Formative Assignment

Exercise 1:

Answer:

The output is 0x63636363636363636363636363636363

Code Implemented:

None

Further Explanation:

Starting initially with 128 0 bits or 32 hex digits for the input plaintext, k(0) and k(1). When the key addition takes place between the plaintext and k(0), due to the nature of XOR there will be no change between any of the values as they are not different, this then moves into the for loop where substitution takes place, all values will be replaced with 63, outputting:

0x63 0x63 0x63 0x63
0x63 0x63 0x63 0x63
0x63 0x63 0x63 0x63
0x63 0x63 0x63 0x63

This means that during the shift rows stage no changes will occur as all elements are the same.

For mix Columns the matrix shown below is used

0x02 0x03 0x01 0x01
0x01 0x02 0x03 0x01
0x01 0x01 0x02 0x03
0x03 0x01 0x01 0x02

0x63 as bits is: 01100011 or x^6+x^5+x + 1
0x01 as bits is: 00000001 or 1
0x02 as bits is: 00000010 or x
0x03 as bits is: 00000011 or x + 1

x ⊗ 0x01 = x
x ⊗ 0x02 = x << 1
x ⊗ 0x03 = x ⊕ (x ⊗ 0x02)

outputing

0xC6 0xA5 0x63 0x63 ⊕ = 0x63
0x63 0xC6 0xA5 0x63 ⊕ = 0x63
0x63 0x63 0xC6 0xA5 ⊕ = 0x63
0xA5 0x63 0x63 0xC6 ⊕ = 0x63

This means for all columns the output would still be 0x63. Finally, when XORd with the key of all zeros, the round 1 ciphertext will be:
0x63636363636363636363636363636363

Exercise 2:

Answer:

K(0) = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

K(1) = 0xE8E9E9E917161616E8E9E9E917161616

Code Implemented:

None

Further Explanation:

Completing Key Schedule

Initial Encryption key K = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

W0-3 = 0xFFFFFFFF

T = W3 <<< 8 = W3 (No change)

T = Subytes(T) = 0x16161616

T = T ⊕ Rci = 0x16161616 ⊕ 0x01000000 = 0x17161616

W4 = W0 ⊕ T = 0xFFFFFFFF ⊕ 0x17161616 = 0xE8E9E9E9

W5 = W1 ⊕ W4 = 0xFFFFFFFF ⊕ 0xE8E9E9E9 = 0x17161616

W6 = W2 ⊕ W5 = 0xFFFFFFFF ⊕ 0x17161616 = 0xE8E9E9E9

W7 = W3 ⊕ W6 = 0xFFFFFFFF ⊕ 0xE8E9E9E9 = 0x17161616

K1 = W4 +W5 + W6 + W7 = 0xE8E9E9E917161616E8E9E9E917161616

Exercise 3:

Answer:

Collision found
Number of tries made: 41465803901
Taking 76632.49720597267 seconds
Plaintext one is: 41465803901
Plaintext two is: 1871
The full hash for the first plaintext is:
7026d0f189b40a2bdb4758f9cd88d850d2516d51
The full hash for the first plaintext is:
7026d0f189b40a2ab2815aa592f49c2e0997bd3a
The part of the hash shared is: 7026d0f189b40a2

Code Implemented (Python):

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import hashlib
import random
#import string
import time

class Task_3:
    def __init__(self):
        self.implement()

    def get_random(stp): #Unused for final run
        strcount = 0
        strleng = random.randint(1,20) #Choose a random integer between 1 and
20
        output = []
        while strcount < strleng:
          output.append(random.choice(stp)) #Append on a random char from
StrPoss
          strcount+=1
        out = ''.join(output)
        return out

    def implement(self):
        #StrPoss = string.ascii_letters+string.digits #All possibile string characters
        Tries = 0 #Number of Tr
        diction = {}
        run = True
        start_time = time.time() #Calculates time taken to find collision
        InputIntoMem = True #Implimented in later i
        while run == True:
            Tries +=1
            #Input = Task_3.get_random(StrPoss) #Initially used but was drastically
slowing down speed
```

```
        Input = str(Tries)
        Hashed = hashlib.sha1(Input.encode("ascii")).hexdigest() #Hashes the
Input with SHA1
        Hashed = Hashed[0:15] # Cuts down the Hash to the length needed: 8
and 10 in testing, 15 final
        if (Tries % 1000000) == 0: #Used to keep reasonably track of how many
tries taken place
            print("Number of tries made so far" ,Tries)
            print(Input)
        if (Tries % 20000000) == 0: #Put into place to stop new input into RAM
            InputIntoMem = False
        if (Hashed not in diction.keys()) and (InputIntoMem == True):
                diction[Hashed] = Input
                #Was initially a memory error trycatch statement but removed as
not needed anymore
        elif (Hashed in diction.keys()) and (diction[Hashed] != Input):
            print ("Collision found")
            print("Number of tries made:" ,Tries)
            print("Taking %s seconds" % (time.time() - start_time))
            HashedInpOne = hashlib.sha1(Input.encode("ascii")).hexdigest()
            HashedInpTwo =
hashlib.sha1(diction[Hashed].encode("ascii")).hexdigest()
            print("Plaintext one is:", Input)
            print("Plaintext two is:", diction[Hashed])
            print("The full hash for the first plaintext is:",HashedInpOne)
            print ("The full hash for the first plaintext is:", HashedInpTwo)
            print ("The part of the hash shared is:", Hashed)
            break

if __name__ == "__main__":
    t3 = Task_3()
```

Further Explanation:

Initial implimentation for smaller finding collisions with smaller nibbles was completed using a random generator for words, this was found to be very inefficient, creating up to 20 random values to add to the output variable. The use of these strings also and allowed for the same value to be outputted several times and so wasted time. The incrementer tries was instead used as the number would be different every time and was already being implimented so could be used efficiently. The script takes a new hash each loop, cuts it down to 15 and compares it before storing in RAM, this occurs until there is 20 millions hashes when no new hashes are entered and the newer hashes are compared to this original 20 million. Eventually after 41 trillion tries a collision was found.

Exercise 4:

Answer:

Photo not uploaded

Code Implemented:

None

Further Explanation:

I am in the process of changing my name and so do not wish for my face to be associated with a name I do not wish to use for much longer.

Because this image will be accessible to all other people on the system, I do not want bad personal interactions or percieved slights that members of staff may believe exist to be able to be linked to the my account and grades they will be giving me for assessments. No matter how objective a lecturer may seem to or believe themselves to be, favouritism can still occur and I believe keeping the data as anonymous as possible is the best course of action.

Completing this task would also introduce even more data to a system that already holds vast amounts of my personal data, doing this (especially adding an image which can have a lot of information garnered from it) is likely going to increase the chances of de-anonymisation and exploitation of said data if a breach were to occur.

I do not believe this addition has enough positive effects to outweigh the privacy and security concerns it causes me.