 6):") or "6")

    # Calculation for N = P * Q
    N = P * Q
    print("-----------------"
          "\nN = P * Q"
          "\n"
          "N =", N)

    # Calculation for 0(n)
    O = (P - 1) * (Q - 1)
    print("-----------------"
          "\n\u03C6(N)"
          "=", O)


    # Calculation for D = e^-1 mod 0(n)
    def gcd(A, B):
        if A == 0:
            return (B, 0, 1)
        G, Y, X = gcd(B % A, A)
        return (G, X - (B // A) * Y, Y)


    def inv(A, M):
        G, X, Y = gcd(A, M)
        if G != 1:
            raise Exception('No modular inverse')
        return X % M


    D = inv(E, O)
    print("-----------------"
          "\nD = E^-1 mod(\u03C6)(N)"
          "\nD = ", D)

    # Return Public Key
    print("-----------------------------"
          "\nThe Public key is", (E, N))

    # Return Private Key
    print("----------------------------"
```

```
          "\nThe Private key is", (D, N))

    # Compute Signatures
    S = (M ** D) % N
    print("-----------------"
          "\nThe Signature is "
          "S = ", S,)

    # Returns (M,S)
    print("-----------------"
          "\nReturns (M,S)"
          , (M, S),)

    # Compute S^E Mod N
    Verify = (S ** E) % N
    print("-----------------"
          "\nS^E Mod "
          "\nN = ", Verify,
          "\n---------------")

    # Verify M matches Verify
    Verify2 = M == Verify
    if Verify2 != 'True':
        print("M is Verifyed "
              "(M = Verify) ")
    else:
        print("M is not Verified "
              "(M dosent equal verify)")




# Elgamal Signature Scheme
# Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep  5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on
win32
elif userinput.upper() == 'Elgamal Signature Scheme' or userinput.upper() == 'Elgamal' or
userinput.upper() == 'E':
    P = int(input("-------------------------"
                  "\nWhat is Alice's P key?"
                  "\n(Default = 17): ") or "17")
    A = int(input("-------------------------"
                  "\nWhat is Alice's Primitive Element (a = ?)?"
                  "\n(Default = 7): ") or "7")
    D = int(input("-------------------------"
                  "\nWhat is the Random Integer (D = ?)?"
                  "\n(Default = 6): ") or "6")
    KE = int(input("-------------------------"
                   "\nWhat is the Ephemeral key (Ke = ?)?"
                  "\n(Default = 7): ") or "7")
    M = int(input("-------------------------"
                  "\nWhat is the message (M = ?)"
                  "\n(Default = 6): ") or "6")

    # Compute B = A^D mod P
    B = (A ** D) % P
    print("-------------------------"
          "\nB = ", B)

    # Public Key
    print("-------------------------"
          "\nThe Public Key is", (P, A, B))

    # Private Key
    print("-------------------------"
          "\nThe Private Key is", D)

    # Alices Public Key
```

```
    print("-------------------------"
          "\nAlices Public Key is", (P, A, B))

    # Compute Signatures
    R = (A ** KE) % P
    print("-------------------------"
          "\nR = ", R)

    # Compute Signatures
    C = (P - 1)
    L = M - (D * R)
    S = L * KE % C
    print("-------------------------"
          "\nS = ", S)

    # Return (M, (R,S))
    print("-------------------------"
          "\n(M, (R,S)) =", (M, (R, S)))

    # Compute T = B^R * R^S mod p
    T = (B ** R) * (R ** S) % P
    print("-------------------------"
          "\nT = ", T,
          "\n-------------------------")

    # Verify If T Matches Verify
    Verify = (A ** M) % P
    Verify2 = Verify == T
    if Verify2 != 'True':
        print("T is Verifyed "
              "(T = Verify) ")
    else:
        print("T is not Verified "
              "(T dosent equal verify)")

else:
    print("That is not a valid input, "
          "Please try again!! ")
```