from sys import exit

from time import time

KeyLength = 10

SubKeyLength = 8

DataLength = 8

FLength = 4

# Tables for initial and final permutations (b1, b2, b3, ... b8)

IPtable = (2, 6, 3, 1, 4, 8, 5, 7)

FPtable = (4, 1, 3, 5, 7, 2, 8, 6)

# Tables for subkey generation (k1, k2, k3, ... k10)

P10table = (3, 5, 2, 7, 4, 10, 1, 9, 8, 6)

P8table = (6, 3, 7, 4, 8, 5, 10, 9)

# Tables for the fk function

EPtable = (4, 1, 2, 3, 2, 3, 4, 1)

S0table = (1, 0, 3, 2, 3, 2, 1, 0, 0, 2, 1, 3, 3, 1, 3, 2)

S1table = (0, 1, 2, 3, 2, 0, 1, 3, 3, 0, 1, 0, 2, 1, 0, 3)

P4table = (2, 4, 3, 1)

def perm(inputByte, permTable):

"""Permute input byte according to permutation table"""

outputByte = 0

for index, elem in enumerate(permTable):

if index >= elem:

outputByte |= (inputByte & (128 >> (elem - 1))) >> (index - (elem - 1))

else:

outputByte |= (inputByte & (128 >> (elem - 1))) << ((elem - 1) - index)

return outputByte

def ip(inputByte):

"""Perform the initial permutation on data"""

return perm(inputByte, IPtable)

def fp(inputByte):

"""Perform the final permutation on data"""

return perm(inputByte, FPtable)

def swapNibbles(inputByte):

"""Swap the two nibbles of data"""

return (inputByte << 4 | inputByte >> 4) & 0xff

def keyGen(key):

"""Generate the two required subkeys"""

def leftShift(keyBitList):

"""Perform a circular left shift on the first and second five bits"""

shiftedKey = [None] \* KeyLength

shiftedKey[0:9] = keyBitList[1:10]

shiftedKey[4] = keyBitList[0]

shiftedKey[9] = keyBitList[5]

return shiftedKey

# Converts input key (integer) into a list of binary digits

keyList = [(key & 1 << i) >> i for i in reversed(range(KeyLength))]

permKeyList = [None] \* KeyLength

for index, elem in enumerate(P10table):

permKeyList[index] = keyList[elem - 1]

shiftedOnceKey = leftShift(permKeyList)

shiftedTwiceKey = leftShift(leftShift(shiftedOnceKey))

subKey1 = subKey2 = 0

for index, elem in enumerate(P8table):

subKey1 += (128 >> index) \* shiftedOnceKey[elem - 1]

subKey2 += (128 >> index) \* shiftedTwiceKey[elem - 1]

return (subKey1, subKey2)

def fk(subKey, inputData):

"""Apply Feistel function on data with given subkey"""

def F(sKey, rightNibble):

aux = sKey ^ perm(swapNibbles(rightNibble), EPtable)

index1 = ((aux & 0x80) >> 4) + ((aux & 0x40) >> 5) + \

((aux & 0x20) >> 5) + ((aux & 0x10) >> 2)

index2 = ((aux & 0x08) >> 0) + ((aux & 0x04) >> 1) + \

((aux & 0x02) >> 1) + ((aux & 0x01) << 2)

sboxOutputs = swapNibbles((S0table[index1] << 2) + S1table[index2])

return perm(sboxOutputs, P4table)

leftNibble, rightNibble = inputData & 0xf0, inputData & 0x0f

return (leftNibble ^ F(subKey, rightNibble)) | rightNibble

def encrypt(key, plaintext):

"""Encrypt plaintext with given key"""

data = fk(keyGen(key)[0], ip(plaintext))

return fp(fk(keyGen(key)[1], swapNibbles(data)))

def decrypt(key, ciphertext):

"""Decrypt ciphertext with given key"""

data = fk(keyGen(key)[1], ip(ciphertext))

return fp(fk(keyGen(key)[0], swapNibbles(data)))

if \_\_name\_\_ == '\_\_main\_\_':

try:

assert encrypt(0b0000000000, 0b10101010) == 0b00010001

except AssertionError:

print("Error on encrypt:")

print("Output: ", encrypt(0b0000000000, 0b10101010), "Expected: ", 0b00010001)

exit(1)

try:

assert encrypt(0b1110001110, 0b10101010) == 0b11001010

except AssertionError:

print("Error on encrypt:")

print("Output: ", encrypt(0b1110001110, 0b10101010), "Expected: ", 0b11001010)

exit(1)

try:

assert encrypt(0b1110001110, 0b01010101) == 0b01110000

except AssertionError:

print("Error on encrypt:")

print("Output: ", encrypt(0b1110001110, 0b01010101), "Expected: ", 0b01110000)

exit(1)

try:

assert encrypt(0b1111111111, 0b10101010) == 0b00000100

except AssertionError:

print("Error on encrypt:")

print("Output: ", encrypt(0b1111111111, 0b10101010), "Expected: ", 0b00000100)

exit(1)

t1 = time()

for i in range(1000):

encrypt(0b1110001110, 0b10101010)

t2 = time()

print("Elapsed time for 1,000 encryptions: {:0.3f}s".format(t2 - t1))