K = [

0x428A2F98D728AE22, 0x7137449123EF65CD, 0xB5C0FBCFEC4D3B2F, 0xE9B5DBA58189DBBC,

0x3956C25BF348B538, 0x59F111F1B605D019, 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,

]

* These are constants used in the SHA-512 algorithm.

HASH\_VALUE = [

0x6A09E667F3BCC908, 0xBB67AE8584CAA73B, 0x3C6EF372FE94F82B, 0xA54FF53A5F1D36F1,

0x510E527FADE682D1, 0x9B05688C2B3E6C1F, 0x1F83D9ABFB41BD6B, 0x5BE0CD19137E2179,

]

* These are the initial hash values for SHA-512.

def Ch(e, f, g):

return (e & f) ^ (~e & g)

* The Ch function is a bitwise operation used in the SHA-512 compression function. It operates on three 64-bit inputs (e, f, and g) and returns a 64-bit result.

def Maj(a, b, c):

return (a & b) ^ (a & c) ^ (b & c)

* The Maj function is another bitwise operation used in the SHA-512 compression function. It operates on three 64-bit inputs (a, b, and c) and returns a 64-bit result.

def rotr(x, n):

return (x >> n) | (x << (64 - n))

* The rotr() (right rotate) function performs a bitwise right rotation of a 64-bit value "x" by "n" bits.

def summation\_a(a):

return rotr(a, 28) ^ rotr(a, 34) ^ rotr(a, 39)

* The summation\_a() function combines right rotations and XOR operations on a 64-bit input "a" and returns a 64-bit result.

def summation\_e(e):

return rotr(e, 14) ^ rotr(e, 18) ^ rotr(e, 41)

* The summation\_e() function combines right rotations and XOR operations on a 64-bit input "e" and returns a 64-bit result.

def sigma\_0(word):

return rotr(word, 1) ^ rotr(word, 8) ^ (word >> 7)

* The sigma\_0() function calculates a value based on right rotations and XOR operations on a 64-bit word.

def sigma\_1(word):

return rotr(word, 19) ^ rotr(word, 61) ^ (word >> 6)

* The sigma\_1 function calculates another value based on right rotations and XOR operations on a 64-bit word.

def addition\_modulo\_2\_64(value):

return value % (2\*\*64)

* The addition\_modulo\_2\_64() function performs modular addition of a 64-bit value. It makes sure that the value doesn't exceed the length of 64.

def pad\_message(message):

message += b"\x80" # Adding 1 byte (10000000)

while len(message) % 128 != 112:

message += b"\x00"

message += (len(message) \* 8).to\_bytes(16, "big")

return message

* The pad\_message() function adds padding to the input message to make its length a multiple of 128 bytes. It also appends the message length in bits at the end.

def divide\_to\_blocks(message):

blocks = []

for i in range(0, len(message), 128):

blocks.append(message[i : i + 128])

return blocks

* The divide\_to\_blocks function divides a padded message into 128-byte blocks, which are processed by the SHA-512 compression function.

def compression\_function(message):

a, b, c, d, e, f, g, h = HASH\_VALUE

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])

# Compression Loop

for t in range(80):

T1 = h + (Ch(e, f, g) + summation\_e(e) + K[t] + W[t])

T2 = summation\_a(a) + 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 values

HASH\_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 first loop calculates W0 - W15. The second loop calculates W16 - W79.
* 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.