File Name: Activity 5 Report

Student Name: Wuli Zuo

Student ID: a1785343

Link to the repository of Activity 5:

In Activity 5, Java codes are developed to solve the first two problems and the provided Python script is modified to solve the third problem. For Ex1, alphabet frequency analysis, observation and guessing are main approaches to get the plaintext, while the code is used as assistance to generate possible keys, to sort, to substitute and to output. For Ex2 and Ex3, because the scale of operation is quite small when the hint is given, Bruce Force is enough to decrypt the ciphers. All three plaintexts are covered successfully. Although this report contains explanations of the implementation, it is most recommended to read the source code to get the details: https://github.com/DoriaSummer/sse_activity5.

Ex1: Mono-alphabetic substitution

- Problem analysis
  - Since the key is one of the permutations of 26 English alphabets, the computation scale will be the factorial of 26, which is quite large.
  - The format of the cipher looks like an email or a letter, with greetings, salutations and punctuation, which means some guessing could be tried.
  - The ciphertext contains 59 words. This is enough to make a simple frequency analysis, which may not give the correct key but could show many useful hints.

- Approaches selection
  - Alphabet frequency analysis
  - Observation and guessing
  - Java programming

- Decryption process
  - Step1:

    Use the method `getFrequencySort()` to count the frequency of each alphabet appears in the ciphertext and correspond it to the known letter frequency in English language to get the initial trial key, and then use it to decrypt the ciphertext with the method `getPlainMono()`. The Output 1 shows the result of the initial trial.

Code pieces 1: Decrypt.java (Ex1)

```
private static void ex1() {
    System.out.println("\nEx1 for Activity 5\n");
    String cipher = readFile("ex1.enc");
    System.out.println("Ciphertext:\n" + cipher);
    char[] frq = getFrequencySort(cipher);
    sortFrequency(frq);

    // try for the frequency
    String plainText = getPlainMono(cipher, frq);
    System.out.println("\nTry with frequency:");
    System.out.println("Alphabet: " + Arrays.toString(letterFrq));
    System.out.println("Try key:  " + Arrays.toString(frq));
    System.out.println("\n" + plainText);
......
private static String getPlainMono(String c, char[] k) {
    String res = "";

    for (int i = 0; i < c.length(); i++) {
        char a = c.charAt(i);
        String as = Character.toString(a);
        if (!as.matches("[a-z]")){
            res += as;
        } else {
            for (int j = 0; j < k.length; j++) {
                if(a == k[j]){
                    res += (char)(97+j);
                }
            }
        }
    }
    return res;
}
......
private static char[] getFrequencySort(String text){
    char[] res = new char[26];
    char[] count = new char[26];
    for (int i = 0; i < res.length; i++) {
        res[i] = (char)(97+i);
        count[i] = 0;
    }
    for (int i = 0; i < text.length(); i++) {
        char a = text.charAt(i);
        String as = Character.toString(a);
        if (as.matches("[a-z]")){
            count[a-97]++;
        }
    }
    for (int i = 0; i < res.length-1; i++) {
        for (int j = i+1; j < res.length; j++) {
            if(count[j] > count[i]) {
                swap(count, i, j);
                swap(res, i, j);
            }
        }
    }
    return res;
}
```

Output 1: Try to decrypt with the frequency (Ex1)

```
Ex1 for Activity 5


Ciphertext:

fc hoh,


fow ct grgxqdfcsz zocsz?


nxg qoa ixgg dfct ixcjnq sczfd?

wg nxg pbnsscsz do fnrg n pnxdq nd kfnxbcg't pbnkg.

dfg cjgn ct do hxcsz wfnd qoa wnsd do gnd nsj znlgt dfnd qoa wnsd do pbnq.

nsj fnrg n bod oi ias dozgdfgx!

iggb ixgg do csrcdg loxg pgopbg.


fopg do tgg qoa dfgxg!


kfggxt,

nbckg




Try with frequency:

Alphabet: [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]

Try key:  [n, h, t, q, g, a, j, b, f, y, v, z, i, c, o, k, e, x, s, d, p, l, r, m, w, u]


in bob,


ioy nc ewerdtinsl lonsl?


are dof mree tinc mrngad snlit?

ye are uhassnsl to iawe a uartd at piarhne'c uhape.
```

```
tie ngea nc to brnsl yiat dof yast to eat asg lavec tiat dof yast to uhad.

asg iawe a hot om mfs toletier!

meeh mree to nswnte vore ueouhe.


ioue to cee dof tiere!


pieerc,

ahnpe
```

- Step 2:

  Read the output of the first trial, give it a guess for the most obvious words (e.g. hi, cheers, are you, 's) and adjust the key to make the next trial. The plaintext comes out only after three times of repetition of the above process.

  The words identified and the hints got in each time are included as the comments in Code pieces 2, and the output of each trial in the process is presented in Output 2.

Code pieces 2: Decrypt.java (Ex1)

```java
private static void ex1() {
......
    // modify the key from observation
    // guess 1: fc=hi, do=to, n=a, ngx=are, qoa=you, tgg=see, 't='s, kfggxt=cheers
    // a-n, b-h, c-k, e-g, h-f, i-c, o-o, r-x, s-t, t-d, u-a, y-q
    char[] key1 = {'n','h', 'k', 'w', 'g', 'p', 'j', 'f', 'c', 'y', 'v', 'z', 'i',
            'b', 'o', 's', 'e', 'x', 't', 'd', 'a', 'l', 'r', 'm', 'q', 'u'};
    plainText = getPlainMono(cipher, key1);
    System.out.println("\nTry with Guess 1:");
    System.out.println("Alphabet: " + Arrays.toString(LetterFrq));
    System.out.println("Try key:  " + Arrays.toString(key1));
    System.out.println("\n" + plainText);

    // guess 2: w-w, grgxqdfcsz=everything, zocsz=going, wg=we, cjgn=idea, hxcsz=bring,
fopg=hope, nbckg=alice
    // v-r, n-s, g-z, p-p, w-w, d=j, l-b
    char[] key2 = {'n','h', 'k', 'j', 'g', 'l', 'z', 'f', 'c', 'y', 'v', 'b', 'i',
            's', 'o', 'p', 'e', 'x', 't', 'd', 'a', 'r', 'w', 'm', 'q', 'u'};
    plainText = getPlainMono(cipher, key2);
    System.out.println("\nTry with Guess 2:");
    System.out.println("Alphabet: " + Arrays.toString(letterFrq));
    System.out.println("Try key:  " + Arrays.toString(key2));
    System.out.println("\n" + plainText);

    // guess 3: ixgg=free, ixcjnq=friday, znlgt=games, oi=of, ias=fun, iggb=feel,
loxg=more
    // f-i, m-l, l-b
    char[] key3 = {'n','h', 'k', 'j', 'g', 'i', 'z', 'f', 'c', 'y', 'v', 'b', 'l',
            's', 'o', 'p', 'e', 'x', 't', 'd', 'a', 'r', 'w', 'm', 'q', 'u'};
    plainText = getPlainMono(cipher, key3);
    System.out.println("\nTry with Guess 3:");
    System.out.println("Alphabet: " + Arrays.toString(letterFrq));
    System.out.println("Try key:  " + Arrays.toString(key3));
    System.out.println("\n" + plainText);
    System.out.println("Found key: " + Arrays.toString(key3));
}
```

Output 2: Try to decrypt with adjusted keys (Ex1)

```
Try with Guess 1:
Alphabet: [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]
Try key:  [n, h, k, w, g, p, j, f, c, y, v, z, i, b, o, s, e, x, t, d, a, l, r, m, q, u]

hi bob,

hod is ewerythipl loipl?

are you mree this mrigay pilht?
```

```
we are fnappipl to hawe a farty at charnie's fnace.

the igea is to bripl dhat you dapt to eat apg laves that you dapt to fnay.

apg hawe a not om mup tolether!

meen mree to ipwite vore feofne.


hofe to see you there!


cheers,

anice




Try with Guess 2:

Alphabet: [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]

Try key:  [n, h, k, j, g, l, z, f, c, y, v, b, i, s, o, p, e, x, t, d, a, r, w, m, q, u]


hi bob,


how is everything going?


are you mree this mriday night?

we are planning to have a party at charlie's place.

the idea is to bring what you want to eat and gafes that you want to play.

and have a lot om mun together!

meel mree to invite fore people.


hope to see you there!


cheers,

alice
```

- Decryption result

```
Output 3: The decrypted plaintext and the key for Ex1
```

```
Try with Guess 3:

Alphabet: [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]

Try key:  [n, h, k, j, g, i, z, f, c, y, v, b, l, s, o, p, e, x, t, d, a, r, w, m, q, u]


hi bob,


how is everything going?


are you free this friday night?

we are planning to have a party at charlie's place.

the idea is to bring what you want to eat and games that you want to play.

and have a lot of fun together!

feel free to invite more people.


hope to see you there!


cheers,

alice


Found key: [n, h, k, j, g, i, z, f, c, y, v, b, l, s, o, p, e, x, t, d, a, r, w, m, q, u]
```

Ex2: Poly-alphabetic shift

- Problem analysis
  - Since the key consists of 4 English alphabets, the computation scale will be quite small, as the permutation of any 4 of the 26 alphabets, that is 26*25*24*23 = 358,800. There is no problem to execute such level of operation with Java.
  - The ciphertext contains the name of the day of the week. This will greatly help with the identification of the correct plaintext. There is even no need to compare the result with a meaningful words set.
  - The ciphertext contains only 6 words, which further limits the scale of computation.

- Approaches selection
  - Bruce force with Java programming
- Decryption process
  - Step1:

    Generate all the possible combinations of any 4 of the 26 English alphabets with the method `dfs()`, and then generate the permutation for each combination by using the method `fullSort()`.

Code pieces 3: Decrypt.java (Ex2)

```java
private static int num = 4; //number of letters
private static char[] arr = {0, 0, 0, 0};
private static HashSet<String> K = new HashSet<>();
......
private static void ex2() {
    System.out.println("\nEx2 for Activity 5\n");
    String cipher = readFile("ex2.enc");
    System.out.println("Ciphertext: " + cipher);
    dfs(0, 0);
}
......
private static void dfs(int deep, int index) {
    if (deep == num) { //get num, print
        String[] keys = fullSort(arr, 0, arr.length);
        K.addAll(Arrays.asList(keys));
        return;
    }
    for (int i = index; i < 26 - num + 1 + deep; i++) { //letters can be get
        arr[deep] = (char) (97 + i);
        dfs(deep + 1, i + 1); //starting index of the next layer
    }
}
private static String[] fullSort(char[] ch, int start, int end) {
    HashSet<String> full = new HashSet<>();
    if (start == end) {
        full.add(String.valueOf(ch));
    } else {
        for (int i = start; i < end; i++) {
            swap(ch, i, start);
            String[] res1 = fullSort(ch, start + 1, end);
            full.addAll(Arrays.asList(res1));
            swap(ch, start, i);
        }
    }
    String[] res = new String[full.size()];
    full.toArray(res);
    return res;
}
```

- Step 2:

  For each permutation string, which is a possible key, try to use it to decrypt the ciphertext, until the plaintext contains any one of the names of the day of the week.

Code pieces 4: Decrypt.java (Ex2)

```
private static HashSet<String> K = new HashSet<>();
private static String[] week = {"monday", "tuesday", "wednesday", "thursday", "friday",
"saturday", "sunday"};

private static void ex2() {
......
    for (String k : K) {
        String plainText = getPlainPoly(cipher, k);
        for (String day : week) {

            if (plainText.contains(day)) {

                System.out.println("Plaintext: " + plainText);

                System.out.println("key: " + k);

            }

        }

    }

    K.clear();

}
```

- Decryption result

Output 4: The decrypted plaintext and the key for Ex2

```
Ex2 for Activity 5


Ciphertext: evfsgm aj vgfjicf oekf mrvud qqzvea


Plaintext: attack or retreat wait until monday


key: wyoi
```

Ex3: Textbook RSA:

- Problem analysis

    - RSA is hard to crack because to find the prime factors of a large digital is known as a hard problem. It is almost impossible to guess the key.
    - Fortunately, the hint tells that the ciphertext contains only three alphabets. The easiest way to solve this problem is just to encrypt all the possible plaintext and then compare each option to the given encrypted file.
    - The scale of computation is 26*26*26 = 17,576, which is easy to a computer.

- Approaches selection
  - Modify the provided RSA textbook script
  - Bruce force with Python programming
- Decryption process
  - Step 1:

    Modify the Python script by rewriting the encrypt method into a version that is able to take a string as the parameter of plaintext, instead of reading from a file.

Code pieces 5: textbook_rsa.py (Ex3)

```
# rewrite encrypt to encrypt plaintext from string
def rsa_encrypt_text(plaintext, fname):
   with open(pkfile, 'rb') as f:
      sk = RSA.importKey(f.read(), 'PEM')

   pt = plaintext
   if len(pt) > 128:
      pt = pt[:128]
   pt = pt.encode('ascii')

   ct = sk.encrypt(pt, 0)[0]

   with open(fname + '.enc', 'wb') as f:
      f.write(ct)
```

  - Step 2:

    Generate all the possible plaintext strings, use the method `diff()` to compare each one with the provided encrypted file.

Code pieces 6: textbook_rsa.py (Ex3)

```
# compare two encrypted files
def diff(f1, f2):
    with open(f1, 'rb') as f:
        s1 = f.read()
    with open(f2, 'rb') as f:
        s2 = f.read()
    if s1 == s2:
        return True
    else:
        return False

# main
print('Ex3 for Activity 5\n')
if __name__ == '__main__':
    alphabet = string.ascii_letters[:26]
    for i in alphabet:
        for j in alphabet:
            for k in alphabet:
                rsa_encrypt_text(i+j+k, 'test')
                if diff('ex3.enc', 'test.enc'):
                    print(i+j+k)
                    break

      sk = RSA.importKey(f.read(), 'PEM')

   pt = plaintext
   if len(pt) > 128:
      pt = pt[:128]
   pt = pt.encode('ascii')

   ct = sk.encrypt(pt, 0)[0]
   with open(fname + '.enc', 'wb') as f:

      f.write(ct)
```

- Decryption result

Output 5: The decrypted plaintext and the key for Ex3

```
Ex3 for Activity 5

sun
```