# SHA-512 Documentation

# XOR Function

def X\_OR(operand1, operand2):

"""

Perform bitwise XOR between two binary operands of the same length.

Args:

operand1 (str): Binary string representing the first operand.

operand2 (str): Binary string representing the second operand.

Returns:

str: Binary string result of the XOR operation.

"""

# Implementation details...

# Right Rotation Function

def ROTR(RotLength, operand):

"""

Right-rotate a binary string by a specified length.

Args:

RotLength (int): Length of the rotation.

operand (str): Binary string to be rotated.

Returns:

str: Resulting binary string after rotation.

"""

# Implementation details...

# Logical Shift Right Function

def SHR(shift\_Length, operand):

"""

Perform a logical shift right operation on a binary string by a specified length.

Args:

shift\_Length (int): Length of the logical shift.

operand (str): Binary string to be shifted.

Returns:

str: Resulting binary string after the logical shift right operation.

"""

# Implementation details...

# Sigma0 Function

def sigma0(operand):

"""

Apply the sigma0 function to a 64-bit binary string.

Args:

operand (str): Binary string input.

Returns:

str: Resulting binary string after applying the sigma0 function.

"""

# Implementation details...

# Sigma1 Function

def sigma1(operand):

"""

Apply the sigma1 function to a 64-bit binary string.

Args:

operand (str): Binary string input.

Returns:

str: Resulting binary string after applying the sigma1 function.

"""

# Implementation details...

# AND Operation

def andOp(operand1, operand2):

"""

Perform a bitwise AND operation between two binary strings of the same length.

Args:

operand1 (str): Binary string representing the first operand.

operand2 (str): Binary string representing the second operand.

Returns:

str: Binary string result of the AND operation.

"""

# Implementation details...

# NOT Operation

def notOp(operand):

"""

Perform a bitwise NOT operation on a binary string.

Args:

operand (str): Binary string to be negated.

Returns:

str: Resulting binary string after the NOT operation.

"""

# Implementation details...

# Ch Function

def Ch(opr1, opr2, opr3):

"""

Compute the Ch function as specified in the SHA-512 algorithm.

Args:

opr1 (str): Binary string representing the first operand.

opr2 (str): Binary string representing the second operand.

opr3 (str): Binary string representing the third operand.

Returns:

str: Resulting binary string after applying the Ch function.

"""

# Implementation details...

# Maj Function

def Maj(opr1, opr2, opr3):

"""

Compute the Maj function as specified in the SHA-512 algorithm.

Args:

opr1 (str): Binary string representing the first operand.

opr2 (str): Binary string representing the second operand.

opr3 (str): Binary string representing the third operand.

Returns:

str: Resulting binary string after applying the Maj function.

"""

# Implementation details...

# SIGMA0 Function

def SIGMA0(operand):

"""

Apply the SIGMA0 function to a 64-bit binary string.

Args:

operand (str): Binary string input.

Returns:

str: Resulting binary string after applying the SIGMA0 function.

"""

# Implementation details...

# SIGMA1 Function

def SIGMA1(operand):

"""

Apply the SIGMA1 function to a 64-bit binary string.

Args:

operand (str): Binary string input.

Returns:

str: Resulting binary string after applying the SIGMA1 function.

"""

# Implementation details...

# SHA-512 Function

def sha\_func(prv\_hash, new\_block):

"""

Compute the SHA-512 hash function for a given block of data and the previous hash value.

Args:

prv\_hash (str): Binary string representing the previous hash.

new\_block (str): Binary string representing the new block of message data.

Returns:

str: Binary string representing the new hash value.

"""

# Implementation details...

# HMAC-SHA512 Function

def HMAC\_SHA512(key, message):

"""

Compute the HMAC-SHA512 hash value for a given key and message.

Args:

key (str): Binary string representing the HMAC key.

message (str): Binary string representing the message.

Returns:

str: Binary string representing the HMAC-SHA512 hash value.

"""

# Implementation details...

# Chunkstring Function

def chunkstring(string, length):

"""

Split a string into chunks of a specified length.

Args:

string (str): The input string to be split.

length (int): The desired length of each chunk.

Returns:

Generator: A generator that yields the string chunks.

"""

# Implementation details...

if \_\_name\_\_ == "\_\_main\_\_":

# Main code that takes user input and computes HMAC-SHA512 hash value.

# Implementation details...