ALEXCTF CR2: Many time secrets

ALEXCTF CR2: Many time secrets

This time Fady learned from his old mistake and decided to use onetime pad as his encryption technique, but he never knew why people call it one time pad!
Flag will start with ALEXCTF{.

https://mega.nz/#!DGxBjaDR!tMWkHf0s0svmkboGd-IASHsS9jACxSYx4zi_ETsyzyQ

```
0529242a631234122d2b36697f13272c207f2021283a6b0c7908
2f28202a302029142c653f3c7f2a2636273e3f2d653e25217908
322921780c3a235b3c2c3f207f372e21733a3a2b37263b313012
2f6c363b2b312b1e64651b6537222e37377f2020242b6b2c2d5d
283f652c2b31661426292b653a292c372a2f20212a316b283c09
29232178373c270f682c216532263b2d3632353c2c3c2a293504
613c37373531285b3c2a72273a67212a277f373a243c20203d5d
243a202a633d205b3c2d3765342236653a2c7423202f3f652a18
2239373d6f740a1e3c651f207f2c212a247f3d2e65262430791c
263e203d63232f0f20653f207f332065262c3168313722367918
2f2f372133202f1426652126372222220733e383f2426386b
```

Flag: CTFlearn{HERE_GOES_THE_KEY?}

Writeup:

created python program and got flag:

```
import string
import collections
import sets, sys

# 11 unknown ciphertexts (in hex format), all encrpyted with the same key
c1 = "0529242a631234122d2b36697f13272c207f2021283a6b0c7908"
c2 = "2f28202a302029142c653f3c7f2a2636273e3f2d653e25217908"
c3 = "322921780c3a235b3c2c3f207f372e21733a3a2b37263b313012"
c4 = "2f6c363b2b312b1e64651b6537222e37377f2020242b6b2c2d5d"
c5 = "283f652c2b31661426292b653a292c372a2f20212a316b283c09"
c6 = "29232178373c270f682c216532263b2d3632353c2c3c2a293504"
c7 = "613c37373531285b3c2a72273a67212a277f373a243c20203d5d"
c8 = "243a202a633d205b3c2d3765342236653a2c7423202f3f652a18"
c9 = "2239373d6f740a1e3c651f207f2c212a247f3d2e65262430791c"
c10 = "263e203d63232f0f20653f207f332065262c3168313722367918"
c11 = "2f2f372133202f1426652126372222220733e383f2426386b"
ciphers = [c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11]

# The target ciphertext we want to crack
target_cipher = "2239373d6f740a1e3c651f207f2c212a247f3d2e65262430791c"

# XORs two string
def strxor(a, b):     # xor two strings (trims the longer input)
    return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b)])

# To store the final key
final_key = [None]*150
# To store the positions we know are broken
known_key_positions = set()

# For each ciphertext
for current_index, ciphertext in enumerate(ciphers):
        counter = collections.Counter()
        # for each other ciphertext
        for index, ciphertext2 in enumerate(ciphers):
                if current_index != index: # don't xor a ciphertext with itself
                        for indexOfChar, char in enumerate(strxor(ciphertext.decode('hex'),
ciphertext2.decode('hex'))): # Xor the two ciphertexts
                                # If a character in the xored result is a alphanumeric character, it means
there was probably a space character in one of the plaintexts (we don't know which one)
                                if char in string.printable and char.isalpha(): counter[indexOfChar] += 1 #
Increment the counter at this index
        knownSpaceIndexes = []

        # Loop through all positions where a space character was possible in the current_index cipher
        for ind, val in counter.items():
                # If a space was found at least 7 times at this index out of the 9 possible XORS, then the space
character was likely from the current_index cipher!
                if val >= 7: knownSpaceIndexes.append(ind)
        #print knownSpaceIndexes # Shows all the positions where we now know the key!

        # Now Xor the current_index with spaces, and at the knownSpaceIndexes positions we get the key back!
        xor_with_spaces = strxor(ciphertext.decode('hex'),' '*150)
        for index in knownSpaceIndexes:
                # Store the key's value at the correct position
                final_key[index] = xor_with_spaces[index].encode('hex')
                # Record that we known the key at this position
                known_key_positions.add(index)

# Construct a hex key from the currently known key, adding in '00' hex chars where we do not know (to make a
complete hex string)
final_key_hex = ''.join([val if val is not None else '00' for val in final_key])
# Xor the currently known key with the target cipher
output = strxor(target_cipher.decode('hex'),final_key_hex.decode('hex'))

print("Fix this sentence:")
print(''.join([char if index in known_key_positions else '*' for index, char in enumerate(output)])+"\n")

# WAIT.. MANUAL STEP HERE
```

```
# This output are printing a * if that character is not known yet
# fix the missing characters like this: "Let*M**k*ow if *o{*a" = "cure, Let Me know if you a"
# if is too hard, change the target_cipher to another one and try again
# and we have our key to fix the entire text!

#sys.exit(0) #comment and continue if u got a good key

target_plaintext = "cure, Let Me know if you a"
print("Fixed:")
print(target_plaintext+"\n")

key = strxor(target_cipher.decode('hex'),target_plaintext)

print("Decrypted msg:")
for cipher in ciphers:
        print(strxor(cipher.decode('hex'),key))

print("\nPrivate key recovered: "+key+"\n")
```

OUTPUT:

```
Fix this sentence:
cure, Let*M**k*ow if *o{*a

Fixed:
cure, Let Me know if you a

Decrypted msg:
Dear Friend, This time I u
nderstood my mistake and u
sed One time pad encryptio
n scheme, I heard that it
is the only encryption met
hod that is mathematically
 proven to be not cracked
ever if the key is kept se
cure, Let Me know if you a
gree with me to use this e
ncryption scheme always.

Private key recovered: ALEXCTF{HERE_GOES_THE_KEY}
```