Breaking Cryptograms at Notacon 5

By Meta and Seoci

April 11, 2008

Making friends

The obvious and best place to start breaking a cipher is to ask around and collect ideas from others. The first goal is to discover which encryption algorithm was used; no significant progress can be made until this is known. However, the most important and rewarding reason to ask your peers about the problem, is to meet and collaborate with interesting individuals. It is in this spirit that such crypto challenges are often presented at conferences, such as Notacon. While breaking a cipher is rewarding in its own right, even more can be learned from the diverse skill sets of one's peers. It certainly adds to the fun!

The encryption challenge:
uegg c uwte mr xb jvr jdepwea cvstiys? ara ucb tioigv mrk vgjgyosfef. oaege ng glohg fs zpqqutf yyaywtc lavlpl ok hogdosu!

Eliminating the basics

Not knowing which algorithm was used to encrypt the text, we started by eliminating Rot13. Brute-force checking every valid key soon eliminated Ceaser ciphers in general. At this stage, we did some basic frequency analysis, guessing that it may be the more generic substitution cipher. We used online scrabble and dictionary websites to obtain lists of the common two and three digit words. Leveraging the frequency analysis, the common word lists, and a lot of guesswork, we tried to find a consistent character map for a substitution cipher. Most of our attempts were foiled by the presence of MR, MRK and YYAYWTC. Because there are very few (less than 100) English words with seven letters that begin with the letter pattern 1121 the possibilities for a substitution cipher were severely limited. MR and MRK further inhibited the possibilities for a substitution cipher because we weren't able to fit any possibilities with MR/MRK and YYAYWTC. Ultimately, starting with the shorter words in the cipher text was instructive, but we failed to find a map which worked. Using the troublesome words we decided that it was most likely not a substitution cipher and hence it wouldn't be worth the time to try brute force methods. It was time to check with our peers again.

Learning about keys and length

To figure out what to do next we did some social engineering on other con-goers. Normally, this would mean pretending to be someone you aren't in order to trick people into giving you information. At a security conference you probably won't get very far if you go this route. On the other hand, if you set out to make friends who will hopefully tell you things, you'll get much better results. The keys to using social engineering in situations like this are being warm, friendly, having beer to share, and above all, befriending other attendees. Using this technique we learned that Nick Farr of HacDC made the challenge, and then we learned from Nick that a Vigenére cipher had been used.

Now we had a place to start! We immediately began writing some basic tools in Python to help decrypt the cipher text given a key. It seemed pretty obvious to try decrypting the cipher text using "hacdc" as the key. We wrote the tools in Python and used them directly from the shell.

```
>>> import Vigenére as v
>>> v.decrypt(v.ct, "hacdc")
'need a nwrb kk xz gtk jbbnpey ztltgvq try rau tglgzv koi oghdwhsdbd hacdc gg eimag dp xiqorry
ywxwpta iyolni md hmdbhss'
```

We immediately know that we are on the right track because of the first two words: "need a". From here we tried to continue guessing the key but accomplished very little. We decided that we should attempt to determine the key length. If we could discover the key length, and if it was not too long, we may be able to brute force the rest of the key. Eventually we came up with an interesting way to solve this problem. One method at a time, our Vigenére decrypting framework was beginning to grow! One important thing we noticed was that encrypting or decrypting with a string of a's has no effect. This may seem obvious, but it provided a quick way to reposition the part of the key we knew. Basically the idea is to slide the key down the cipher text until you start seeing partially correct decrypted words later in the cipher text. The index of the first such occurrence may be the key length.

```
>>> v.slide(v.ct1,"hacdc",20)
0   need a nwrb kk xz gtk jbbnpey ztltgvq
1   uxge z swme ko vb cvp gbeiwcx avltgvq
2   uezg a rute fr vy hvr cdcmuea vvqqgys
3   uegz c stre mr qb hsp jdeiwcx avstbyq
4   uegg v uuqc mr xb cvp gbepwet ctpriys
5   uegg c nwrb kr xb jvk jbbnwea cvltgvq
6   uegg c uptc jp xb jvr jwentca cvstirs
7   uegg c uwme ko vb jvr jdeiwcx avstiys
8   uegg c uwtx mp uz jvr jdepwxa asqtiys
9   uegg c uwte fr vy hvr jdepwea vvqqgys
10  uegg c uwte mk xz gtr jdepwea cvltgvq
11  uegg c uwte mr qb hsp jdepwea cvstbyq
12  uegg c uwte mr xu jto hdepwea cvstiyl
13  uegg c uwte mr xb cvp gbepwea cvstiys
14  uegg c uwte mr xb jor hacpwea cvstiys
15  uegg c uwte mr xb jvk jbbnwea cvstiys
16  uegg c uwte mr xb jvr cdcmuea cvstiys
17  uegg c uwte mr xb jvr jwentca cvstiys
18  uegg c uwte mr xb jvr jdxpuby cvstiys
19  uegg c uwte mr xb jvr jdeiwcx avstiys
```

One word at a time

Line 14 provides the first interesting result. Although not entirely obvious, given the context we guessed that "jor hacpwea" may decrypt to "for hacking" or "for hackers". If we could confirm this, we would know that the key was fourteen characters long and potentially a few more characters of it. At this point we needed a method to extract the correct key for a cipher text string given a guess at the plain text. In this case, I needed to know what key would produce "for" from "jvr".

```
>>> v.getkey("jvr", "for")
'eha'
```

From this, we were able to determine that the key probably ended in 'e'. The next step was to repeat this process and to see if decrypting "jdepwea" to "hacking" or "hackers" revealed any further information about the key.

```
>>> v.getkey("jdepwea", "hacking")
'cdcforu'

>>> v.getkey("jdepwea", "hackers")
'cdcfsni'
```

This yeiled a very strong correlation with our current guess for the key: "hacdc_____e". It is always important to consider context; since this code was designed as a fun challenge, we suspected that the key contained a message as well instead of just random characters. Hence "foru" seemed to be a better fit than "fsni" for digits of the key. We checked both in the context of the whole string by deciphering the cipher text with each key using 'a' for characters we did not yet know.

```
>>> v.decrypt(v.ct, "hacdcforuaaaae")
'need a pick mr xb for hacking cvsters yoy pok zioigr fri seeshusfef kteeb lb suuhg fs viqorra khgywtc
htvjmj jw qugdosq'

>>> v.decrypt(v.ct, "hacdcfsniaaaae")
'need a pegw mr xb for hackers cvsters yoy pko lioigr fri seeolgsfef kteeb lb oyghg fs viqorra glsywtc
htvjmj js uggdosq'
```

One step at a time we discovered more about the key. Also, the usefulness of our simple Python framework for exploring Vigenére cipher text was becoming very evident. We now had a methodology and tools to guess individual words of the cipher text and verify those guesses by matching the reversed key with what we had already discovered.

At this point there were several choices of words to attack next; the two digit words, or "cvsters". Given the number of words that end in "ers" it seemed best to attack the two digit words. We again used our list of common two digit words to help guess. After a bit of guesswork we were still not able to figure out a good match for "mr xb". We tried a brute force approach by generating all possible pairs of the two letter words in our list, but the outcome was a bit too verbose to be very useful. Maybe it was the energy drinks or maybe it was persistence, but eventually we figured out that "me up" worked well. We had been working on this puzzle for quite a while and we definitely agreed, it was time for a pick me up!

```
>>> v.getkey("mrxb", "meup")
'andm'
```

Almost there

We have our key! "hacdcforuandme". It's fourteen characters long, it decrypts the first part of the cipher text perfectly, and the text of the key is itself a message. However, it didn't decrypt all of the cipher text!

```
>>> v.decrypt(v.ct, "hacdcforuandme")
'need a pick me up for hacking ciphers yoy pok zibfur fri seeshussbt kteeb lb suuht cg viqorra khgyjqq htvjmj jw qugqlgq'
```

We were very close but not there yet. It was certainly possible that a different key was used for each sentence, but at least it was clear that whatever was being done, it was being to each sentence. So, we separated each sentence and tried the next obvious thing: decrypt the second sentence with our existing key.

```
>>> v.decrypt(v.ct1, "hacdcforuandme")
'need a pick me up for hacking ciphers'

>>> v.decrypt(v.ct2, "hacdcforuandme")
'try raw fruits and vegetables'

>>> v.decrypt(v.ct3, "hacdcforuandme")
'hacdc is proud to sponsor healthy eating at notacon'
```

Woot! We cracked the code! Thanks HacDC and Notacon for the cool challenge and the veggies. :)

Source code for vigenere.py

```
# cipher text

# This challenge was provided by Hacdc as a challenge at Notacon 5, Cleveland, 2008

ct = "uegg c uwte mr xb jvr jdepwea cvstiys ara ucb tioigv mrk vgjgyosfef oaege ng glohg fs zpqqutf yyaywtc lavlpl ok hogdosu"


# the ciphertext broken into sentences

ct1 = "uegg c uwte mr xb jvr jdepwea cvstiys"

ct2 = "ara ucb tioigv mrk vgjgyosfef"

ct3 = "oaege ng glohg fs zpqqutf yyaywtc lavlpl ok hogdosu"


# a list of common two digit words

two = ['am', 'an', 'as', 'at', 'ax', 'be', 'by', 'do', 'go', 'he', 'if', 'in', 'is', 'it', 'me', 'my', 'no', 'of', 'on', 'or', 'ox', 'so', 'to', 'up', 'us', 'we']


# and the answer is...

keyguess = "hacdcforuandme"
```

```
def decrypt(ct, keyString):
    """decrypt the vigenere cipher text using keyString"""
    pt = ""
    spaces = 0
    # remove spaces
    ctx = "".join(ct.split())
    for i in range(0, len(ctx)):
        pt += decryptchar(ctx[i], keyString[i%len(keyString)])
        if i+spaces+1 < len(ct):
            if ct[i+spaces+1] == ' ':
                pt += ' '
                spaces += 1
    return pt

def decryptchar(char, keyChar):
    """decrypt a particular character given a key character.
    helper function for shiftword"""
    al = 'abcdefghijklmnopqrstuvwxyz'
    if char in al:
        n = (al.index(char) - al.index(keyChar)) % len(al)
        return al[n]
    else:
        return " "

def keychar(cipherChar, plainChar):
    """Returns the key character needed to encrypt a plain text char into the cipher char"""
    al = 'abcdefghijklmnopqrstuvwxyz'
    k = al.index(cipherChar) - al.index(plainChar) % len(al)
    return al[k]
```

```
def getkey(cipherWord, plainWord):
    """Return the key string that would encrypt the plain text into the cipher word"""
    keyWord = ""
    for i in range(0, len(cipherWord)):
        keyWord += keychar(cipherWord[i], plainWord[i])
    return keyWord


def slide(ct, key, num=20):
    """use key to decrypt the ciphertext and print the result when that key starts at positions 0 to
    100. this is creates a list which may be useful to visually determine the keylength.
    The key needs to be at lest partialy correct.j"""
    for i in range(0,num):
        k = "a"*i
        k += key
        print i, " ", decrypt(ct, k)
```