Name:-Amrit Ranjan

Reg No:-21BCE1847

Cryptography And Network Security

Assignment 5

Faculty Name:-Dr Tapabarta Roy

SHA 256 Algorithm



Ques:- Take a binary string of your choice of length 1000 and apply SHA 256 on it and determine the output.

Both thing is pre standard library so no need to install anything extra

Code:-

def sha256(input\_string):

    bytes = ""

    h = [0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a, 0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19]

    h0 = h.copy()  # Create a copy of h

    k = [0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,

         0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,

         0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,

         0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,

         0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,

         0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,

         0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,

         0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2]

    # Pre-processing (Padding):

    L = len(input\_string) \* 8  # bit length of input\_string

    input\_string = input\_string + chr(0b10000000)

    input\_string = input\_string + chr(0) \* ((56 - len(input\_string) % 64) % 64)

    input\_string = input\_string + chr(L >> 56 & 0xFF) + chr(L >> 48 & 0xFF) + chr(L >> 40 & 0xFF) + chr(L >> 32 & 0xFF) + chr(L >> 24 & 0xFF) + chr(L >> 16 & 0xFF) + chr(L >> 8 & 0xFF) + chr(L & 0xFF)

    # Process the message in successive 512-bit chunks:

    for i in range(0, len(input\_string), 64):

        w = [0] \* 64

        for j in range(16):

            w[j] = ord(input\_string[i + j \* 4]) << 24 | ord(input\_string[i + j \* 4 + 1]) << 16 | ord(input\_string[i + j \* 4 + 2]) << 8 | ord(input\_string[i + j \* 4 + 3])

        for j in range(16, 64):

            s0 = (w[j - 15] >> 7 | w[j - 15] << 25) ^ (w[j - 15] >> 18 | w[j - 15] << 14) ^ (w[j - 15] >> 3)

            s1 = (w[j - 2] >> 17 | w[j - 2] << 15) ^ (w[j - 2] >> 19 | w[j - 2] << 13) ^ (w[j - 2] >> 10)

            w[j] = (w[j - 16] + s0 + w[j - 7] + s1) & 0xFFFFFFFF

        a, b, c, d, e, f, g, h = h0

        # Main loop:

        for i in range(64):

            S1 = (e >> 6 | e << 26) ^ (e >> 11 | e << 21) ^ (e >> 25 | e << 7)

            ch = (e & f) ^ (~e & g)

            temp1 = (h + S1 + ch + k[i] + w[i]) & 0xFFFFFFFF

            S0 = (a >> 2 | a << 30) ^ (a >> 13 | a << 19) ^ (a >> 22 | a << 10)

            maj = (a & b) ^ (a & c) ^ (b & c)

            temp2 = (S0 + maj) & 0xFFFFFFFF

            h = g

            g = f

            f = e

            e = (d + temp1) & 0xFFFFFFFF

            d = c

            c = b

            b = a

            a = (temp1 + temp2) & 0xFFFFFFFF

        # Add this chunk's hash to result so far:

        h0 = [(x + y) & 0xFFFFFFFF for x, y in zip(h0, [a, b, c, d, e, f, g, h])]

    # Produce the final hash value (big-endian):

    return ''.join([('%08x' % i) for i in h0])

# Test the function with a binary string of length 1000

print(sha256('1' \* 125))

Output ScreenShots:-



