Homework #3B Solutions - CIS 3362

1) (30 pts) The following ciphertext was encrypted using the Playfair cipher. The first eight letters of the plaintext are "themainr". Determine the secret key and decrypt the whole ciphertext. (Note: I may choose to reveal more of the plaintext, but I haven't made that decision yet.)

```
qknvbdradnrlbexrbmzilqkzenqkpbgqvnzwbgbnlazkfvnfybblqhybqhenoqnzzspdpnl
qkzenlwybkvlwbaimfdkeanliznrlbebmfdlubnlanzhaebkokdblfdsfmrybsvzitzrfrl
nihspczpyzmtosnednmlfdmqhsybmrmgnhnzmdglizrttqchqlqgpfqknzmdgmwskeenwtv
xpttmnpmdbnptenarqlowqpnffbzkkqbpybbmfdogdmnffkbdsgmqnpbltlqlfdfbzkkqmq
tqcyaeznfnbqkqqhlkqhzbbrixlwyzmtmcebiqfdrttqht
```

. . . . . .

We begin the decryption process by mapping the given ciphertext-to-plaintext letters and looking for any ways to fill out the 5x5 Playfair key:

Before filling in the key at all, we can also notice that the given plaintext "themainr" is likely "themainreason", and so we can extrapolate a few more plaintext-to-ciphertext pairs from this (marked with an *).

Corresponding plaintext-to-ciphertext:

th → qk
em → nv
ai → bd
nr → ra
ea → dn *
so → rl *

One prominent plain-to-cipher pair is nr → ra, as this tells us that `n`, `r`, and `a` must share a row or column by the connecting `r` in both the plain & cipher pairs. We can also likely deduce that these letters share a row rather than a column, as any combination of `n`, `r`, and `a` in a column would contradict the alphabetical ordering of the Playfair grid after the keyword. One spot the letters could go is as follows:

n	r	a		

Where `n` and `r` are likely part of the keyword and `a` is the beginning of the alphabetical ordering past the keyword. We can now examine the `ea` → `dn` pair - knowing the position of both the `a` and `n` allow us to deduce this is a box-rule mapping, and so we know that `e` must share a row with `d` and a column with `n`. The only possible mapping for this is as follows:

e		d		
n	r	a		

Next, we can consider the `em` → `nv` mapping. Since `e` and `n` share a column, we deduce that this is a column-rule mapping, and so one likely ordering for this mapping is:

e		d		
n	r	a		
m				
v				

as `v` resides near the end of the Playfair grid in alphabetical order (likely not a part of the key). This also allows us to fill in the last 4 cells in the last row, as `w` - `z` are the only 4 letters after `v`, meaning they almost certainly form this last row:

e		d		
n	r	a		
m				
v	w	x	y	z

Next we consider `ai` → `bd`. As we know where `a` and `d` are, we again deduce this as a box-rule mapping, and considering `b` must be on the same row as `a`, we can assume that `b` immediately follows `a` in the grid (as it is not in the keyword and thus must follow alphabetical ordering):

e		d	i	
n	r	a	b	
m				
v	w	x	y	z

This mapping also allows us to fill in the cells from `b` to `m`, as the remaining letters between these two cells match the letters not used in the keyword, resulting in:

e		d	i	
n	r	a	b	c
f	g	h	k	l
m				
v	w	x	y	z

Next we consider `th` $\rightarrow$ `qk`. Again, we deduce this mapping as a box-rule as we know the positions of `h` and `k`, and so fill in:

e		d	i	
n	r	a	b	c
f	g	h	k	l
m		q	t	
v	w	x	y	z

At this point we've run out of mappings to analyze, and so we can take a guess as to the remaining 4 cells to complete the grid:

e	s	d	i	p
n	r	a	b	c
f	g	h	k	l
m	o	q	t	u
v	w	x	y	z

Even though this is almost certainly not the fully-correct Playfair key, it will likely give us some indication of how we're doing with regards to breaking this cipher. Plugging in this key int oa Playfair decrypter gives us:

```
themainreacgniwantyphulyvethichomeyvrkachcylnmentickhatihavemocvwpisec
hulyvegztifygzarethefirckpvccgnintheclachccvadingthickheegontiewypuyng
cgbegdzpuzxyuqgweveaufheuogdtionofafcvqefkpyboqoaluhohelthcvqefoowfive
yozwiuquceqeaciuvernuhgoudenknylhtcitinthegreqenlhaiwruoceckukuhheknyl
htuoqobzndvcneaththakhhaycandygzxyuqunindtheboqokq
```

As there's definitely some readable plaintext here ("themainrea...." and ..."theboqokq"), we're likely almost at our key. One thing that we could try is changing the initial position of the `n r a` row, as there's no guarantee we had that position initially correct (this is also supported by the fact that our `so` → `rl` mapping is not possible in our current key). Since there's good chunks of readable plaintext, we don't want to change *too* much, and so moving the `n r a` once to the right gives us another good starting point to build our key:

	n	r	a	

Using the `em` → `nv` as before gives us:

	e			
	n	r	a	
	m			
	v			

Using the `ea` → `dn` and `ai` → `bd` mappings give us:

	e		d	i
	n	r	a	b
	m			
	v			

Filling in between `b` and `m`...

	e		d	i
	n	r	a	b
c	f	g	h	k
l	m			
	v			

The `th` → `qk` mapping gives us:

	e		d	i
	n	r	a	b
c	f	g	h	k
l	m		q	t
	v			

Filling in the `o` between m and `q` and the `u` between `t` and `v`, as well as the `so` $\rightarrow$ `rl` mapping gives us:

	e		d	i
s	n	r	a	b
c	f	g	h	k
l	m	o	q	t
u	v			

Running out of mappings, we can guess the last 5 boxes and see if any extra mappings arise. One such key is:

z	e	p	d	i
s	n	r	a	b
c	f	g	h	k
l	m	o	q	t
u	v	w	x	y

Which give us the plaintext

```
themainreasoniwantidtocivethirhomepurkasqsicnmentisthatihavemoseuzeper
tociveoutifyouarethefirstzessonintheclasqsseadingthisthencontinuidling
sobecazgieuilqlrevealthelocationofafseqecodiboqokgotohegthseqefourfive
youwioqlreqeasiovernotgoodenknichtritinthegreqenchairclorestqtotheknic
htloqokundesneaththatchaisandyouuilqlfindtheboqokq
```

In the plaintext, the phrase "asqsicnment" is likely "assignment", and so by finding out the location of these letters in the ciphertext, we can come up with one final mapping:

`ig` $\rightarrow$ `zk`

Plugging this into our penultimate unfinished key gives us:

p	e	z	d	i
s	n	r	a	b
c	f	g	h	k
l	m	o	q	t
u	v	w	x	y

Plugging in this key into a Playfair decrypter now results in the true plaintext:

```
themainreasoniwantedtogivethishomeworkasqsignmentisthatihavemoreprizes
togiveoutifyouarethefirstpersonintheclasqsreadingthisthencontinuedoing
sobecauseiwilqlrevealthelocationofafreqecodeboqokgotohecthreqefourfive
youwilqlseqeasilvernotgoldenknightsitinthegreqenchairclosestqtotheknig
htloqokunderneaththatchairandyouwilqlfindtheboqokq
```

With the key above derived from "PEZDISPENSER".

---

2) (20 pts) The following question is more challenging, and I don't necessarily expect a lot of students to get it (which is why I have it worth as fewer points). Here is how I created this ciphertext:

a) Applied a substitution cipher to the plaintext and created an output I'll call c_1.
b) Applied a 2 x 2 Hill encryption key to c_1 to produce the final output.

The way I believe this can be broken is as follows:

1. Try all possible 2 x 2 matrices (with valid inverses).

2. For each, apply them to the ciphertext and just gather the letter frequencies of the resulting output, one of which, is guaranteed to be c_1. If we got the correct Hill key, we would expect these frequencies to mirror those of English. To detect this automatically, run the index of coincidence between each resulting output and English and rank as candidates for c_1 the ones that create the maximal index of coincidence with English. Then, take the top candidate, and use regular techniques to break substitution on it. Also, if you don't see any repeated n-grams that are kind of long, it's not going to be the answer, because I'll place at least one long repeated word in the plaintext.

For credit, I'll give a full 10 points for just generating all Hill keys, and another 5 points for calculating the corresponding frequencies for the supposed c_1's. Thus, even without cracking this one, you can get 15 out of 20 points and 45 out of 50 total.

```
pdyezowxbdlaprhualudbdgbdnpqgdnjwkhatovbgcvipqsutplfspooaxkwvakvukyjuuc
kihyidwxzbypuprwejkahdwtqplnhphhgfvdohvhulfbypuxcjrfwzgvvjtqkdnwkzmefdl
tczmmogauwpumfyeiazrmblwhvpvwnexzowxbdchtphupqmhimqqmpidmbhulgmoqmbbnmv
adjmfyvprpubdlsqaillhcirrebxhidxkphiwoqzgozofwjrxzfjtwhihupqmhimqqmpwx
hallnkfiukbkdxihjwkvsozodxtwxkxzaivyrdzovyhaewexdnzobcweezpqtywxkpinimw
khpyelfhfdjvu
```

.....

Ok, so this is definitely a tough one to crack. But as long as we take it one step at a time, it's doable.

Step 1: Finding c_1

In order to do this, we have to try every single possible 2x2 key matrix that could be used for Hill. In essence, this means we try all keys $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$, with every possible combination of numbers for a - d in the range of [0, 25]. We can achieve this fairly easily in a program using 4 nested for-loops running from 0-25, and then using each for-loop's counting variable as a key entry.

Next, we have to eliminate all non-valid keys from the set we just created. We can do so by calculating the determinant of each key (i.e. $ad - bc$) and then checking if that determinant has an inverse with mod 26.

Lastly, we have to "decrypt" the given ciphertext with every valid key we calculated, and then run an Index of Coincidence (IoC) on each resulting output. The Hill cipher decrypter code is just a standard Hill decryption and can be seen in **hillWithIOC.py**, but the logic for finding the IoC for each output is as follows:

1) Store how many times each letter shows up in the output
2) Use the formula $\Sigma_{i=0}^{26} \frac{f_i(f_i-1)}{n(n-1)}$, where f_i is the frequency of the i^{th} character and n is the length of the output
3) Print out the key, IoC, and plaintext for an output if its IoC is close to English (≈ 0.0676)

After doing all this, you might find this output:

```
IoC: 0.070652  Key: 25 21 23 20
wpiijxtekfptejxpxmrakfpckrjpfqulgetfppajkwrtjpmikjqplmuepskufppaikjekcamjmokrpipjjprejkotrrkrpfditexcrpvupflsjxpqpjjprtx
kbpjxtetapmwkrgetjxkudxjtjwkuiaqprpmiiscufjkjrsjklrmlgjxtekfpekjxpjpfakiimredkjkxpljwkckurftfpeppmbkejprkfdpkepomfjtlepd
opfjmjtkfjmgpkccjxpjkbipcjbtfjxpjpfakiimretetfmfpfhpikbpmjjmlxpaqsjxmjbtfbipmepbujjxpbtfqmlgkrjxplkobujprhtetkfckigewtii
qpubepjz
```

which turns out to be the correct Hill key!

Step 2: Substitution

Now that we have found c_1, we can begin working on the substitution cipher using the BHCSI Cryptanalysis Tool:

On using the Letter Frequency and N-Grams tools, we get the following output:

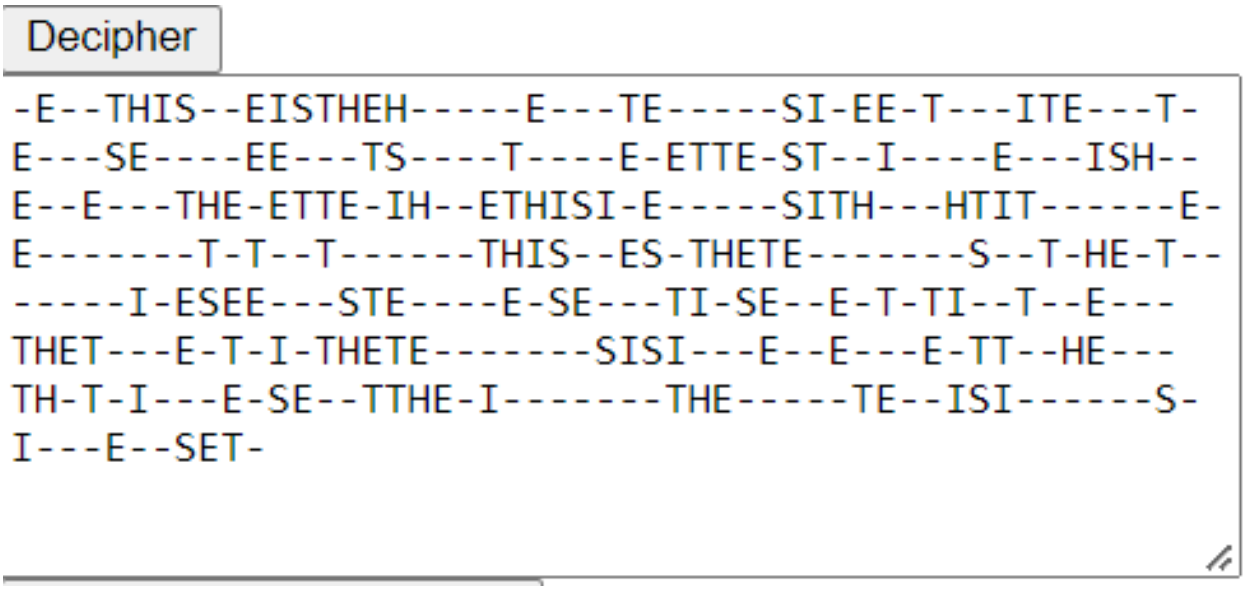
A	2.4%	N	0%
B	3%	O	1.4%
C	2.4%	P	14.4%
D	1.4%	Q	1.9%
E	6%	R	5.4%
F	6.5%	S	1.4%
G	1.6%	T	6.3%
H	0.5%	U	3%
I	5.2%	V	0.3%
J	12.2%	W	1.6%
K	9.8%	X	4.9%
L	2.7%	Y	0%
M	5.4%	Z	0.3%

Find Repeated N-grams

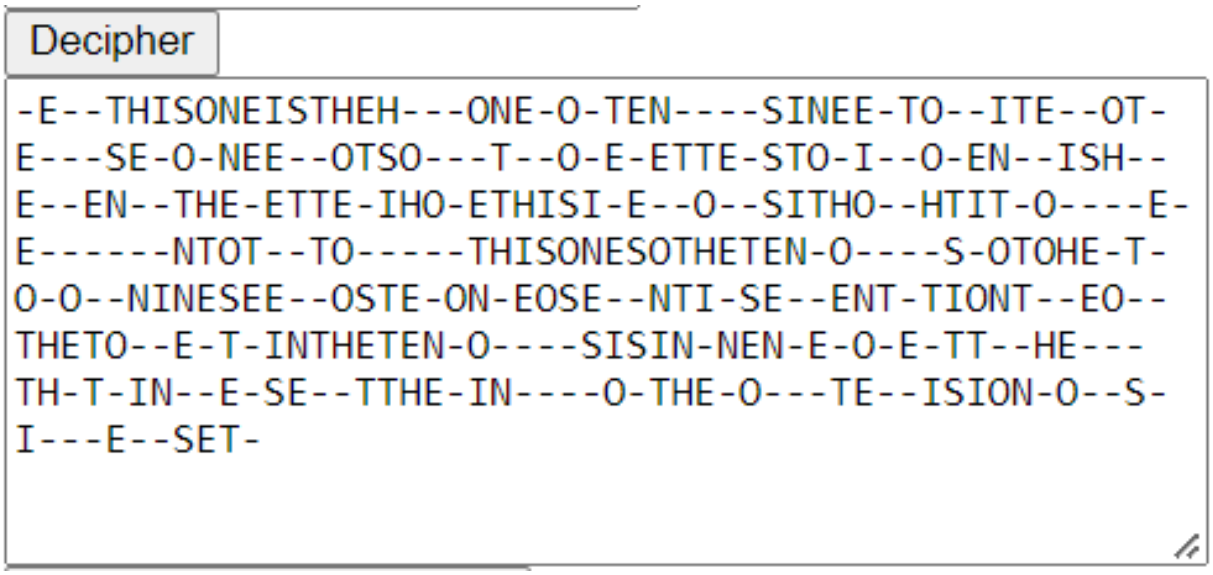
```
JXTE JXTEKFP JXP KRJ JPF GET ETF TFP FPPA PMI IKJ
KRP PJJPR SJX KBP TET JWK MLG FPE JXPJ JXPJPFAKIIMRE
XPL FJM TKF BIP JBTF BUJ
```

Analyzing this information we see that JX appears multiple times in the N-grams along with JXP; along with the letter frequency information showing J and P as high frequency letters, we make the assumption that JXP = THE.

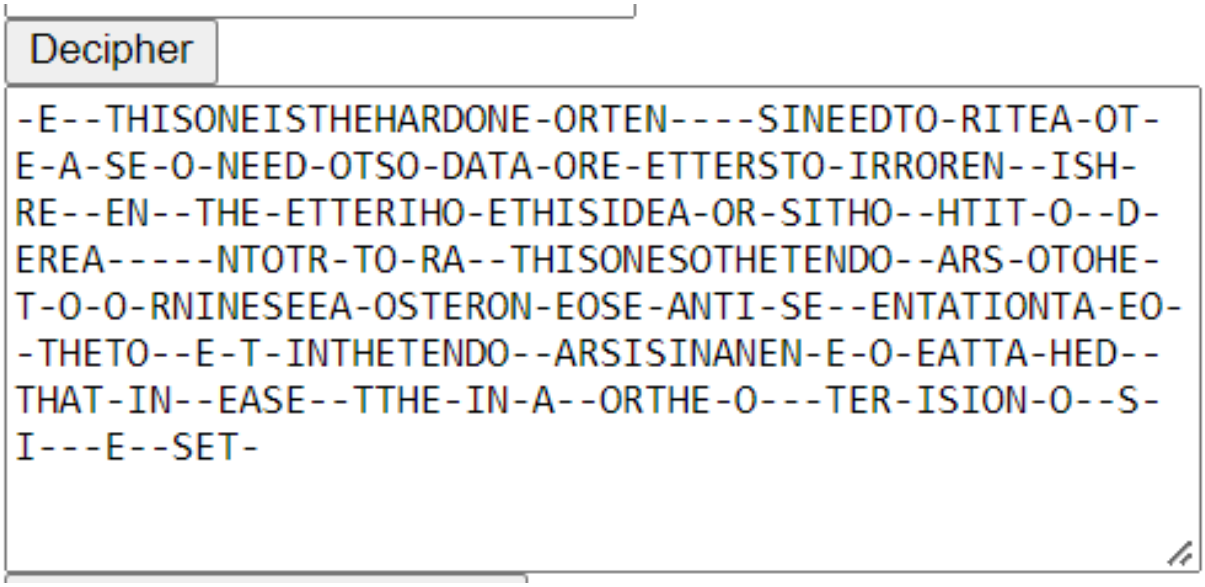
Working off this assumption JXTE becomes TH– and JXTEKFP becomes TH—-E. Analyzing JXTE and the corresponding letter frequencies, we can make a guess that JXTE = THIS. Plugging in our guesses so far gives us:



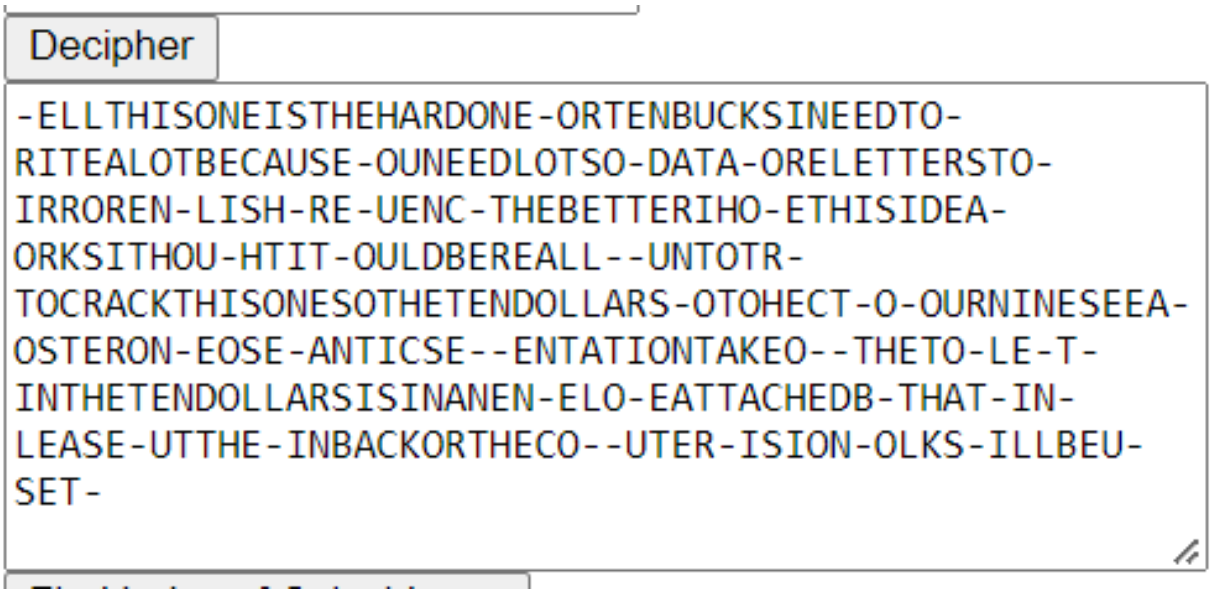
In analyzing the phrase "THIS–EISTHE", there's a fair chance that this plaintext is saying "THISONEISTHE", and so we can now map O and N into the substitution cipher:



In continuing to analyze the phrase "THISONEISTHEH—ONE", the incomplete word "H—" is likely "HARD", making the phrase "THISONEISTHEHARDONE". Mapping A, R, and D into the substitution cipher gives:



Now we've only got a couple more phrases to analyze until we reach our completed key! Within the pseudo-plaintext, we see references to "TENDO–ARS" which likely refers to "TENDOLLARS" (the prize!). There's also a reference to "TEN—–S" on the first line, which also likely refers to the prize, and so we can deduce that this is some synonym for dollars such as.... "BUCKS"! Plugging in these mappings gives us:



From this point, it's just a matter of using the existing plaintext to deduce the last remaining bits of the key before reaching our final message of:

WELL THIS ONE IS THE HARD ONE FOR TEN BUCKS I NEED TO WRITE A LOT BECAUSE YOU NEED LOTS OF DATA MORE LETTERS TO MIRROR ENGLISH FREQUENCY THE BETTER I HOPE THIS IDEA WORKS I THOUGHT IT WOULD BE REALLY FUN TO TRY TO CRACK THIS ONE SO THE TEN DOLLARS GO TO HEC TWO FOUR NINE SEE A POSTER ON GEOSEMANTIC SEGMENTATION TAKE OFF THE TOP LEFT PIN THE TEN DOLLARS IS IN AN ENVELOPE ATTACHED BY THAT PIN PLEASE PUT THE PIN BACK OR THE COMPUTER VISION FOLKS WILL BE UPSET Z

With that last 'Z' character acting as a padding-character for the Hill cipher.

---

hillWithIOC.py

```
# Array of all numbers w/ inverses for mod 26
mod_inverses = [1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25]

# Checks if a value <det> has an inverse mod 26 and return True/False
def has_inverse(det):
    if det in mod_inverses:
        return True
    else:
        return False

# Generates a 2x2 array with <a>, <b>, <c>, and <d>
def generate_key(a, b, c, d):
    key = [[a, b], [c, d]]
    return key

# Takes a String <plain> and finds its Index of Coincidence (IoC)
def get_ioc(plain):
    res = 0.0
    alphabet = [0] * 26
    length = len(plain)

    for c in plain:
        ind = ord(c) - ord('a')
        alphabet[ind] += 1

    for freq in alphabet:
        res += freq * (freq - 1)

    res = res / (length * (length - 1))

    return res

# Takes a String <cipher> and decrypts it with <key>
# Assumes <cipher> is even length and <key> is a 2x2 matrix
def hill_decrypt(cipher, key):
    # Store plaintext in <plain>
    plain = ""
    length = len(cipher)
```

```
    # Breaks up <cipher> into digrams and applies <key> to each
    for i in range(0, length, 2):
        # Calculate plaintext chars for <plain> using <key>
        d1 = ord(cipher[i]) - ord('a')
        d2 = ord(cipher[i+1]) - ord('a')
        c1 = chr((key[0][0] * d1 + key[0][1] * d2) % 26 + ord('a'))
        c2 = chr((key[1][0] * d1 + key[1][1] * d2) % 26 + ord('a'))

        plain += (c1 + c2)

    return plain

cipher = <FROM QUESTION 2>

for a in range(26):
    for b in range(26):
        for c in range(26):
            for d in range(26):
                # Calculate determinant mod 26
                det = (a * d - b * c)
                if (det < 0):
                    det += 26
                if has_inverse(det):
                    key = generate_key(a, b, c, d)
                    plain = hill_decrypt(cipher, key)
                    ioc = get_ioc(plain)

                    # Find IoCs close to English
                    if (ioc > 0.06):
                        print("IoC: {:.6f}".format(ioc))
                        print(Key: {:d} {:d} {:d} {:d}".format(a, b, c, d))
                        print(plain)
                        print()
```

Homework 3B Solutions Addendum

1) If you visit my (Arup) office, you notice that I have an affinity for Pez dispensers…so had you come to my office, you might have been able to guess the key word once a couple letters were potentially in place!

2) According to the TA who did the solutions (Josh), here was the output he received when running his program screening for potential Hill keys:

IoC: 0.060775 Key: 1 2 1 5
IoC: 0.070652 Key: 1 2 9 19
IoC: 0.060775 Key: 1 5 1 2
IoC: 0.070652 Key: 1 5 3 6
IoC: 0.070652 Key: 3 6 1 5
IoC: 0.060775 Key: 9 18 9 19
IoC: 0.070652 Key: 9 19 1 2
IoC: 0.060775 Key: 9 19 9 18
IoC: 0.060775 Key: 17 7 17 8
IoC: 0.060775 Key: 17 8 17 7
IoC: 0.070652 Key: 17 8 23 11
IoC: 0.070652 Key: 23 11 17 8
IoC: 0.070652 Key: 23 20 25 21
IoC: 0.070652 Key: 25 21 23 20

My hint was based on his output. Some students said that their program identified a key that comes after (25 21 23 20) lexicographically.

The reality is that in the "real world" no problems will give you hints, and while most problems themselves won't be adversarial, it's extremely normal for there to be about 100 misleading hints that you think you see and one that works. Unfortunately, there's no way to know which ones are misleading and which ones aren't without some trial and error.

In cryptography, instead of giving hints, the person making the code will actively try NOT to give hints, or actively try to mislead in one way or another. I was impressed that some students were able to get this problem **WITHOUT** the hint. While my scoring system doesn't reward them proportionately for their efforts, I believe very strongly in giving students challenges, and thought this was a neat one.

Arup