CISS451

[Christian Elliott] (February 23, 2021)

Contents

Chapter 1

Basic number theory

Chapter 2

Classical ciphers

2.1 Shift cipher

Definition 2.1.1. Shift Cipher

The **shift cipher** is a cipher where a message is shifted by a constant value.

To unencrypt the message, the shift is applied to the encrypted message in reverse.

Exercise 2.1.1. Suppose we want to encrypt the string 'popcorn'.

Let s be the set of our alphabet, the lowercase English letters. Therefore $s = \{a, b, c, \ldots, z\}$

The shift cipher is not complex. It works by simply shifting the values of your alphabet by a constant integer.

Let us use the shift of 7.

So, we simply shift each letter of our plaintext message to the right by 7.

Therefore, $\{p, o, p, c, o, r, n\}$ becomes $\{w, v, w, j, v, y, u\}$

What were to happen if we had used a shift of 7 on the word 'today'?

This is where the shift cipher makes use of modular arithmetic. $t \rightarrow a$, because when you go past z, you simply wrap around back to the beginning of the alphabet, a.

Now we know that our ciphertext 'wvwjvyu' is the encrypted version of 'popcorn'.

To decrypt, one must simply apply the shift in the opposite direction to the ciphertext.

Therefore, $\{w, v, w, j, v, y, u\}$ becomes $\{p, o, p, c, o, r, n\}$.

2.2 Encryption and Decryption

The shift cipher is only a simple example of a type of cipher. It helps to discuss ciphers in a more general way.

We can describe a cipher as a pair of functions $E : P \to C$ and $D : C \to P$ where E is called the encryption function and D is the decryption function.

This allows us to consider P, the plaintext (unencrypted message) as follows:

$$D(E(x)) = x \text{ for all } x \in P$$

In plain words, this means you can decrypt the encrypted message to attain the original plaintext message.

In the case of the Caesar Cipher, can express our encryption function as

$$E(x) = (SHIFT + ASCII\_VALUE(x))\%26$$

2.3 Attacking the Shift Cipher

Recall that the shift cipher is based off a simple shift value. In our example, the length of the alphabet was 26. We represented our values using 0 ... 25. Therefore, there were only 26 options for our shift value.

This means that we can bruteforce the shift values.

$0 \leq k \leq 25$. We can print all the decryptions of ciphertext c, by simply decrypting by every possible shift value.

Heuristic

If the message is long enough, we can actually break the shift cipher without needing to brute-force it. The letters in the English language appear in fairly predictable frequencies. The letter `e` is the most commonly appearing letter.

The cipher doesn't mutate at all. That is to say, the same k is used for every input. Therefore, if `e` $\to$ `i`, it will do so for the whole plaintext. This means that if we determine a best guess for which letter `e` $\to$?, we can guess at what the shift is. It is simply the value of the ciphertext minus the value of `e` (mod 26).

Here is a table of the average probability of English letters a - z:

Letter	Probability
e	0.127
t	0.091
a	0.082
o	0.075
i	0.070
n	0.067
s	0.063
h	0.061
r	0.060
d	0.043
l	0.040
c	0.028
u	0.028
m	0.024
w	0.023
f	0.022
g	0.020
y	0.020
p	0.019
b	0.015
v	0.010
k	0.008
j	0.002
x	0.001
q	0.001
z	0.001

These are merely the letters frequencies found in a large body of text. They are sure to change in some degree depending on the sample text.

We can extend this “commonly occurring logic can be extended to pairs of letters, known as **digrams**. Why stop at pairs of two letters? Of course, we can continue hunting for commonly occuring groups of letters.

2.4 Affine cipher

The Affine cipher encryption algorithm is as follows:

$$E((a,b),x) = (ax+b) \pmod{26}$$

The importance of this algorithm is the key value. Notice that it uses two values, a and b, in the form $ax+b$. This means that the encryption value range is much larger, and there are many more keys for an attacker to try to compute the plaintext value.

So, we know the formula for E of the Affine cipher. What about D?

We can write D in terms of E.

$$\begin{aligned} D((a,b),(E(a,b),x)) &= x \pmod{26} \\ D((a,b),ax+b) &= x \pmod{26} \end{aligned}$$

We know that every encryption must be decryptable for it to be used as a cipher. Let's assume that D has the form

$$D((a,b),x) = cx+d \pmod{26}$$

If we apply some algebra to E and D, we can arrive at

$$D((a,b),ax+b) \equiv x \pmod{26}$$

And then

$$\begin{aligned} c(ax+b)+d &\equiv x \pmod{26} \\ cax+cb+d &\equiv x \pmod{26} \end{aligned}$$

For now, let's assume

$$cax \equiv x \pmod{26}$$

and

$$cb+d \equiv 0 \pmod{26}$$

For the above to be true, c and d must meet some specific criteria. Notice that $cax \equiv x \pmod{26}$. This means c and a must somehow cancel out (in mod 26).

How do numbers cancel each other out in the modular world? Through multiplicitive inverses!

We will be able to discuss what to do with $cb + d \equiv 0 \pmod{26}$ once we look at $cax \equiv x \pmod{26}$ first.

We know that c must be the multiplicitive inverse of $a \pmod{26}$. The multiplicitive inverse of a, in this case denoted c, satisfies

$$ca \equiv 1 \pmod{26}$$

In other words, what number multiplied by a equals 1 in mod 26?

The following python program produces all integers with multiplicitive inverses in mod 26.

```
for i in range(26):
  for j in range(26):
    if i * j % 26 == 1:
      print('%d is a mult. inverse of %d (mod 26)' % (j, i))
```

Outputs:

```
[student@localhost tmp]$ python cipher-tools.py
1 is a mult. inverse of 1 (mod 26)
9 is a mult. inverse of 3 (mod 26)
21 is a mult. inverse of 5 (mod 26)
15 is a mult. inverse of 7 (mod 26)
3 is a mult. inverse of 9 (mod 26)
19 is a mult. inverse of 11 (mod 26)
7 is a mult. inverse of 15 (mod 26)
23 is a mult. inverse of 17 (mod 26)
11 is a mult. inverse of 19 (mod 26)
5 is a mult. inverse of 21 (mod 26)
17 is a mult. inverse of 23 (mod 26)
25 is a mult. inverse of 25 (mod 26)
```

As you can see, not all integers in mod 26 have a multiplicitive inverse. This means that the value chosen for a in our cipher must posess this quality.

Alright, so we know the value of c. This multiplicitive inverse is denoted a^{-1}. Since we can now label c, we can do more work on the second part of the equation:

$$\begin{aligned} cb + d &\equiv 0 \pmod{26} \\ a^{-1}b + d &\equiv 0 \pmod{26} \\ d &\equiv -a^{-1}b \pmod{26} \end{aligned}$$

We then can put it all together:

$$\begin{aligned} D((a,b),x) &\equiv a^{-1}x - a^{-1}b \pmod{26} \\ &\equiv a^{-1}(x-b) \pmod{26} \end{aligned}$$

where a must be invertible mod 26.

Exercise 2.4.1. Encrypt the plaintext message `gollum` using the key (3, 12) with the Affine Cipher.

Let $x =$ `gollum`

- $\mathrm{g} = 6 \to 3(6) + 12 = 4 \pmod{26}$
- $\mathrm{o} = 14 \to 3(14) + 12 = 2 \pmod{26}$
- $\mathrm{l} = 11 \to 3(11) + 12 = 19 \pmod{26}$
- $\mathrm{l} = 11 \to 3(11) + 12 = 19 \pmod{26}$
- $\mathrm{u} = 20 \to 3(20) + 12 = 20 \pmod{26}$
- $\mathrm{m} = 12 \to 3(12) + 12 = 22 \pmod{26}$

Now we have $E((3,12),x) =$ `[4, 2, 19, 19, 20, 22]`
Converting the ciphertext integer values back to lowercase ascii we get `ecttuv`.

2.5 Vigenere cipher

The Vigenere cipher differs from the previous two ciphers in that it is polyalphabetic.

Let's dig in.

You select a keyword. For example "hello". [7, 4, 11, 11, 14]
Suppose our plaintext is `ihavethehighgroundnow`

Notice that the key is shorter than the plaintext message. No worries. We simply must repeat the key as needed to satisfy the length of the plaintext. (with some unused overflow)

Length(plaintext) = 21. Length of key = 5. We can repeat the key 5 times to match the length of the plaintext.

The encryption works as follows: for every letter of the plaintext, we shift the alphabet by the value of the corresponding letter in the key. So, plaintext[0] = i = 8. key[0] = h = 7. Therefore, ciphertext[0] = (8 + 7) % 26 = 15. Therefore ciphertext[0] = 15 = p.

This process repeats for every value in the plaintext, shifting by the value of the key[i] % len(key).

Exercise 2.5.1. Let's do a full example using the key and message we already discussed above.

Let message = `ihavethehighgroundnow` and key = `hello`

Let's go ahead and convert both the key and the message to their numerical value.

```
to_ints(message) = [8, 7, 0, 21, 4, 19, 7, 4, 7, 8, 6,
  7, 6, 17, 14, 20, 13, 3, 13, 14, 22]
```

and `to_ints(key) = [7, 4, 11, 11, 14]`
Let's iterate through message and convert each value.

```
ciphertext = []
for i in range(len(message)):
  cipher_char = (message[i] + key[i % len(key)]) % 26
  ciphertext.append(cipher_char)
```

```
    return to_chrs(ciphertext)
```

This will give us

```
[15, 11, 11, 6, 18, 0, 11, 15, 18, 22, 13,
  11, 17, 2, 2, 1, 17, 14, 24, 2, 3]
= pllgsalpswnlrccbroycd
```

That is how we encrypt using the Vigenere cipher. To decrypt, we must have the key. We can then simply take the ciphertext and apply the key to it in the same way, only this time subtracting the value of the key from the encrypted character. We will arrive at the decrypted char.

Todo: Attacking the Vigenere cipher using frequency analysis to determine the key length.

2.6 Substitution cipher

2.7 Permutation cipher

2.8 Hill cipher

2.9 One-time pad cipher

2.10 Linear feedback shift register

Bibliography

Index