Old cryptography

CTF: 0ctf 2015 (/challenges/ctf/0ctf%202015/)
Category: Cryptography (/challenges/category/Cryptography/)
Write-up *by* ***GSAir*** *on March 29, 2015, 9:54 p.m.*

Old Cryptography

Challenge Description

Old crypto is not old enough to be broken. Notice: all in lowercase

Material

1. A python script used to encrypt and decrypt message.
2. A ciphertext of 1583 letters

Solution

- First looking at the encryption function, we notice it is poly-alphabetic substitution cipher (`tr[k][p]`) with a non-uniform shift (`+ i ** 7`).

  ```
  def encrypt(plaintext, key):
      ciphertext = ""
      for i in xrange(len(plaintext)):
          p = ord(plaintext[i]) - ord('a')
          k = ord(key[i % len(key)]) - ord('a')
          c = (tr[k][p] + i**7) % 26
          ciphertext += chr(c + ord('a'))
       return ciphertext
  ```

- There are 26 different permutations, and the key tells the algorithm which one to choose:

> If the key is "hello", the 8th permutation is used for the 1st, 6th, 11th ... letters, the 5th permutation is used for the 2nd, 7th ... letter etc.

- The shift is quite useless and can be removed immediately in order to concentrate on the difficult part.

```
def normalize(ciphertext):
    nciphertext = ""
    for i in xrange(len(ciphertext)):
        c = ord(ciphertext[i]) - ord('a')
        p = (c - i**7) % 26
        nciphertext += chr(p + ord('a'))
    return nciphertext
```

- The cipher can now be broken by frequency analysis (FA)because the ciphertext is long enough :). On a poly alphabetic cipher, we need to guess the length of the key in order to apply the FA. This cipher is just a little more complex than the Vigenere cipher, but has the same vulnerability, using Kasiski analysis gives us information on the key length. [1][http://crypto.interactive-maths.com/kasiski-analysis-breaking-the-code.html]. The most likely key length is 20, or a divisor of 20.

Steps

Groups the letters

We groups the letter which are encrypted with the same substitution

```
def split(ciphertext, keylength):
    l = [None] * keylength
    for i in xrange(len(ciphertext)):
        if (l[i%keylength] == None):
            l[i%keylength] = []
    l[i%keylength].append(ciphertext[i])
    return l
```

Apply Frequency analysis one every sets

For each set we try the 26 possible substitutions and we choose the letter which makes the frequency (of the letter in the deciphered set) match the closest to the English frequency distribution.

```
def freqanalysis(group):
    l = [0] * 26
    for i in xrange(len(group)):
        c = ord(group[i]) - ord('a')
        l[c] = l[c] + 1
    s = float(sum(l))
    l = [float(n)/s for n in l]
    return l

# true frequency of the English language
truefreq = [0.08167, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015, 0.0609
4, 0.06966, 0.00153, 0.00772, 0.04025, 0.02406, 0.06749, 0.07507, 0.01929, 0.0009
5, 0.05987, 0.06327, 0.09056, 0.02758, 0.00978, 0.02360, 0.00150, 0.01974, 0.0007
4]

# perfect match
pmatch = 0.065

def freq(group, i):
    # decipher letter
    plain = [chr(tr[i][ord(n) - ord('a')] + ord('a')) for n in group]
    # make frequency analysis
    fr = freqanalysis(plain)

    # compute the match "criterion"
    ret = 0
    for j in xrange(26):
        ret += truefreq[j] * fr[j]

    return ret

def findkeyletter(group):
    freqs = [abs(pmatch - freq(group, n)) for n in xrange(26)]
    return freqs.index(min(freqs))
```

Try

Now we can try with the key length of 20 or a divisor of 20. But like the frequency analysis is very fast (linear in the size of the ciphertext and the key length), we can just try all values between 0 and 20 and even more if we don't find the solution:

```
def decryptnormalize(ciphertext, key):
    plaintext = ""
    for i in xrange(len(ciphertext)):
        c = ord(ciphertext[i]) - ord('a')
        k = ord(key[i % len(key)]) - ord('a')
        p = tr[k][c]
        plaintext += chr(p + ord('a'))
    return plaintext

f = open('ciphertext', 'r')
ciphertext = f.read()

nciphertext = normalize(ciphertext)

for guessedkeylength in [4,5,10,20]:
    sp = split(nciphertext, guessedkeylength)

    key= ""
    for i in xrange(guessedkeylength):
        letter = chr(findkeyletter(sp[i]) + ord('a'))
        key += letter

    print "key: " + key
    print "plaintext: " + decryptnormalize(nciphertext, key)
```

The key length was 20:

```
key: classicalcipherisfun
plaintext: incryptanalysis...(lot of text)...ooooooooooooooooooooopsflagisthekeywit
hoopsprefixandbraces
```

Flag 0ctf{classicalcipherisfun}