Cryptography Coursework

Matthew Consterdine

February 2018

Three ciphertexts were presented, and three plaintexts were found.

Cipher one is the Vingère cipher using the key "BYO". Cipher two is an XOR cipher using **[19, 32]** as the masks. Cipher three is a product cipher comprised of an XOR cipher using **[12, 29]** as the masks, and the Atbash cipher. To help with this task and for fun, a number of python scripts have been developed to both decipher and encipher text.

1 Cipher One: Vingère Cipher

1.1 Solution

Key: **[1, 24, 14]** "BYO"

> The brain is a wondrous machine, but requires sustained effort to maintain it. You must train your brain to consider all possibilities before making decisions. Let your mind explore new places and probabilities. Learn from others; wisdom shared is wisdom expanded. When faced with something different and difficult, consider it a challenge.

- "Dynamic Positive Thinking", by William Dollar. Published 2014 by Lulu Press.

1.2 Cryptanalysis

While the letters of the ciphertext appear random, the punctuation and casing is not. This indicates that a very simple cipher was used, a cipher that probably cannot handle anything other than a single cased Latin alphabet. Looking through the course material, it could be a Caesar, Vingère, Rotor, or another substitution cipher. It is unlikely to be a transposition cipher as even when enciphered, it looks like an English sentence. It is not a bitwise or modern block/stream cipher.

Now each cipher can be attempted. It is easy to bruteforce every possible Caesar shift, all of which are obviously wrong. This was done more for fun than to solve the problem, as it was highly unlikely that the word "gg" in the ciphertext could be decoded using any such cipher. In scrabble there are only two such words[1]: "aa", and "mm"; both of which are rare onomatopoeia.

0. Ufs cpojl wt y kplrsmit kodfwoc, pvr ffoijpst qitrojlse ctgmfu rc nyworojl wu. Wcv ...
1. Ter bonik vs x jokqrlhs jncevnb, ouq eenhiors phsqnikrd bsflet qb mxvnqnik vt. Vbu ...
2. Sdq anmhj ur w injpqkgr imbduma, ntp ddmghnqr ogrpmhjqc arekds pa lwumpmhj us. Uat ...
3. Rcp zmlgi tq v hmiopjfq hlactlz, mso cclfgmpq nfqolgipb zqdjcr oz kvtlolgi tr. Tzs ...
4. Qbo ylkfh sp u glhnoiep gkzbsky, lrn bbkeflop mepnkfhoa ypcibq ny jusknkfh sq. Syr ...
5. Pan xkjeg ro t fkgmnhdo fjyarjx, kqm aajdekno ldomjegnz xobhap mx itrjmjeg rp. Rxq ...
6. Ozm wjidf qn s ejflmgcn eixzqiw, jpl zzicdjmn kcnlidfmy wnagzo lw hsqilidf qo. Qwp ...
7. Nyl vihce pm r dieklfbm dhwyphv, iok yyhbcilm jbmkhcelx vmzfyn kv grphkhce pn. Pvo ...
8. Mxk uhgbd ol q chdjkeal cgvxogu, hnj xxgabhkl ialjgbdkw ulyexm ju fqogjgbd om. Oun ...
9. Lwj tgfac nk p bgcijdzk bfuwnft, gmi wwfzagjk hzkifacjv tkxdwl it epnfifac nl. Ntm ...
10. Kvi sfezb mj o afbhicyj aetvmes, flh vveyzfij gyjhezbiu sjwcvk hs domehezb mk. Msl ...
11. Juh redya li n zeaghbxi zdsuldr, ekg uudxyehi fxigdyaht rivbuj gr cnldgdya lj. Lrk ...
12. Itg qdcxz kh m ydzfgawh ycrtkcq, djf ttcwxdgh ewhfcxzgs qhuati fq bmkcfcxz ki. Kqj ...
13. Hsf pcbwy jg l xcyefzvg xbqsjbp, cie ssbvwcfg dvgebwyfr pgtzsh ep aljbebwy jh. Jpi ...
14. Gre obavx if k wbxdeyuf wapriao, bhd rrauvbef cufdavxeq ofsyrg do zkiadavx ig. Ioh ...
15. Fqd nazuw he j vawcdxte vzoqhzn, agc qqztuade bteczuwdp nerxqf cn yjhzczuw hf. Hng ...
16. Epc mzytv gd i uzvbcwsd uynpgym, zfb ppystzcd asdbytvco mdqwpe bm xigybytv ge. Gmf ...
17. Dob lyxsu fc h tyuabvrc txmofxl, yea ooxrsybc zrcaxsubn lcpvod al whfxaxsu fd. Fle ...
18. Cna kxwrt eb g sxtzauqb swlnewk, xdz nnwqrxab yqbzwrtam kbounc zk vgewzwrt ec. Ekd ...
19. Bmz jwvqs da f rwsyztpa rvkmdvj, wcy mmvpqwza xpayvqszl jantmb yj ufdvyvqs db. Djc ...
20. Aly ivupr cz e qvrxysoz qujlcui, vbx lluopvyz wozxupryk izmsla xi tecuxupr ca. Cib ...
21. Zkx hutoq by d puqwxrny ptikbth, uaw kktnouxy vnywtoqxj hylrkz wh sdbtwtoq bz. Bha ...
22. Yjw gtsnp ax c otpvwqmx oshjasg, tzv jjsmntwx umxvsnpwi gxkqjy vg rcasvsnp ay. Agz ...
23. Xiv fsrmo zw b nsouvplw nrgizrf, syu iirlmsvw tlwurmovh fwjpix uf qbzrurmo zx. Zfy ...
24. Whu erqln yv a mrntuokv mqfhyqe, rxt hhqklruv skvtqlnug eviohw te payqtqln yw. Yex ...
25. Vgt dqpkm xu z lqmstnju lpegxpd, qws ggpjkqtu rjuspkmtf duhngv sd ozxpspkm xv. Xdw ...

Next the Vingère cipher was tested. To do so, a Python script was written which would identify the key length using the Kasiki test. The following repeats were found:

{y: [11, 272], rc: [54, 87], wu: [64, 271], dmbtgrfp: [95, 269], blr: [162, 252]}

Now, the distances between repeats can be calculated: **[261, 33, 207, 174, 90]**. Of them, the greatest common divisor is three. Three is small with a mere 25^3 combinations. Therefore bruteforce was used. To reduce the number of possible answers, the word "the" was searched for as it is the most common word in the English language.

The plaintext was found with the key **[1, 24, 14]**, "BYO".

[1] List of two letter scrabble words: https://en.wikibooks.org/wiki/Scrabble/Two_Letter_Words

2 Cipher Two: XOR Cipher

2.1 Solution

Key: **[19, 32]**

A 24 year old boy seeing out from the train's window shouted...

"Dad, look the trees are going behind!"
Dad smiled and a young couple sitting nearby, looked at the 24 year old's childish behavior with pity, suddenly he again exclaimed...

"Dad, look the clouds are running with us!"

The couple couldn't resist and said to the old man...

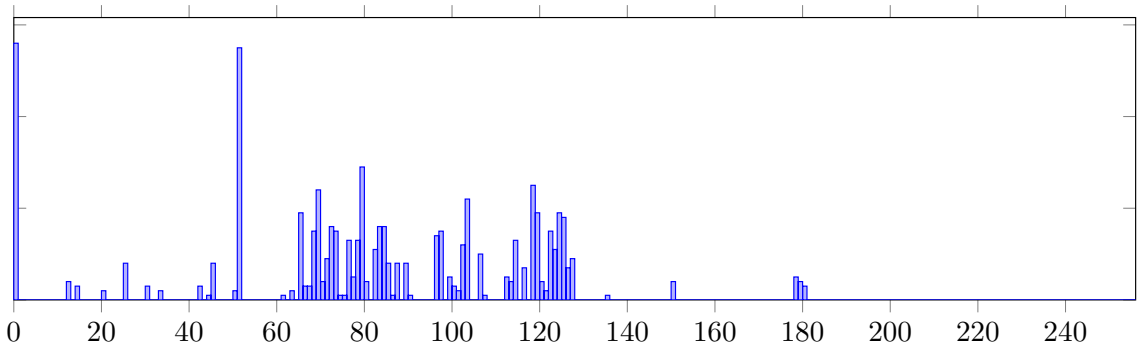
"Why don't you take your son to a good doctor?" The old man smiled and said..."I did and we are just coming from the hospital, my son was blind from birth, he just got his eyes today."

Every single person on the planet has a story. Don't judge people before you truly know them. The truth might surprise you.

- "Everyone Has a Story in Life" in "Life of War", by Ranbir Singh. Published 2017 by Notion Press.

2.2 Cryptanalysis

When deciphering an unknown ciphertext, it pays to start simple. As the ciphertext contains many unusual or unprintable characters it is not a classic cipher, and instead one which operates on the byte level. It could be a modern block/stream cipher however plotting the frequency distribution reveals that the ciphertext has low entropy:



Clearly the cipher is simple and insecure. It could be a shift cipher operating at the byte level, or it could be an XOR cipher. Simply shifting the bytes had little effect, but XOR proved fruitful. As it is known that the first character is "A", the mask, M, can be calculated and applied to the string:

$$M = ord(\text{"}A\text{"}) \oplus ord(\text{"}R\text{"}) \qquad M = 0x41 \oplus 0x52 \qquad M = 0x13$$

With the results:

```
00000000: 4113 3207 204a 6552 7213 6f5f 6413 625c  A.2. JeRr.o_d.b\
00000010: 7913 7356 655a 6e54 205c 7547 2055 725c  y.sVeZnT \uG Ur\
00000020: 6d13 745b 6513 7441 615a 6ec2 a173 1377  m.t[e.tAaZn..s.w
00000030: 5a6e 576f 4420 4068 5c75 4765 57e2 80a6  ZnWoD @h\uGeW...
```

There are many invalid ASCII bytes, but only every other byte. This suggests that it is an XOR cipher with a key length of two. Looking at the ciphertext the NULL byte seems incredibly common, maybe it is a SPACE?

$$M = ord(SPACE) \oplus ord(NULL) \qquad M = 0x20 \oplus 0x00 \qquad M = 0x20$$

Applying returns the plaintext:

```
00000000: 4120 3234 2079 6561 7220 6f6c 6420 626f  A 24 year old bo
00000010: 7920 7365 6569 6e67 206f 7574 2066 726f  y seeing out fro
00000020: 6d20 7468 6520 7472 6169 6ee2 8099 7320  m the train...s
00000030: 7769 6e64 6f77 2073 686f 7574 6564 e280  window shouted..
```

The last piece of the puzzle is that the text isn't using ASCII or UTF-8, it is using CP1252.

3 Cipher Three: Product Cipher (XOR + Atbash)

3.1 Solution

Key: **[12, 29]**

therearemorevolcanoesonvenusthananyotherplanetwithinoursolarsystem

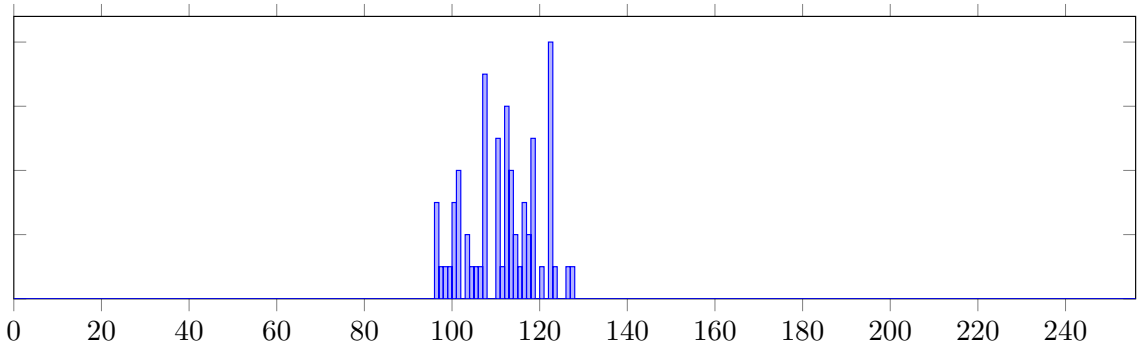
- Unknown

3.2 Cryptanalysis

3.2.1 Part 1

After solving the first two ciphers, it is worth thinking about which ciphers haven't been used. Namely: a substitution ciphers, transposition ciphers, and modern ciphers. It is likely that at least one of them will be used as part of this cipher.

Looking at the ciphertext, it appears to simply be printable ASCII characters, however closer examination reveals ASCII DELETE (0x7f) towards the end of the plaintext. Therefore, at least one of the two ciphers operates on the level of bytes. Rerunning the frequency distribution shows the plaintext values clustered between 0x96 and 0x127. This is larger than the Latin alphabet, but smaller than any real encoding.



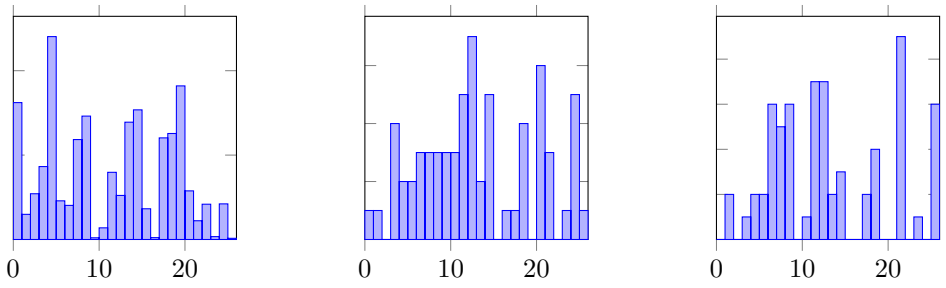
If the second of the ciphers is a substitution or transposition cipher, the first cipher needs to transform the ciphertext into a printable form as it is known that the plaintext is alphabetic. However, even after splitting the ciphertext into multiple bins it is impossible to do so by merely shifting the bytes. So, code was written to try every possible XOR mask, searching for one that just was printable. For this task, a regular expression was used:

/^[a−z]{$length}$/ where **$length** is equal to the length of the ciphertext.

The following two texts were found:

- **[15, 29]** dsuiuzjvmljvfllxymovklneumehdsymymaldsuihoymuggrdsqmofjhooyikbkgun
- **[12, 29]** gsvivzivnliveloxzmlvhlmevmfhgszmzmblgsvikozmvgdrgsrmlfihlozihbhgvn

3.2.2 Part 2



(a) English (b) dsuiuzj... (c) gsvivzi...

Plotting the frequency distribution of the two texts reveals that the second middletext appears to match the frequency distribution of the English language, just reversed. So, the characters in both texts were inverted, with the results as follows:

- **[15, 29]** whfrfaqenoqeuoocbnlepomvfnvswhbnbnzowhfrslbnfttiwhjnluqsllbrpyptfm
- **[12, 29]** therearemorevolcanoesonvenusthananyotherplanetwithinoursolarsystem

The first is obviously wrong, and the second is likely right.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

This is the Atbash cipher where Z is mapped to A, Y is mapped to B and so on.

4 Appendix

4.1 Cipher One Code

caesar.py

```
ciphertext = "Ufs cpojl wt y kplrsmit kodfwoc, pvr ffoijpst qitrojlse ctgmfu rc nyworojl
    ↪ wu. Wcv kitr hsywo..."

for shift in range(26):
  print(shift, "".join(list(map(lambda c: chr(((ord(c) - ord("a") - shift + 26) % 26) +
      ↪ ord("a")) if c>="a" and c<="z" else (chr(((ord(c) - ord("A") - shift + 26) % 26)
      ↪  + ord("A")) if c>="A" and c<="Z" else c), ciphertext))))
```

part1.py

```
from fractions import gcd
from functools import reduce
from collections import defaultdict
import math, re

# Part One (Viginere)

# Find the key length
def kasiki(ciphertext):
  # Adding a space ensures the last word is always added.
  ciphertext = re.sub(r"[^a-z ]", "", ciphertext.lower() + " ")
  positions = defaultdict(list)
  position = 0
  word = ""

  # Identify words
  for character in enumerate(ciphertext):
    if character == " ":
      if word != "":
        positions[word].append(position)
        word = ""
    else:
      word += character
      position += 1

  # Find deltas
  keylengths = []
  for entry in dict(filter(lambda entry: len(entry[1]) > 1, positions.items())).items():
    for index in range(len(entry[1]) - 1):
      keylengths.append(entry[1][index+1] - entry[1][index])

  # Reduce to find the common gcd
  return reduce(gcd, keylengths)

# Reformat lowercase alphabetic plaintext using ciphertext formatting.
def reformat(plain, cipher):
  return "".join(list(map(lambda c: plain.pop(0).lower() if c>="a" and c<="z" else (plain
      ↪ .pop(0).upper() if c>="A" and c<="Z" else c), cipher)))

def decipher(ciphertext, keylength, prefix):
  # Brute force, as the keylength is only 3
  lowertext = re.sub(r"[^a-z]", "", ciphertext.lower())
  for it in range(26**keylength):
    keys = [int((it / (26**n)) % 26) for n in range(keylength)]
    deciphered = [chr((ord(c) - ord("a") - keys[i%keylength]) % 26 + ord("a")) for i,c in
        ↪  enumerate(lowertext)]
    if "".join(deciphered).startswith(prefix):
      print(keys, reformat(deciphered, ciphertext))
```

```
# Find the keylength
ciphertext = "Ufs cpojl wt y kplrsmit kodfwoc, pvr ffoijpst qitrojlse ctgmfu rc nyworojl
    ↪ wu. Wcv kitr hsywo wcvp psywo rc dmbtgrfp omj dpqgjzwmghjcg cctpps nyyjlu
    ↪ ecqjqwplg. Mch zmis kwob synzpps ock qjodcg blr qpccypjjwugst. Jsbpb gpcn mhicft;
    ↪ uwtbcn qvbpse gg xggema fvdblrfb. Kicb gyqfb kjrv tmafrvjlu egtgcfflh blr
    ↪ egtggqvjh, dmbtgrfp wu y qiyzmcbhc."
keylength = kasiki(ciphertext)
print(keylength)
decipher(ciphertext, keylength, "the") # "the" is the prefix
```

4.2 Cipher Two Code

frequency.py

```
with open("secret.hex","rb") as f: ciphertext = f.read()
# ciphertext = bytearray("...".encode("cp1252"))
count = Counter(ciphertext)
output = ""
for i in range(0, 256):
  output += (str(i-ord("a")) + "\n") * count[i]
print(output)
```

part2.py

```
import codecs, sys

# Part Two (XOR)

if len(sys.argv) == 2:
  # Decipher using keys from argv and data from stdin
  keys = list(map(int, sys.argv[1].split(",")))
  data = sys.stdin.buffer.read()
  print(bytearray([c^keys[i%len(keys)] for i,c in enumerate(data)]).decode("cp1252"))
elif len(sys.argv) == 3:
  # Decipher using keys and filename from argv
  keys = list(map(int, sys.argv[1].split(",")))
  with open(sys.argv[2],"rb") as f: data = f.read()
  print(bytearray([c^keys[i%len(keys)] for i,c in enumerate(data)]).decode("cp1252"))
else:
  print("Usage:", sys.argv[0], "KEY1,KEY2,... | KEY1,KEY2,... FILE")
  # Default to show it working
  with open("secret.hex","rb") as f: data = f.read()
  print(bytearray([c^[0x13,0x20][i%2] for i,c in enumerate(data)]).decode("cp1252"))
  exit(1)
```

4.3 Cipher Three Code

english.py

```
data = {"E": 12.02, "T": 9.10, "A": 8.12, "O": 7.68, "I": 7.31, "N": 6.95, "S": 6.28, "R"
    ↪ : 6.02, "H": 5.92, "D": 4.32, "L": 3.98, "U": 2.88, "C": 2.71, "M": 2.61, "F":
    ↪ 2.30, "Y": 2.11, "W": 2.09, "G": 2.03, "P": 1.82, "B": 1.49, "V": 1.11, "K": 0.69,
    ↪  "X": 0.17, "Q": 0.11, "J": 0.10, "Z": 0.07}

for point in data:
  for it in range(int(100*data[point])):
    print(ord(point) - ord("A"))
```

part3.py

```
import math, re, sys

# Part Three (XOR and Atbash)

def decipher(ciphertext):
  # Go forever basically.
  for keycount in range(1, 1000000):
    found = False
    print(str(keycount) + ":")

    for it in range(256**keycount):
      # Generate keys, then XOR each byte.
      keys = [math.floor(it / (256 ** i) % 256) for i in range(keycount)]
      middletext = [chr((ord(c)^keys[i%len(keys)])%256) for i,c in enumerate(ciphertext)]
      # Cut down results, to decode more complex strings alter the regex.
      if re.match(r"[a-z]{"+str(len(ciphertext))+"}", "".join(middletext)):
        found = True
        plaintext = [chr(25 - ord(c) + 2*ord("a")) for c in middletext]
        print(keys, "".join(plaintext))

    if not found: print("Nothing found.")
    print()

def encipher(keys, plaintext):
  # Substitute then XOR
  middletext = [chr(25 - ord(c) + 2*ord("a")) for c in plaintext]
  plaintext = [chr((ord(c)^keys[i%len(keys)])%256) for i,c in enumerate(middletext)]
  print("".join(plaintext))

if len(sys.argv) == 2:
  # Brute force decipher
  decipher(sys.argv[1])
elif len(sys.argv) == 3:
  # Encipher text
  keys = list(map(int, sys.argv[1].split(",")))
  encipher(keys, sys.argv[2])
else:
  print("Usage:", sys.argv[0], "- | CIPHERTEXT | KEY1,KEY2,... PLAINTEXT")
  # Default to show it working
  decipher("knztzgekbqekiqcevp`kdqaxzpjuknvpvpnqknztgrvpzzhokn~p`{eu`rvtd-dzzs")
  exit(1)
```