```
from operator import xor
## unsigned 32 bit integers

from ctypes import c_uint32

MASK32 = 0xffffffff
delta = c_uint32(0x9e3779b9).value
ROUNDS = 32


def add(x,y):
    return c_uint32(x + y).value

def subtract(x,y):
    return c_uint32(x - y).value

def leftshift4add(x,k):
    return add(c_uint32(x << 4).value, c_uint32(k).value)

def rightshift5add(x,k):
    return add(c_uint32(x >> 5).value, c_uint32(k).value)


class TEA:

    def __init__(self, p,k):
        self.P = p
        print(c_uint32(P).value)
        self.K = [c_uint32(((MASK32 << x) & k) >> x).value for x in range(96,-1,-32)]
        for x in self.K: print (hex(x))



## you need to define the following two functions
    def encrypt(self):
      v = [c_uint32(((MASK32 << y) & P) >> y).value for y in range(64,-1,-32)]
```

```
        #for y in v: print(hex(y))
        print(hex(v[1]),hex(v[2]))
        u = [c_uint32(((MASK32 << x) & K) >> x).value for x in range(96,-1,-32)]
        #for x in u: print(hex(x))

        sum = c_uint32(0).value
        for i in range(32):
          sum += delta
          v[1] += (leftshift4add(v[2], u[0])) ^ (add(v[2], sum)) ^ (rightshift5add(v[2], u[1]))
          v[2] += (leftshift4add(v[1], u[2])) ^ (add(v[1], sum)) ^ (rightshift5add(v[1], u[3]))
        return v[1],v[2]

      def decrypt(self, C):
        a,b = C[0], C[1]
        u = [c_uint32(((MASK32 << x) & K) >> x).value for x in range(96,-1,-32)]
        #for x in u: print(hex(x))
        sum = c_uint32(0).value
        #sum = (delta * 32)
        sum = rightshift5add(delta, sum)
        for i in range(32):
          a -= (leftshift4add(b, u[2])) ^ (add(b, sum)) ^ (rightshift5add(b, u[3]))
          b -= (leftshift4add(a, u[0])) ^ (add(a, sum)) ^ (rightshift5add(a, u[1]))
          sum -= delta
        return a,b


    """
    Output answers
    """

    if __name__ == '__main__':


        # TEA ALGORITHM
        K = 0xA56BABCD00000000FFFFFFFFABCDEF01
        P = 0x0123456789ABCDEF
```

```
    T = TEA(P, K)
    C = T.encrypt()
    P2 = T.decrypt(C)
    print ("Original plaintext: " + str(hex(P)))
    #Cip = (hex(C[0]) << 32) | (hex(C[1]))
    print ("Ciphertext: " + str(hex(C[0])),str(hex(C[1])))
    #print (hex(C))
    print ("Decrypted plaintext: " + str(hex(P2[0])),str(hex(P2[1])))
```

```
 2309737967
 0xa56babcd
 0x0
 0xffffffff
 0xabcdef01
 0x1234567 0x89abcdef
 Original plaintext: 0x123456789abcdef
 Ciphertext: 0x1024f5e79b 0x1031e96f53
 Decrypted plaintext: 0x1dcce25c3 0x2784405e4
```

```
from ctypes import *

MASK32 = 0xffffffff

def encrypt(v, k):
  v0, v1 = c_uint32(v[0]), c_uint32(v[1])
  print(v0 ,"wow")
  delta = 0x9e3779b9
  k0, k1, k2, k3 = k[0], k[1], k[2], k[3]
  total = c_uint32(0)
  for i in range(32):
    total.value += delta
    v0.value += ((v1.value<<4) + k0) ^ (v1.value + total.value) ^ ((v1.value>>5) + k1)
    v1.value += ((v0.value<<4) + k2) ^ (v0.value + total.value) ^ ((v0.value>>5) + k3)
  return v0.value, v1.value
def decrypt(v, k):
  v0, v1 = c_uint32(v[0]), c_uint32(v[1])
  delta = 0x9e3779b9
```

```
    k0, k1, k2, k3 = k[0], k[1], k[2], k[3]
    total = c_uint32(delta * 32)
    for i in range(32):
      v1.value -= ((v0.value<<4) + k2) ^ (v0.value + total.value) ^ ((v0.value>>5) + k3)
      v0.value -= ((v1.value<<4) + k0) ^ (v1.value + total.value) ^ ((v1.value>>5) + k1)
      total.value -= delta
    return v0.value, v1.value
# test
if __name__ == "__main__":
    #K = 0xA56BABCD00000000FFFFFFFFABCDEF01
    #P = 0x0123456789ABCDEF
# Plaintext to be encrypted , Two 32 An integer , namely 64bit The plaintext data of
    #v = [c_uint32(((MASK32 << y) & P) >> y).value for y in range(64,-1,-32)]
    value = [0x1234567, 0x89abcdef]
# four key, Each is 32bit, That is, the key length is 128bit
    #u = [c_uint32(((MASK32 << x) & K) >> x).value for x in range(96,-1,-32)]
    key = [0xa56babcd, 0x0, 0xffffffff, 0xabcdef01 ]
print("Data is : ", hex(value[0]), hex(value[1]))
res = encrypt(value, key)
print("Encrypted data is : ", hex(res[0]), hex(res[1]))
res = decrypt(res, key)
print("Decrypted data is : ", hex(res[0]), hex(res[1]))
```

```
Data is :  0x1234567 0x89abcdef
c_uint(19088743) wow
Encrypted data is :  0xa0761126 0xd09724fd
Decrypted data is :  0x1234567 0x89abcdef
```

```
#from operator import xor
## unsigned 32 bit integers

from ctypes import c_uint32

MASK32 = 0xffffffff

def encrypt(v, k):
```

```
    v0, v1 = c_uint32(v[0]), c_uint32(v[1])
    delta = 0x9e3779b9
    k0, k1, k2, k3 = k[0], k[1], k[2], k[3]
    total = c_uint32(0)
    for i in range(32):
      total.value += delta
      v0.value += ((v1.value<<4) + k0) ^ (v1.value + total.value) ^ ((v1.value>>5) + k1)
      v1.value += ((v0.value<<4) + k2) ^ (v0.value + total.value) ^ ((v0.value>>5) + k3)
      #print(v0.value, v1.value)
    return v0.value, v1.value
  def decrypt(v, k):
    v0, v1 = c_uint32(v[0]), c_uint32(v[1])
    delta = 0x9e3779b9
    k0, k1, k2, k3 = k[0], k[1], k[2], k[3]
    total = c_uint32(delta * 32)
    for i in range(32):
      v1.value -= ((v0.value<<4) + k2) ^ (v0.value + total.value) ^ ((v0.value>>5) + k3)
      v0.value -= ((v1.value<<4) + k0) ^ (v1.value + total.value) ^ ((v1.value>>5) + k1)
      total.value -= delta
    return v0.value, v1.value


  """
  Output answers
  """

  if __name__ == '__main__':


      # TEA ALGORITHM
      K = 0xA56BABCD00000000FFFFFFFFABCDEF01
      P = 0x0123456789ABCDEF

      w = [c_uint32(((MASK32 << y) & P) >> y).value for y in range(64,-1,-32)]
      #for y in w: print(hex(y))
      #print(hex(w[1]),hex(w[2]))
      i0 = (int((hex(w[1])[2:]), 16))
      i1 = (int((hex(w[2])[2:]), 16))
```

```
    #print(i0,i1)

    v=[i0,i1]

    value = [i0,i1]
    #value = [hex(w[1]),hex(w[2])]
    # four key, Each is 32bit, That is, the key length is 128bit
    u = [c_uint32(((MASK32 << x) & K) >> x).value for x in range(96,-1,-32)]
    l0 = (int((hex(u[0])[2:]), 16))
    l1 = (int((hex(u[1])[2:]), 16))
    l2 = (int((hex(u[2])[2:]), 16))
    l3 = (int((hex(u[3])[2:]), 16))
    #key = [0xa56babcd, 0x0, 0xffffffff, 0xabcdef01 ]
    key = [l0, l1, l2, l3]
    print("Data is : ", hex(w[1])[2:],hex(w[2])[2:])
    res = encrypt(value, key)
    print("Encrypted data is : ", hex(res[0])[2:], hex(res[1])[2:])
    res = decrypt(res, key)
    print("Decrypted data is : ", hex(res[0])[2:], hex(res[1])[2:])
    c = hex(res[0])[2:]+ hex(res[1])[2:]
    print("DEC data:", c)
```

```
Data is :  1234567 89abcdef
Encrypted data is :  a0761126 d09724fd
Decrypted data is :  1234567 89abcdef
DEC data: 123456789abcdef
```

```
#from operator import xor
## unsigned 32 bit integers

from ctypes import c_uint32

MASK32 = 0xffffffff

def encrypt(v, k):
  v0, v1 = c_uint32(v[0]), c_uint32(v[1])
  delta = 0x9e3779b9
```

```
    k0, k1, k2, k3 = k[0], k[1], k[2], k[3]
    total = c_uint32(0)
    for i in range(32):
      total.value += delta
      v0.value += ((v1.value<<4) + k0) ^ (v1.value + total.value) ^ ((v1.value>>5) + k1)
      v1.value += ((v0.value<<4) + k2) ^ (v0.value + total.value) ^ ((v0.value>>5) + k3)
      #print(v0.value, v1.value)
    return v0.value, v1.value
  def decrypt(v, k):
    v0, v1 = c_uint32(v[0]), c_uint32(v[1])
    #v0, v1 = (v[0]), (v[1])
    delta = 0x9e3779b9
    k0, k1, k2, k3 = k[0], k[1], k[2], k[3]
    total = c_uint32(delta<<5)
    for i in range(32):
      v1.value -= ((v0.value<<4) + k2) ^ (v0.value + total.value) ^ ((v0.value>>5) + k3)
      v0.value -= ((v1.value<<4) + k0) ^ (v1.value + total.value) ^ ((v1.value>>5) + k1)
      total.value -= delta
    return v0.value, v1.value


  """
  Output answers
  """

  if __name__ == '__main__':


      # TEA ALGORITHM
      K = 0xA56BABCD00000000FFFFFFFFABCDEF01
      P = 0x0123456789ABCDEF

      w = [c_uint32(((MASK32 << y) & P) >> y).value for y in range(64,-1,-32)]
      #for y in w: print(hex(y))
      #print(hex(w[1]),hex(w[2]))
      i0 = (int(hex(w[1]), 16))
      i1 = (int(hex(w[2]), 16))
      #print(i0,i1)
```

```
    v=[i0,i1]

    value = [i0,i1]
    #value = [hex(w[1]),hex(w[2])]
    # four key, Each is 32bit, That is, the key length is 128bit
    u = [c_uint32(((MASK32 << x) & K) >> x).value for x in range(96,-1,-32)]
    l0 = (int(hex(u[0]), 16))
    l1 = (int(hex(u[1]), 16))
    l2 = (int(hex(u[2]), 16))
    l3 = (int(hex(u[3]), 16))
    #key = [0xa56babcd, 0x0, 0xffffffff, 0xabcdef01 ]
    key = [l0, l1, l2, l3]
    print("Data is : ", hex(w[1]),hex(w[2]))
    res = encrypt(value, key)
    print("Encrypted data is : ", hex(res[0]), hex(res[1]))
    res = decrypt(res, key)
    print("Decrypted data is : ", hex(res[0]), hex(res[1]))
    c = (hex(res[0])[2:]).zfill(8)+(hex(res[1])[2:]).zfill(8)
    print("DEC data:", c)

     Data is :  0x1234567 0x89abcdef
     Encrypted data is :  0xa0761126 0xd09724fd
     Decrypted data is :  0x1234567 0x89abcdef
     DEC data: 0123456789abcdef


import os

from ctypes import c_uint32

MASK32 = 0xffffffff

def encrypt(v, k):
  v0, v1 = c_uint32(v[0]), c_uint32(v[1])
  delta = 0x9e3779b9
  k0, k1, k2, k3 = k[0], k[1], k[2], k[3]
  total = c_uint32(0)
```

```
    for i in range(32):
      total.value += delta
      v0.value += ((v1.value<<4) + k0) ^ (v1.value + total.value) ^ ((v1.value>>5) + k1)
      v1.value += ((v0.value<<4) + k2) ^ (v0.value + total.value) ^ ((v0.value>>5) + k3)
      #print(v0.value, v1.value)
    return v0.value, v1.value

  def decrypt(v, k):
    v0, v1 = c_uint32(v[0]), c_uint32(v[1])
    delta = 0x9e3779b9
    k0, k1, k2, k3 = k[0], k[1], k[2], k[3]
    total = c_uint32(delta<<5)
    for i in range(32):
      v1.value -= ((v0.value<<4) + k2) ^ (v0.value + total.value) ^ ((v0.value>>5) + k3)
      v0.value -= ((v1.value<<4) + k0) ^ (v1.value + total.value) ^ ((v1.value>>5) + k1)
      total.value -= delta
    return v0.value, v1.value

  def open_file(filename, chunk_size):
          """
          Opens a file as binary and puts its content
          into an array in which each array cell is
          chunk_size bits in hexadecimal form written
          as string.
          """
          with open(filename, "rb") as f:
              hex_array = []
              for offset in range(0, os.path.getsize(filename), 8):
                  hex_array.append(bytes.hex(f.read(8)))
                  f.seek(offset + 8)

              f.close()

          return hex_array


  if __name__ == '__main__':
```

```
op_list= []
result = ""
actstr = ""
op_list = open_file('/content/msg.txt', 64)
print(op_list)
op_list.pop()
print(op_list)

K = 0xA56BABCD00000000FFFFFFFFABCDEF01
u = [c_uint32(((MASK32 << x) & K) >> x).value for x in range(96,-1,-32)]
l0 = (int(hex(u[0]), 16))
l1 = (int(hex(u[1]), 16))
l2 = (int(hex(u[2]), 16))
l3 = (int(hex(u[3]), 16))
  #key = [0xa56babcd, 0x0, 0xffffffff, 0xabcdef01 ]
key = [l0, l1, l2, l3]

for ih in op_list:
  P = int(ih, base=16)
  w = [c_uint32(((MASK32 << y) & P) >> y).value for y in range(64,-1,-32)]
  #for y in w: print(hex(y))
  #print(hex(w[1]),hex(w[2]))
  i0 = (int(hex(w[1]), 16))
  i1 = (int(hex(w[2]), 16))
  #print(i0,i1)
  v=[i0,i1]
  value = [i0,i1]
  res = encrypt(value, key)
  result += (hex(res[0])[2:]+hex(res[1])[2:])
  #c = hex(res[0])+hex(res[1])[2:]
  res = decrypt(res, key)
  actstr += (hex(res[0])[2:]+hex(res[1])[2:])
print(result)
print(actstr)
#print(type(actstr))
s = (bytes.fromhex(actstr).decode('utf-8'))
```

```
  print(s)
  #print(type(f))


  f= open("msg.txt.enc","w+")
  f.write(result)
  f.close()

  f1= open("msg.txt.dec","w+")
  f1.write(s)
  f1.close()
```

```
['466f75722073636f', '726520616e642073', '6576656e20796561', '72732061676f206f', '7572206661746865', '72732062726f7567', '68742
['466f75722073636f', '726520616e642073', '6576656e20796561', '72732061676f206f', '7572206661746865', '72732062726f7567', '68742
1ea3fc3ba790b0db61bc567c15792ae22ca0d03736bb590f15c4f9faa627897961ce4057a207a285eae0135f0c00cedf9178030b0d778c5755fc405f47bdff3
466f75722073636f726520616e6420736576656e207965617273206167462657273206270675676874206666f727468206f6e207468697
Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated t
```

```
from array import array
import os

from ctypes import c_uint32

MASK32 = 0xffffffff
import secrets

def encrypt(v, k, pp):
  v0, v1 = c_uint32(v[0]), c_uint32(v[1])
  v0.value = v0.value ^ (pp[0])
  v1.value = v1.value ^ (pp[1])
  delta = 0x9e3779b9
  k0, k1, k2, k3 = k[0], k[1], k[2], k[3]
  total = c_uint32(0)
```

```
  for i in range(32):
    total.value += delta
    v0.value += ((v1.value<<4) + k0) ^ (v1.value + total.value) ^ ((v1.value>>5) + k1)
    v1.value += ((v0.value<<4) + k2) ^ (v0.value + total.value) ^ ((v0.value>>5) + k3)
    #print(v0.value, v1.value)
  return v0.value, v1.value

def decrypt(v, k):
  v0, v1 = c_uint32(v[0]), c_uint32(v[1])
  delta = 0x9e3779b9
  k0, k1, k2, k3 = k[0], k[1], k[2], k[3]
  total = c_uint32(delta<<5)
  for i in range(32):
    v1.value -= ((v0.value<<4) + k2) ^ (v0.value + total.value) ^ ((v0.value>>5) + k3)
    v0.value -= ((v1.value<<4) + k0) ^ (v1.value + total.value) ^ ((v1.value>>5) + k1)
    total.value -= delta
  return v0.value, v1.value

def open_file(filename, chunk_size):
        """
        Opens a file as binary and puts its content
        into an array in which each array cell is
        chunk_size bits in hexadecimal form written
        as string.
        """
        with open(filename, "rb") as f:
            hex_array = []
            for offset in range(0, os.path.getsize(filename), 8):
                hex_array.append(bytes.hex(f.read(8)))
                f.seek(offset + 8)

            f.close()

        return hex_array


if __name__ == '__main__':
```

```
op_list= []
ip_list = []
result = ""
actstr = ""
prev = 0
op_list = open_file('/content/msg.txt', 64)
print(op_list)
op_list.pop()
#op_list[-1:] = op_list[-1:].zfill(16)
print(op_list)
K = 0xA56BABCD00000000FFFFFFFFABCDEF01
#print(type(K))
u = [c_uint32(((MASK32 << x) & K) >> x).value for x in range(96,-1,-32)]
l0 = (int(hex(u[0]), 16))
l1 = (int(hex(u[1]), 16))
l2 = (int(hex(u[2]), 16))
l3 = (int(hex(u[3]), 16))
  #key = [0xa56babcd, 0x0, 0xffffffff, 0xabcdef01 ]
key = [l0, l1, l2, l3]
  #P = 0x0123456789ABCDEF

IV = 0x182a7402d94f82ef
t = [c_uint32(((MASK32 << z) & IV) >> z).value for z in range(64,-1,-32)]
a0 = (int(hex(t[1]), 16))
a1 = (int(hex(t[2]), 16))
IVV = [a0,a1]
Ftime = True

for ih in op_list:
  P = int(ih, base=16)
  w = [c_uint32(((MASK32 << y) & P) >> y).value for y in range(64,-1,-32)]
  #for y in w: print(hex(y))
  #print(hex(w[1]),hex(w[2]))
  i0 = (int(hex(w[1]), 16))
  i1 = (int(hex(w[2]), 16))
  #print(i0,i1)
```

```
    v=[i0,i1]
    value = [i0,i1]
    if(Ftime):
      res = encrypt(value, key, IVV)
      result += (hex(res[0])[2:]).zfill(8)+(hex(res[1])[2:]).zfill(8)
      ip_list.append((hex(res[0])[2:]).zfill(8)+(hex(res[1])[2:]).zfill(8))
      Ftime = False
    else:
      res = encrypt(value, key, res)
      result += (hex(res[0])[2:]).zfill(8)+(hex(res[1])[2:]).zfill(8)
      ip_list.append((hex(res[0])[2:]).zfill(8)+(hex(res[1])[2:]).zfill(8))

    #c = hex(res[0])+hex(res[1])[2:]

  print(result)
  print(ip_list)

  f= open("msgcbc.txt.enc","w+")
  f.write(result)
  f.close()

  Fdtime = True
  for ch in ip_list:
    Q = int(ch, base=16)
    w = [c_uint32(((MASK32 << y) & Q) >> y).value for y in range(64,-1,-32)]
    #for y in w: print(hex(y))
    #print(hex(w[1]),hex(w[2]))
    i0 = (int(hex(w[1]), 16))
    i1 = (int(hex(w[2]), 16))
    #print(i0,i1)
    value = [i0,i1]
    if(Fdtime):
      res = (decrypt(value, key))
      pt0 = res[0]^IVV[0]
      pt1 = res[1]^IVV[1]
      pt = pt0, pt1
      prev = value
```

```
      actstr += (hex(pt[0])[2:]+hex(pt[1])[2:])
      #ip_list.append(hex(pt[0])[2:]+hex(pt[1])[2:])
      Fdtime = False
    else:
      res = decrypt(value, key)
      pt0 = res[0]^prev[0]
      pt1 = res[1]^prev[1]
      pt = pt0, pt1
      prev = value
      actstr += (hex(pt[0])[2:]+hex(pt[1])[2:])
      #ip_list.append(hex(pt[0])[2:]+hex(pt[1])[2:])
```

```
print(actstr)
ox = (bytes.fromhex(actstr).decode('utf-8'))
print(ox)
```

```
f1= open("msg.txt.cbcdec","w+")
f1.write(ox)
f1.close()
```

```
['466f75722073636f', '726520616e642073', '6576656e20796561', '72732061676f206f', '7572206661746865', '72732062726f7567', '68742
['466f75722073636f', '726520616e642073', '6576656e20796561', '72732061676f206f', '7572206661746865', '72732062726f7567', '68742
7d1233696d60c3b0825193e2763b317a8a59851cef0cec6594f03d5db56826905f9e6a1aefd6e19bd838edf0650fc9bdfbc64544ebff1cc7dbb3e2b9ad1d534
['7d1233696d60c3b0', '825193e2763b317a', '8a59851cef0cec65', '94f03d5db5682690', '5f9e6a1aefd6e19b', 'd838edf0650fc9bd', 'fbc64
466f75722073636f726520616e6420736576656e207965617273206167468657273206272675676870666f727468206f6e207468697
Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated t
```

```
qa = "{:x}".format(0x0a742c2061206e65)
print(qa)
hex(0x0a742c2061206e65).replace("L","").replace("0x","")
```

```
a = hex(2022)
print(a)
print(a[2:].zfill(4))
```

```
a742c2061206e65
0x7e6
07e6
```

Colab paid products - Cancel contracts here

✓ 0s completed at 13:35