Cryptography & Network Security Lab

PRN/ Roll No: 2019BTECS00090

Full name: Udaykumar Gadikar

Assignment No. 13

Title: SHA-512(Secured Hash Algorithm) Algorithm

Aim: To Demonstrate SHA-512 Algorithm

Theory:

SHA (Secure Hash Algorithm) is a set of cryptographic hash functions, they are built using the Merkle-Damgård construction, from a one-way compression function itself. They are built using the Davies–Meyer structure from a specialized block cipher.

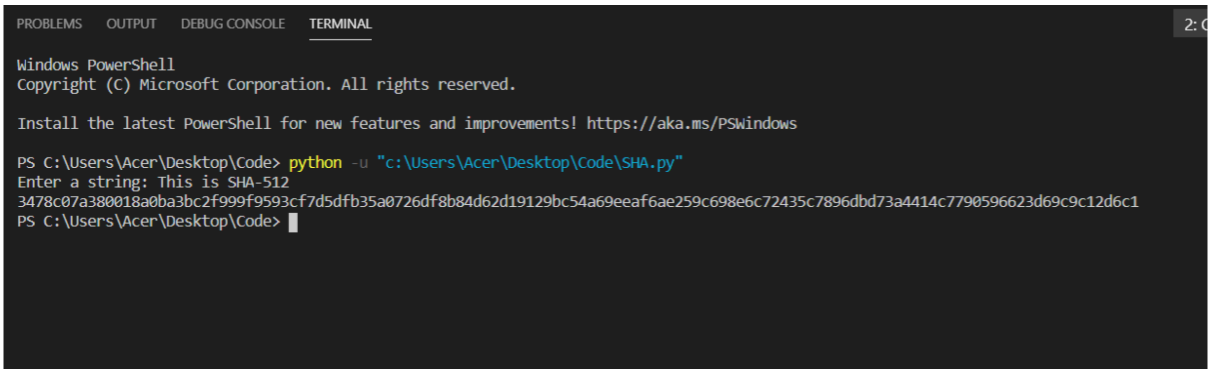
The SHA family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits.

Code:

```
import binascii
import struct

initial_hash = (
    0x6a09e667f3bcc908,
    0xbb67ae8584caa73b,
    0x3c6ef372fe94f82b,
    0xa54ff53a5f1d36f1,
    0x510e527fade682d1,
    0x9b05688c2b3e6c1f,
    0x1f83d9abfb41bd6b,
    0x5be0cd19137e2179,
)

round_constants = (
    0x428a2f98d728ae22, 0x7137449123ef65cd, 0xb5c0fbcfec4d3b2f,
    0xe9b5dba58189dbbc, 0x3956c25bf348b538, 0x59f111f1b605d019,
```

```python
    0x923f82a4af194f9b, 0xab1c5ed5da6d8118, 0xd807aa98a3030242,
    0x12835b0145706fbe, 0x243185be4ee4b28c, 0x550c7dc3d5ffb4e2,
    0x72be5d74f27b896f, 0x80deb1fe3b1696b1, 0x9bdc06a725c71235,
    0xc19bf174cf692694, 0xe49b69c19ef14ad2, 0xefbe4786384f25e3,
    0x0fc19dc68b8cd5b5, 0x240ca1cc77ac9c65, 0x2de92c6f592b0275,
    0x4a7484aa6ea6e483, 0x5cb0a9dcbd41fbd4, 0x76f988da831153b5,
    0x983e5152ee66dfab, 0xa831c66d2db43210, 0xb00327c898fb213f,
    0xbf597fc7beef0ee4, 0xc6e00bf33da88fc2, 0xd5a79147930aa725,
    0x06ca6351e003826f, 0x142929670a0e6e70, 0x27b70a8546d22ffc,
    0x2e1b21385c26c926, 0x4d2c6dfc5ac42aed, 0x53380d139d95b3df,
    0x650a73548baf63de, 0x766a0abb3c77b2a8, 0x81c2c92e47edaee6,
    0x92722c851482353b, 0xa2bfe8a14cf10364, 0xa81a664bbc423001,
    0xc24b8b70d0f89791, 0xc76c51a30654be30, 0xd192e819d6ef5218,
    0xd69906245565a910, 0xf40e35855771202a, 0x106aa07032bbd1b8,
    0x19a4c116b8d2d0c8, 0x1e376c085141ab53, 0x2748774cdf8eeb99,
    0x34b0bcb5e19b48a8, 0x391c0cb3c5c95a63, 0x4ed8aa4ae3418acb,
    0x5b9cca4f7763e373, 0x682e6ff3d6b2b8a3, 0x748f82ee5defb2fc,
    0x78a5636f43172f60, 0x84c87814a1f0ab72, 0x8cc702081a6439ec,
    0x90befffa23631e28, 0xa4506cebde82bde9, 0xbef9a3f7b2c67915,
    0xc67178f2e372532b, 0xca273eceea26619c, 0xd186b8c721c0c207,
    0xeada7dd6cde0eb1e, 0xf57d4f7fee6ed178, 0x06f067aa72176fba,
    0x0a637dc5a2c898a6, 0x113f9804bef90dae, 0x1b710b35131c471b,
    0x28db77f523047d84, 0x32caab7b40c72493, 0x3c9ebe0a15c9bebc,
    0x431d67c49c100d4c, 0x4cc5d4becb3e42b6, 0x597f299cfc657e2a,
    0x5fcb6fab3ad6faec, 0x6c44198c4a475817,
)


def _right_rotate(n: int, bits: int) -> int:
    return (n >> bits) | (n << (64 - bits)) & 0xFFFFFFFFFFFFFFFF


def sha512(message: str) -> str:
    if type(message) is not str:
        raise TypeError('Given message should be a string.')
    message_array = bytearray(message, encoding='utf-8')

    mdi = len(message_array) % 128
    padding_len = 119 - mdi if mdi < 112 else 247 - mdi
    ending = struct.pack('!Q', len(message_array) << 3)
    message_array.append(0x80)
    message_array.extend([0] * padding_len)
    message_array.extend(bytearray(ending))

    sha512_hash = list(initial_hash)
    for chunk_start in range(0, len(message_array), 128):
        chunk = message_array[chunk_start:chunk_start + 128]

        w = [0] * 80
        w[0:16] = struct.unpack('!16Q', chunk)

        for i in range(16, 80):
            s0 = (
                _right_rotate(w[i - 15], 1) ^
                _right_rotate(w[i - 15], 8) ^
                (w[i - 15] >> 7)
            )
            s1 = (
                _right_rotate(w[i - 2], 19) ^
                _right_rotate(w[i - 2], 61) ^
                (w[i - 2] >> 6)
            )
            w[i] = (w[i - 16] + s0 + w[i - 7] + s1) & 0xFFFFFFFFFFFFFFFF

        a, b, c, d, e, f, g, h = sha512_hash

        for i in range(80):
            sum1 = (
                _right_rotate(e, 14) ^
                _right_rotate(e, 18) ^
                _right_rotate(e, 41)
            )
            ch = (e & f) ^ (~e & g)
            temp1 = h + sum1 + ch + round_constants[i] + w[i]
            sum0 = (
                _right_rotate(a, 28) ^
                _right_rotate(a, 34) ^
                _right_rotate(a, 39)
            )
            maj = (a & b) ^ (a & c) ^ (b & c)
            temp2 = sum0 + maj

            h = g
            g = f
            f = e
            e = (d + temp1) & 0xFFFFFFFFFFFFFFFF
            d = c
            c = b
            b = a
            a = (temp1 + temp2) & 0xFFFFFFFFFFFFFFFF

        sha512_hash = [
            (x + y) & 0xFFFFFFFFFFFFFFFF
            for x, y in zip(sha512_hash, (a, b, c, d, e, f, g, h))
        ]

    return binascii.hexlify(
        b''.join(struct.pack('!Q', element) for element in sha512_hash),
    ).decode('utf-8')


s = input("Enter a string: ")
print(sha512(s))
```

Output:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Acer\Desktop\Code> python -u "c:\Users\Acer\Desktop\Code\SHA.py"
Enter a string: This is SHA-512
3478c07a380018a0ba3bc2f999f9593cf7d5dfb35a0726df8b84d62d19129bc54a69eeaf6ae259c698e6c72435c7896dbd73a4414c7790596623d69c9c12d6c1
PS C:\Users\Acer\Desktop\Code> 
```

Conclusion:

SHA-512, or Secure Hash Algorithm 512, is a hashing algorithm used to convert text of any length into a fixed-size string. Each output produces a SHA-512 length of 512 bits (64 bytes). This algorithm is commonly used for email addresses hashing, password hashing, and digital record verification.