```
#!/usr/bin/env python

## Dimcho Karakashev
### Encryption of file

import sys
import os
from BitVector import *


expansion_permutation = [31, 0, 1, 2, 3, 4, 3, 4, 5, 6, 7, 8, 7, 8, 9, 10, 11,
12, 11, 12, 13, 14, 15, 16, 15, 16, 17, 18, 19, 20, 19, 20, 21, 22, 23, 24, 23,
24, 25, 26, 27, 28, 27, 28, 29, 30, 31, 0]


key_permutation_1 = [56,48,40,32,24,16,8,0,57,49,41,33,25,17,
                     9,1,58,50,42,34,26,18,10,2,59,51,43,35,
                     62,54,46,38,30,22,14,6,61,53,45,37,29,21,
                     13,5,60,52,44,36,28,20,12,4,27,19,11,3]

key_permutation_2 = [13,16,10,23,0,4,2,27,14,5,20,9,22,18,11,
                     3,25,7,15,6,26,19,12,1,40,51,30,36,46,
                     54,29,39,50,44,32,47,43,48,38,55,33,52,
                     45,41,49,35,28,31]

p_box_perm = [15, 6, 19, 20, 28, 11, 27, 16,
              0, 14, 22, 25, 4, 17, 30 , 9,
              1, 7, 23 , 13, 31, 26, 2, 8,
              18, 12, 29, 5, 21, 10, 3 ,24]

shifts_for_round_key_gen = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

s_boxes = [None,None,None,None,None,None,None,None]


s_boxes[0] = [ [14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7],
               [0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8],
               [4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0],
               [15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13] ]

s_boxes[1] = [ [15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10],
               [3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5],
               [0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15],
               [13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9] ]

s_boxes[2] = [ [10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8],
               [13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1],
               [13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7],
               [1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12] ]

s_boxes[3] = [ [7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15],
               [13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9],
               [10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4],
               [3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14] ]

s_boxes[4] = [ [2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9],
               [14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6],
               [4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14],
               [11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3] ]

s_boxes[5] = [ [12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11],
               [10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8],
               [9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6],
               [4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13] ]
```

```
s_boxes[6] = [ [4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1],
               [13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6],
               [1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2],
               [6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12] ]

s_boxes[7] = [ [13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7],
               [1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2],
               [7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8],
               [2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11] ]


def get_encryption_key():
    while True:
        key = input("Enter 8 character key: ")
        if len(key) == 8:
            break
        else:
            print("\nYou have to enter exactly 8 characters!\n")
            continue
    #creates a bitvector rep. of the key
    key = BitVector(textstring=key)
    key = key.permute(key_permutation_1) #initial permutation of the key
    return key

def get_enc_or_dec():
    while True:
        enc_or_dec = input("Do you want to encrypt(type e) or decrypt(type d)?
");
        if enc_or_dec == "e" or enc_or_dec == "d":
            break
        else:
            print("\nChoose either e or d!\n")
            continue
    return enc_or_dec

def extract_round_key( key_after_per_1 ):
    round_key_list = []
    key_temp = key_after_per_1.deep_copy()                    #copys the key so I
don't mess the original version
    for round in range(16):
        [left_half, right_half] = key_temp.divide_into_two() #separating the key
into two parts
        left_half << shifts_for_round_key_gen[round]          #shifting each half
        right_half << shifts_for_round_key_gen[round]
        key_temp = (left_half + right_half)                  #combining the new
halfs
        round_key = key_temp.permute(key_permutation_2)       #making the key to
46 bits
        round_key_list.append(round_key)
    return round_key_list

def s_box_substitution ( out_xor ):
    output = BitVector( size = 32 )                          #new vector created
for output
    #six_bit_seg = [ out_xor[x:x+6] for x in range(48) if x % 6 == 0 ]
#seperates the 48 bit to 6 parts of 8 bits
    #{print(out_xor[x*6:x*6+6], end = " ") for x in range(8)}
    six_bit_seg = [ out_xor[x*6:x*6+6] for x in range(8)]
    counter = 0                                              #keep track how many
4 bits were written
    for seg in six_bit_seg:
        row = 2*seg[0] + seg[-1]                              #finds the row and
column of the s-table
```

```
        column = int(seg[1:-1])
        output[counter*4:counter*4+4] = BitVector( intVal = s_boxes[counter]
[row][column], size = 4)
        counter += 1
    return output

def encrypt():

    key = get_encryption_key()
    #print key.get_bitvector_in_ascii()
    round_keys = extract_round_key( key )                    #finds an array of
keys for each round of encryption
    enc_or_dec=get_enc_or_dec()

    if enc_or_dec == "d":
        try:
            fileOutput = open('decrypted.txt', 'wb')
        except:
            print("Error opening decrypted.txt")
            sys.exit(1)
        round_keys.reverse()
        bv = BitVector(filename='encrypted.txt')
    else:
        try:
            fileOutput = open('encrypted.txt', 'wb')
        except:
            print("Error opening encrypted.txt")
            sys.exit(1)
        bv = BitVector(filename='message.txt')

    while bv.more_to_read:
        bitvec = bv.read_bits_from_file( 64 )
        size = bitvec.length()
        if size > 0:                              #Note: is is true?
            if size < 64:
                bitvec.pad_from_right(64 - size)
            [LE, RE] = bitvec.divide_into_two()  # splits into two halfs
            for round_key in round_keys:
                newRE = RE.permute(expansion_permutation) #expansion permutation
                newRE = newRE ^ round_key                #xoring the round key
and expanded perm
                newRE = s_box_substitution(newRE)        #using s-tables and
output 32 bits
                newRE = newRE.permute(p_box_perm)        #p-box perm
                newRE = LE ^ newRE
                [LE, RE] = [RE, newRE]  # needed for one last swap after the
16th round
            bitvec = RE + LE
            bitvec.write_to_file(fileOutput)

    bv.close_file_object()  # closing the file for reading
    fileOutput.close()  # close the file to encrypt


encrypt()

sys.exit(0)



Key:
ecepurdu

Encrypted Version:
```

_iT@Z3b[illegible]N)[illegible]kE@iYv[illegible]#@z[illegible].#Yr[illegible]r?î}ar[:=␣NM"Z#i@+!@z[illegible]m#Q57#@iA@L@![illegible]!
[illegible]U@? kpg#h[illegible]##[illegible]5[illegible]` #@~@Ak#@I[illegible]Z#[illegible]{/ [illegible]'[illegible]0[illegible]L@v[illegible]9#jʌ[illegible]#[illegible]#a<@?
@{k@###|@$@!2@)#ê#@M[illegible]M#@C#@#@kp@'UV#@Xg@w[illegible]V#@?î}ar[:0#o
{[illegible]g3=DȟYEVḍQ#@d.[illegible]#DNU[illegible]Q#j#/U@#[illegible]<O[illegible]#@#e#*R[illegible]#[illegible]#[illegible]#[illegible]##V|
[illegible]q[illegible]#P@QKi#[illegible]$@#[illegible]#a<@d[illegible]S@#RS@#j*[illegible]aE[illegible]`#[illegible]I@FxexY[illegible]y##[illegible]#@1@H[illegible]34I@J
%##71@±@LM@#[illegible](##?#mb#z@#0k@R$[illegible]c#i@p@&[illegible]'@:+# [illegible]@G=[illegible]V[illegible]{@9@ N@5@#@8m
###@#_@,1@#[illegible]v@&#%0;.V@!;#@CS@W##[illegible]~˚@~@v#@@[illegible]#@5#[illegible]#c@o@W#@mGxF@M
ǰi@G+I#[illegible]kP#@Q@(o[illegible]Ù[illegible]8@SR[illegible]ˌ@x@w[illegible]#4[illegible]'[illegible]a[7:[illegible]m[illegible]ibC[illegible]u#/&[illegible]*@#@
@0nf/@[illegible].@F@p=A[illegible]T@
#@{[illegible]mH[illegible]^[illegible]wf@z@#@岁VR@K[illegible]C-
[illegible]s@v@U@Ɏ[illegible]###[illegible]4])@#[illegible]i[illegible]0hƑ@m$!]q@P#@##f_=@ü@+
(#@d4@Bl@B@#\~#@j~#@ti[0@T$@ẃ@@C@⊝@#b&ɜy[illegible]G[illegible]6@I C@##[illegible]#ST##V#@t[illegible]#&K@ [illegible]
▫[illegible]M@|[illegible]XI1[illegible]n@6@Bkv@J@1)@b@=#[illegible]#@ļA#kAL@W#3[illegible]ω@N;<#y A@!@#[illegible]#@ń[illegible]?
@.##@y@)@2@x[illegible]n@#K[illegible]^P*[illegible]#[illegible]#@J#@###@[illegible]#R@]H@wE ɟ@Ỏ[illegible]H##h@k@#@j[illegible]#lG@UZ@@.
#[illegible]=#/@8#@m1@Ű][illegible]-
Ƀ#ED#a=@4[illegible]k[illegible]E#@e@###[illegible]#@###[illegible]_@S@T[illegible]_@#[illegible]Ƀ▫▫@a~;@7& [illegible]R@m[illegible]2#rM?[illegible]m@Π[illegible]#@

Decrypted Version:

US investigators say the culprits spent at least two months copying critical files. A purported member of the Guardians of Peace (GOP) who have claimed to have performed the hack stated that they have had access for at least a year prior to its discovery in November 2014, according to Wired. The hackers involved claim to have taken more than 100 terabytes of data from Sony, but that claim has never been confirmed. The attack was conducted using malware. Although Sony was not specifically mentioned in its advisory, US-CERT said that the attackers used a Server Message Block (SMB) Worm Tool to conduct attacks against a major entertainment company. Components of the attack included a listening implant, backdoor, proxy tool, destructive hard drive tool, and destructive target cleaning tool. The components clearly suggest an intent to gain repeated entry, extract information, and be destructive, as well as remove evidence of the attack. The cleaning tool used on Sony's computer infrastructure, Wiper, is a malware program designed to erase data from the servers.
#######