**Implementation of SHA-512**

**Description**

This is a simple implementation of SHA-512 in Python.

**What is SHA-512**

SHA-512, short for Secure Hash Algorithm 512-bit, is a cryptographic hash function that takes an input message and produces a fixed-size output hash value (512 bits or 64 bytes). It is designed to be a one-way function, meaning it is computationally infeasible to reverse the process and obtain the original input from the hash value. SHA-512 is widely used for data integrity verification, password storage, digital signatures, and various security applications where ensuring data integrity and authenticity is crucial.

**Algorithm**

* Get the input message from the user.
* Encode the input message with `UTF-8`
* Add the padding bits as per necessity.
* Now, divide the message in `N \\* 1024` blocks if the length of message >= 896.
* Initialize the constant K and HASH\_VALUE
* For every block of input apply the `compression\_function()`
* The final hash value will be stored in the HASH\_VALUE constant. This value will be used as seed for hashing another message.

**SHA-512 Hashing Algorithm Documentation**

**1. Constants and Initial Hash Values**

K: This list contains constant values (64-bit words) used in the SHA-512 compression function. These values are used in the message schedule during the hashing process.

HASH\_VALUE: This list contains the initial hash values (64-bit words) used in the SHA-512 algorithm. These values are the first eight prime numbers' fractional parts (in hexadecimal representation) square roots.

**2. Utility Functions**

2.1. rotr(x, n)

Description: Performs a right circular shift operation on the 64-bit input x by n bits.

Parameters:

x: 64-bit input value.

n: Number of bits to shift.

Returns: The result of the right circular shift operation.

2.2. addition\_modulo\_2\_64(value)

Description: Performs addition modulo 264264 on the input value.

Parameters:

value: Value to perform the modulo operation on.

Returns: The result of the addition modulo 264264.

2.3. pad\_message(message)

Description: Pads the input message according to the SHA-512 padding scheme.

Parameters:

message: The input message to be padded.

Returns: The padded message in bytes.

2.4. divide\_to\_blocks(message)

Description: Divides the padded message into 128-byte (64-bit) blocks.

Parameters:

message: The padded message in bytes.

Returns: A list of 128-byte blocks.

**3. SHA-512 Compression Functions**

3.1. Ch(e, f, g)

Description: Computes the Ch(x, y, z) function used in the SHA-512 compression function.

Parameters:

e, f, g: 64-bit input values.

Returns: The result of Ch(x, y, z).

3.2. Maj(a, b, c)

Description: Computes the Maj(x, y, z) function used in the SHA-512 compression function.

Parameters:

a, b, c: 64-bit input values.

Returns: The result of Maj(x, y, z).

3.3. sigma\_0(word)

Description: Computes the sigma\_0 function used in the SHA-512 compression function.

Parameters:

word: 64-bit input word.

Returns: The result of sigma\_0 function.

3.4. sigma\_1(word)

Description: Computes the sigma\_1 function used in the SHA-512 compression function.

Parameters:

word: 64-bit input word.

Returns: The result of sigma\_1 function.

3.5. compression\_function(message)

Description: Implements the SHA-512 compression function.

Parameters:

message: A 128-byte (64-bit) block of the padded message.

**Working:**

Initializes eight working variables (a to h) with the initial hash values.

Processes the block in 80 rounds, updating the working variables using various logical and arithmetic operations.

Updates the intermediate hash values (HASH\_VALUE) after each round.

Note: The K constants are used during the rounds, and the functions Ch, Maj, sigma\_0, and sigma\_1 are applied based on the SHA-512 algorithm specifications.

**4. Main Code Flow**

Input: Accepts a message from the user encoded in UTF-8.

Padding: The input message is padded using the pad\_message function.

Block Division: The padded message is divided into 128-byte blocks using the divide\_to\_blocks function.

Processing: Each block is processed using the compression\_function, updating the HASH\_VALUE.

Output: After processing all blocks, the final SHA-512 hash value is calculated and displayed to the user.

**SHA-15 Compression Function**

for t in range(16):

W[t] = int.from\_bytes(message[t \* 8 : (t + 1) \* 8], byteorder="big")

for t in range(16, 80):

W[t] = sigma\_1(W[t - 2] + W[t - 7]) + sigma\_0(W[t - 15] + W[t - 16])

The first loop calculates W0 - W15. The second loop calculates W16 - W79.

for t in range(80):

T1 = h + (Ch(e, f, g) + (rotr(e, 14) ^ rotr(e, 18) ^ rotr(e, 41)) + K[t] + W[t])

T2 = (rotr(a, 28) ^ rotr(a, 34) ^ rotr(a, 39)) + Maj(a, b, c)

h = g

g = f

f = e

e = addition\_modulo\_2\_64(d + T1)

d = c

c = b

b = a

a = addition\_modulo\_2\_64(T1 + T2)

#intermediate Hash valuesHASH\_VALUE[0] = addition\_modulo\_2\_64(HASH\_VALUE[0] + a)HASH\_VALUE[1] = addition\_modulo\_2\_64(HASH\_VALUE[1] + b)HASH\_VALUE[2] = addition\_modulo\_2\_64(HASH\_VALUE[2] + c)HASH\_VALUE[3] = addition\_modulo\_2\_64(HASH\_VALUE[3] + d)HASH\_VALUE[4] = addition\_modulo\_2\_64(HASH\_VALUE[4] + e)HASH\_VALUE[5] = addition\_modulo\_2\_64(HASH\_VALUE[5] + f)HASH\_VALUE[6] = addition\_modulo\_2\_64(HASH\_VALUE[6] + g)HASH\_VALUE[7] = addition\_modulo\_2\_64(HASH\_VALUE[7] + h)

The compression\_function is the core of the SHA-512 hashing process. It takes a 128-byte message block as input and updates the HASH\_VALUE accordingly.

**USAGE**

* Input a message after running the program.
* The code pads the message, divides it into blocks, and initiates the SHA-512 hashing process.
* The final 512-bit (128-byte) hash value is printed as a hexadecimal string.