James Chad Ballay

**Week 8 Cipher Lab**

**Encrypted Text**

|  |
| --- |
| Svwfiv pfl svtfdv kff vekiretvu nzky xfixvflj xruxvkj reu dvjdvizqzex mzuvf uzjgcrpj, cvk dv ivdzeu pfl kyrk zewfidrkzfe zj efk befncvuxv, befncvuxv zj efk nzjufd, reu nzjufd zj efk wfivjzxyk. Vrty xifnj flk fw kyv fkyvi, reu nv evvu kyvd rcc.  Rikyli T. Tcribv |

**Decrypted Text**

|  |
| --- |
| BEFORE YOU BECOME TOO ENTRANCED WITH GORGEOUS GADGETS AND MESMERIZING VIDEO DISPLAYS, LET ME REMIND YOU THAT INFORMATION IS NOT KNOWLEDGE, KNOWLEDGE IS NOT WISDOM, AND WISDOM IS NOT FORESIGHT. EACH GROWS OUT OF THE OTHER, AND WE NEED THEM ALL.  ARTHUR C. CLARKE |

**Key**

9

**Learnings**

Not to be blunt but I would make a poor cryptoanalyst without automated tools. I didn’t find elegant solutions but I am able to think of methodical approaches. It takes a special, devious thinker to keep track of all the threads of possibility and work through them. With access to tools it compensated for my lack of focus. Much of the by hand analysis is trying many many false trails and sometimes going 2, 3 or even 4 branches deep on a trail before rolling back.

**First solution**

I didn’t read close enough and dove right into writing a quick shell script. Finished it and then realized that wasn’t quite the way you wanted us to solve this. However, it is close enough that you may appreciate the way I did it. The gist is that I take the encrypted string and iterate through the 26 shifts in the alphabet for the key. Then for the resulting 26 versions of decrypted text, I break each one into tokens by the white space. For each of those tokens I see if that token exists in the dictionary. The idea being that if this specific key was correct it would generate real words and those words would be in the dictionary. All other keys would be incorrect and not generate real english words. Then for each of these 26 iterations I add up the count of real words and pick the iteration that has the highest. I’ll attach the shell script at the end after I go over my second solution. It isn’t pretty or polished, I but it solves it.

**Second Solution**

After re-reading the instructions, I attacked this again.

First let’s get a frequency of characters. From this we see V is a likely candidate for being an E. (Most common letter.)



Next I looked for what was the most common letter beginning a word since that would likely be a T. This didn’t prove to be successful since there was a tie for first place among 4 letters.



Looking at the end of the word boundry also didn’t help. Another 4 way tie for what could be the letter E.



Starting with the idea that V could be the letter E, lets start. (Going to use lowercase for encrypted and UPPERCASE for decrypted.)

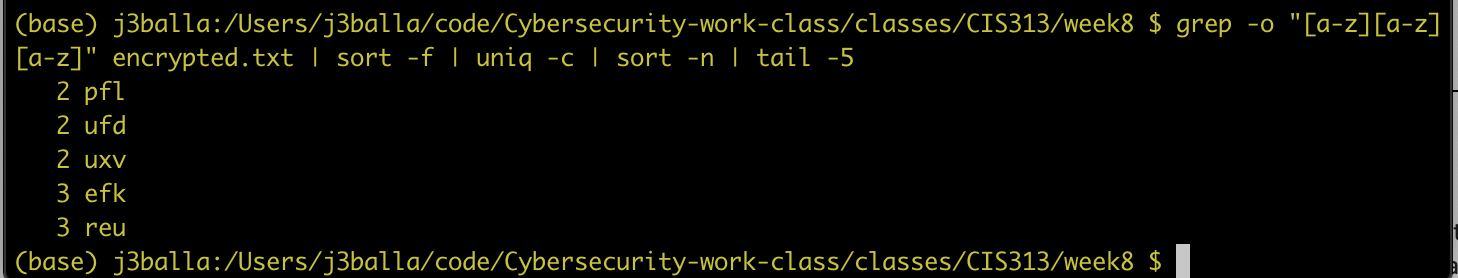
|  |
| --- |
| svwfiv pfl svtfdv kff vekiretvu nzky xfixvflj xruxvkj reu dvjdvizqzex mzuvf uzjgcrpj, cvk dv ivdzeu pfl kyrk zewfidrkzfe zj efk befncvuxv, befncvuxv zj efk nzjufd, reu nzjufd zj efk wfivjzxyk. vrty xifnj flk fw kyv fkyvi, reu nv evvu kyvd rcc.    rikyli t. tcribv |

Becomes

|  |
| --- |
| sEwfiE pfl sEtfdE kff EekiretEu nzky xfixEflj xruxEkj reu dEjdEizqzex mzuEf uzjgcrpj, cEk dE iEdzeu pfl kyrk zewfidrkzfe zj efk befncEuxE, befncEuxE zj efk nzjufd, reu nzjufd zj efk wfiEjzxyk. Erty xifnj flk fw kyE fkyEi, reu nE eEEu kyEd rcc.    rikyli t. tcribE |

Next up let’s make some guess on short words. There aren’t that many two letter words and lookng at the ones with E narrows down the guesses. dE and nE are the only two. Be, He, Me and We. Again a dead end but still something to remember.

Next I looked for three letter groupings wherever they occurred in the text. Pulled those out and tried to sort and count on them.



At this point I think start going off of the v->E and shift other letters as well. The info we were given says it’s a Caeser cipher. So r->A, s->B, t->C, etc, etc.... 9 spaces on the alphabet for all. This reveals the text but I unfortunately don’t know if this is just a happy path chosen since I already know the answer from my first attempt.

|  |
| --- |
| #!/bin/bash    tmpfile=$(mktemp)    # Read in cmd line arguments as text to decode. Shift to uppercase  # and then make sure there is input.  INPUT=$\*  INPUT=${INPUT^^}  if [ ${#INPUT} -le 0 ];  then  echo "Input blank"; exit  fi    # Technically someone could provide ROT-26 encrypted text...  for KEY in $(seq 26); do  # Take the string. Use tr to shift it forward by the iterator.  STRING=`echo $INPUT | tr $(printf %${KEY}s | tr ' ' '.')\A-Z A-ZA-Z`    # Count the number of "real" words in the decrypted string.  # To do this we'll split it by whitespace and then check the  # standard english dictionary for that word. If it's there  # we increment the counter. At the end we write the count  # to the tmp file.  WORD\_COUNT=0  for TOKEN in $STRING  do  let "WORD\_COUNT=WORD\_COUNT+`grep -ci "^$TOKEN$" /usr/share/dict/words`"  done  echo "$WORD\_COUNT $STRING" >> $tmpfile  done  # The string with the highest word count is probably our decrypted text  # so print that one.  sort -k1 -n $tmpfile | tail -1  rm $tmpfile |