```python
# main.py -- put your code here!
# Tasks:
# https://cours-info.iut-bm.univ-fcomte.fr/pmwiki-2.2.131/pmwiki.php/MonWiki/
  SecurityForIoT

import pyb
from sys import exit
import utime

'''THE ONE-TIME PAD'''

def encrypt(p, k):
    c = ""
    if len(p) > len(k):
        #print('Very Long Message')
        exit()
    for i in range(len(p)):
        a = int(p[i])
        b = int(k[i])
        tmp = a^b
        c += str(tmp)
    return c

def decrypt(c, k):
    return encrypt(c, k)

'''LINEAR CONGRUENTIAL GENERATORS'''

def lcg(a,c,m,seed,nums):
    rand = [];
    for i in range(nums):
        if(coprime(c,m)):
            print("{0} and {1} are coprime".format(c,m))
        seed = (a*seed + c) % m
        rand.append(seed)
    return rand

def gcd(num1,num2):
    c = num1
    m = num2
    while(num2 != 0):
        temp = num2
        num2 = num1 % num2
        num1 = temp
    gcd = num1
    #print("\n HCF of {0} and {1} = {2}".format(c,m, gcd))
    return gcd

def coprime(c, m):
    return gcd(c,m) == 1

'''Decimal to Binary'''

def destobin(x):
    if x == 0:
        return '0'
```

```
    res = ''
    while x > 0:
        res = ('0' if x % 2 == 0 else '1') + res
        x //= 2
    return res

'''Count the number of bit of key'''
def bitCount(n):
    zeroCount = int(n.count('0'))
    oneCount = int(n.count('1'))
    #numberOfBit = zeroCount + numberOfBit
    return int(zeroCount + oneCount)

'''Number of bit of key and message should be equal'''

def sameSize(numberOfBitinKey,numberOfBitinMesg,msg,key,type):
    bitmesg = msg
    bkey = key
    if((numberOfBitinKey > numberOfBitinMesg) or type=="message"):
        extraBit = numberOfBitinKey - numberOfBitinMesg
        if(extraBit>0):
            for j in range(0,extraBit):
                bitmesg = "0{0}".format(bitmesg)
            return bitmesg
        else:
            return bitmesg

    if((numberOfBitinKey < numberOfBitinMesg)  or type=="key"):
        extraBit = numberOfBitinMesg - numberOfBitinKey
        if(extraBit>0):
            for j in range(0,extraBit):
                bkey = "0{0}".format(bkey)
            return bkey
        else:
            return bkey

'''THE XORSHIFT'''

def xorshift(a,b,c):

    x = 123456789
    y = 362436069
    z = 521288629
    w = 88675123

    def _random():
        nonlocal x, y, z, w
        t = x ^ ((x << b) & 0xFFFFFFFF)  # 32bit
        x, y, z = y, z, w
        w = (w ^ (w >> a)) ^ (t ^ (t >> c))
        return w

    return _random

''' Fast exponentiation word'''
def getWord(e):
```

```
    #mess = e
    message = e

    for mes in message:
        index = message.find('0')
        if(index == 0):
            message = message.replace('0','',1)
            print(message)
    message = message.replace('1','',1)
    print(message)
    word = ''
    for elem in message:
        #print(elem)
        if(elem == "0"):
            word = word + "S"
        if(elem == "1"):
            word = word + "SX"
    return word

''' Fast exponentiation '''

def fast_exp(message,X):
    word = getWord(message)
    print(word)
    res = X;
    for elem in word:
        if elem == "S":
            res = res*res
        else:
            res = res*X
    return res

'''Recursive Fast exponentiation for test '''

def binpow (a,n):
    if (n == 0):
        return 1
    if (n % 2 == 1):
        return binpow (a, n-1) * a;
    else:
        b = binpow (a, n/2);
        return b * b;

'''Diffie-Hellman Key Exchange'''

def deff():
    # Variables Used
    sharedPrime = 23    # p
    sharedBase = 5      # g

    aliceSecret = 6     # a
    bobSecret = 15      # b

    # Begin
    print( "Publicly Shared Variables:")
```

```
    print( "Publicly Shared Prime:" , sharedPrime )
    print( "Publicly Shared Base:" , sharedBase )

    # Alice Sends Bob A = g^a mod p
    A = (sharedBase**aliceSecret) % sharedPrime
    print( "\nAlice Sends Over Public Chanel:" , A )

    # Bob Sends Alice B = g^b mod p
    B = (sharedBase ** bobSecret) % sharedPrime
    print("Bob Sends Over Public Chanel: ", B )

    print( "\n------------\n" )
    print( "Privately Calculated Shared Secret:" )
    # Alice Computes Shared Secret: s = B^a mod p
    aliceSharedSecret = (B ** aliceSecret) % sharedPrime
    print( "Alice Shared Secret: ", aliceSharedSecret )

    # Bob Computes Shared Secret: s = A^b mod p
    bobSharedSecret = (A**bobSecret) % sharedPrime
    print( "Bob Shared Secret:", bobSharedSecret )

'''To obtain the prime factors of n'''

def factor(n):
  if n==1:
      return set([])
  else:
      for k in range(2,n+1):
          if n%k == 0:
              L = factor(n/k)
  return L.union([k])

''' Key generation for El Gamal '''

def keys(p,g,a):
    A = g**a%p
    public = (p,g,A)
    private = a
    return (public, private)

'''Encryption of El Gamal'''

def digit(message, public, b):
    (p,g,A) = public
    B = g**b%p
    c = message*A**b%p
    return (B, c)

'''Decryption of El Gamal'''

def decipher(cryptogram, public, private):
    (p,g,A), a = public, private
    (B,c) = cryptogram
    return B**(p-1-a)*c%p

def checkPrime(n):
```

```
    m = int(n/2)
    j = 2
    for i in range(0,m):
        if(n % j == 0):
            return 0 # Not Prime
        j +=1
    return 1;                   # Prime

def powMod(a,b,n):
    x = 1
    y = a
    while b > 0:
        if (int(b % 2) == 1):
            x = int((x * y) % n) # multiplying with base
            #print(x)
        y = int((y * y) % n) # Squaring the base
        #print(y)
        b = int(b/2)
        #print(b)
    return int(x % n)


def main():
    start_time_lcg = utime.ticks_us()
    key_lcg = lcg(1140671485,128201163,2**24,pyb.rng(),10);
    end_time_lcg = utime.ticks_us()
    totalTime_lcg = utime.ticks_diff(end_time_lcg,start_time_lcg)
    print("Time using LCG: {0} micoseconds".format(totalTime_lcg))
    numberOfBit = 0;
    bkey = 0;

    r = xorshift(12,25,27)
    rand = []
    start_time_XORShift = utime.ticks_us()
    for i in range(10):
        rand.append(r())
    end__time_XORShift = utime.ticks_us()
    totalTime_XORShift = utime.ticks_diff(end__time_XORShift,start_time_XORShift)
    print("Time using XORShift: {0} micoseconds".format(totalTime_XORShift))
    print("Time difference between LCG and XORShift : {0} microseconds".format
      (totalTime_lcg-totalTime_XORShift))


    for j in rand:
        print("{0} using xorshift".format(j))

    '''Change the LED color to check the key is randomized or not'''

    even = pyb.LED(4)
    odd = pyb.LED(3)

    '''Fast Exponentials'''

    print(fast_exp('10011',3));
    print(binpow(3,19));
    bitmesg = '0010010100011110011111110'
```

```
    #bitmesg = '001001010001111001111110'

    key = '1011000101110010001110100'

    encM = encrypt(bitmesg,key);
    decryptM = decrypt(encM,key);
    if(encM == '1001010001101100010001010'):
        pyb.LED(2).on()

    if(decryptM == bitmesg):
        pyb.LED(3).on()

    deff()
    p = int(input("Enter a prime number (17, 19, 23, etc): "))
    q = int(input("Enter another prime number (Not one you entered above): "))
    if not (checkPrime(p)) and not (checkPrime(q)):
        print("Both numbers are not prime. Please enter prime numbers only...\n");
    elif (not checkPrime(p)):
        print("The first prime number you entered is not prime, please try again...
          \n");
    elif (not checkPrime(q)):
        print("The second prime number you entered is not prime, please try
          again...\n");
    n = p * q;
    '''=====================================
       =========== Euler Function ===========
       ====================================='''
    phin = (p - 1) * (q - 1);

    '''=====================================
       =========== Public key ===========
       ====================================='''
    e = 0
    for i in range(3,phin):
        if (gcd(phin, i)== 1):
            e = i
            break;
    #d = int(e+1);
    d = 0;
    for d in range(e+1,n):
        if (((d * e) % phin) == 1):
            break;
    mess = int(input("Enter some numerical data: "))
    if (mess > n-1):
        print("Your message is too big. Please send another message or increase n
          ('p' and 'q')\n");
    cipher_rsa = powMod(mess, e, n);
    print("The cipher text is: {0}".format(cipher_rsa))

    decrypt_rsa = powMod(cipher_rsa, d, n);
    print("The decrypted text is: {0}".format(decrypt_rsa))

    for i in key_lcg:
        if (i % 2) == 0:
          print("{0} is Even".format(i))
          bkey = destobin(i)
```

```
            numberOfBitinKey = bitCount(bkey)
            numberOfBitinMesg = bitCount(bitmesg)
            newbkey = sameSize(numberOfBitinKey,numberOfBitinMesg,bitmesg,bkey,"key")
            newbitmesg = sameSize
              (numberOfBitinKey,numberOfBitinMesg,bitmesg,bkey,"message")
            encM_lcg = encrypt(newbitmesg,newbkey)
            decryptM_lcg = decrypt(encM_lcg,newbkey)

            print("Number of bit in key = {0}".format(numberOfBitinKey))
            print("Number of bit in Message = {0}".format(numberOfBitinMesg))
            print("key = {0}".format(newbkey))
            print("msg = {0}".format(newbitmesg))
            print("encrypt = {0}".format(encM_lcg))
            print("decrypt = {0}".format(decryptM_lcg))
            x = int(newbitmesg) - int(decryptM_lcg)
            print("msg-decrypt = {0}".format(x))

            even.toggle()
            pyb.delay(1000)
            even.toggle()


          else:
            print("{0} is Odd".format(i))
            bkey = destobin(i)
            numberOfBitinKey = bitCount(bkey)
            numberOfBitinMesg = bitCount(bitmesg)
            newbkey = sameSize(numberOfBitinKey,numberOfBitinMesg,bitmesg,bkey,"key")
            newbitmesg = sameSize
              (numberOfBitinKey,numberOfBitinMesg,bitmesg,bkey,"message")
            encM_lcg = encrypt(newbitmesg,newbkey)
            decryptM_lcg = decrypt(encM_lcg,newbkey)

            print("Number of bit in key = {0}".format(numberOfBitinKey))
            print("Number of bit in Message = {0}".format(numberOfBitinMesg))
            print("key = {0}".format(newbkey))
            print("msg = {0}".format(newbitmesg))
            print("encrypt = {0}".format(encM))
            print("decrypt = {0}".format(decryptM_lcg))
            x = int(newbitmesg) - int(decryptM)
            print("msg-decrypt = {0}".format(x))
            odd.toggle()
            pyb.delay(1000)
            odd.toggle()
            even.off()
            odd.off()



if __name__ == "__main__":
    main()
```