CSE 5473 — Network Security Autumn 2020

Assignment #1

Symmetric Cryptography *Due September 23rd, 11:59PM*

1 Secret Key Cryptography - DES [30 pts]

1. Search the web for a DES implementation (e.g., `http://des.online-domain-tools.com/`) and use it to find the result of encrypting the plaintext `A23B716BA14C32C9` with the key `D43CB19AF490D7CE` in hexadecimal format

2. The keys `0000000000000000` and `FFFFFFFFFFFFFFFF` (in hexadecimal format) are considered "weak" keys for DES (`http://en.wikipedia.org/wiki/Weak_key`). Explain why these keys are weak. (Hint: Consider the per-round keys that they generate and the effect these per-round keys have on each round of the algorithm – You might want to check your textbook or do a google search to understand what a "weak" key means.)

3. If DES keys had been 128 bits as IBM originally proposed, how long would one million computers, each trying one trillion keys per second, take to examine all possible keys?

Answer:

1. The result of encrypting the plaintext `A23B716BA14C32C9` with the key `D43CB19AF490D7CE` is: `37F32E892018D7ED`.

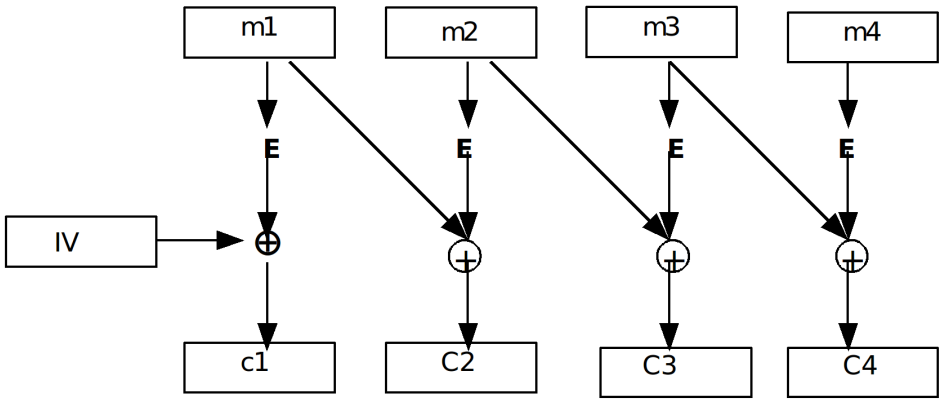
2. The keys `0000000000000000` and `FFFFFFFFFFFFFFFF` are considered "weak keys" because of the per-round keys that they generate. To back up a bit, "weak keys" are keys that cause DES encryption to act exactly like DES decryption. By encrypting plaintext twice with a weak key, the result is the original plaintext. We also know that DES uses a 56-bit key that is broken up into 16 keys. Now, the keys above are "weak keys" because they produce 16 identical subkeys. Since all the keys are identical, encrypting plaintext twice will result in the original plaintext.

3. To try all the 128 bit keys: $\frac{2^{128}}{10^6 \times 10^{12} \times 3600 \times 24 \times 365} = 1.08 \times 10^{13}$ years.

2 Encrypting Large Messages [40 pts]

1. Suppose a sequence of plaintext blocks, x_1; x_2; ...; x_n yields the ciphertext sequence y_1; y_2; ...; y_n. Suppose that one ciphertext block, say y_i, is transmitted incorrectly (i.e., some 1's are changed to 0's and vice versa). What is the number of plaintext blocks that will be decrypted incorrectly when the following modes are used: (1) ECB, (2) CBC, (3) OFB, and (4) CFB modes are used.

2. What pseudo-random block stream is generated by 64-bit OFB with a weak DES key?



3. The pseudo-random stream of blocks generated by 64-bit OFB must eventually repeat (since at most 2^{64} different blocks can be generated). Will $k\{IV\}$ necessarily be the first block to be repeated? Explain.

4. A common message integrity check is DES-CBC, which means encrypting the message in CBC mode and using the final ciphertext block as the checksum. This final block is known as the "CBC residue". This is what is used in the banking industry. Show how you can construct a message with a particular CBC residue, with the only constraint that somewhere in the message you have to be able to embed 64 bits of "garbage".

5. Consider the following alternative method to CBC for encrypting large messages. In this CBC variant, the encryption is done as shown in the Figure below. Is this variant reversible (can you decrypt the ciphertext generated by this scheme)? How? What are the security implications of this scheme, if any, as opposed to the "normal" CBC? (Hint: Think about passive and active attacks that are possible to carry on this scheme.)

Answer:

1. ECB: 1. This scenario will only affect one block (i.e. the block where the ciphertext was transmitted incorrectly). Only one block is affected because each block is encrypted and decrypted independently.

   CBC: 2. Two blocks will be affected in this scenario (i.e. the current block and the next one). In CBC decryption, a corrupted ciphertext block C_i' will be decrypted and then XOR'ed with the previous ciphertext block C_{i-1} (for the sake of this explanation, let's assume the previous ciphertext block is not corrupted). Because a corrupted ciphertext block (C_i') was XOR'ed with correct ciphertext block (C_{i-1}), the decrypted plaintext P_i' will be incorrect. Now, the corrupted ciphertext C_i' is then XOR'ed with the next decrypted ciphertext C_{i+1} in the sequence. This will result in yet another corrupted plaintext P_{i+1}'. However, assuming this "next decrypted ciphertext" C_{i+1} is okay, no other blocks are affected by the corruption.

   OFB: 1. Only one block is affected because there are no chaining dependencies. Each ciphertext is encrypted and decrypted independently of the other blocks. As a result, errors will

only be contained in the current block.

CFB: Here, the error propagation is the same as CBC (i.e. two blocks will be affected, the current one and the next one). There is a chaining dependency with the ciphertext during decryption, so an error in block i will affect block i+1.

2. With a weak DES key k, $E(x,k) = D(x,k)$, then $E(E(x,k),k) = D(E(x,k),k) = x$. Also, OFB has a block size of 64, so the current pad is encrypted (in whole) to generate the next pad. Therefore, the one-time pads are: $k\{IV\}, IV, k\{IV\}, IV, ...$

3. Yes, Name the pad sequence as $p_1 = k\{IV\}, p_2, ... , p_i, ... , p_j$, assume p_i is the first repeated pad (by p_j) and $1 < i < j$. Then, $p_{i-1} = D(p_i)$ and $p_{k-1} = D(p_k)$. Therefore, $p_{i-1} = p_{k-1}$. This contradicts with the assumption that p_i is the first repeated pad. This process can be repeated until there does not exist p_{i-1}. Only $i = 1$ does not has a precedent. So, p_1 must be the first repeated key.

4. In this question, we are dealing with message integrity check. Therefore, the message itself is in plaintext, with CBC residue appended to protect the message integrity. If you assume CBC is used for both privacy and integrity, you *need* to also assume that two different keys are used, one for privacy (encryption) and one for integrity. Moreover, the question says "the only constraint...". As such, you can make many assumptions (yes, include knowing the secret key).

   In this reference answer, I assume we already know the key and the plaintext message (IV is not used to compute CBC residue). The goal is to insert a garbage block somewhere in the message so the new message will have the specified CBC residue. We can construct the message in the following steps:

   a. write down the message in the plaintext blocks and leave one block blank (to be used as the garbage block). Assume the garbage block is the ith block, so we need to compute (m_i).
   b. using the key and previous message blocks, we can get c_{i-1} by computing forward from the first block: $c_{i-1} = E((c_{i-2} \oplus m_{i-1}), k)$
   c. using the key, the residue and message blocks after m_i, we can also get c_i by computing backward from the CBC residue: $c_i = m_{i+1} \oplus D(c_{i+1}, k)$
   d. Assign the garbage block: $m_i = c_{i-1} \oplus D(c_i, k)$

5. a. Yes, this scheme is reversible. To decrypt the message: $m_1 = D(c_1 \oplus IV, k)$, $m_2 = D(c_2 \oplus m_1, k)$, $m_3 = D(c_3 \oplus m_2, k)$...

   b. There are several security implications:
      - It is not self-synchronizing. Modifying, deleting or rearranging one block will garble the current block and all the blocks after the current one. Moreover, the IV only affects the first message block.
      - Since the plaintext block is computed by applying the decryption function last, the attacker can not predictably change the plaintext by modifying the cphertext without

knowing the key. In CBC, this is doable because $m_i = c_{i-1} \oplus D(c_i, k)$. Changing c_{i-1} will garble m_{i-1}, but predictably flip bits in m_i.

- Attackers knowing plaintext and ciphertext can rearrange the ciphertext with few limitations. This is because he can compute $E(m_i, k)$ for any message block without the key: $E(m_i, k) = m_{i-1} \oplus c_i$. With $E(m_i, k)$, he can rearrange ciphertext without garbling the message. For example, given $m_1|m_2|m_3$, the attacker can rearrange it into $m_3|m_2|m_1$ by sending the following ciphertext: $IV \oplus E(m_3, k)|m_3 \oplus E(m_2, k)|m_2 \oplus E(m_1, k)$. If IV is unknown to him, he will be forced to use m_1 as the first block (the only limitation).

3 Multiple Encryption DES [30 pts]

In answering this question, assume that an average close to 2^{32} steps to break an encryption system (find the corresponding key(s)) is tractable, while an average close to 2^{64} steps or larger is not.

1. Let's assume that you do DES double encryption by encrypting with k_1 and doing DES in decrypt mode with k_2. Does the same meet-in-the-middle attack described in the textbook work as with double encryption with k_1 and k_2? If not, how could it be made to work? (Hint: it is possible but requires some tweaks. Please describe your tweaks)

2. Devise a meet-in-the-middle attack to break EDE. That is, find the keys k_1 and k_2 assuming knowledge of a set of plaintext/ciphertext pairs <m_i, c_i>obtained by applying EDE on each p_i. (Hint: you meet either after ED, or before DE)

3. Assume that EDE was deployed using three keys k_1 (for step 1 encryption), k_2 (for step 2 decryption), and k_3 (for step 3 encryption). Is this new version of EDE more secure than the traditional EDE? Explain.

4. Consider the following approach to increase the security of DES. Encryption is done by encrypting the plaintext twice using the same shared secret. Decryption is done by decrypting the ciphertext twice using the same shared secret. Is this approach more secure than the traditional DES?

Answer:

1. Yes. The key to answer this question is to prove that using a second and third <m, c>pairs will significantly reduce the probability that $E(m_i, k_1) = E(c_i, k_2)$. The same reasoning applies to this variant as well except when $k_1 = k_2$. Though, we can safely assume $k_1 \neq k_2$ because it is obviously an unsafe use of ED: no work is required to reveal the plaintext, the ciphertext is always the same as the plaintext.

2. With triple DES (EDE), we can either meet between ED or DE. Assume the attack meets at the latter point. Then the meet-in-the-middle attack to break EDE is as

   a. Make a table with 2^{112} entries (because two keys are used in this table, each has 2^{56} choices) where each entry consists of two DES keys K_1 and K_2 and the result $r = D(E(m_1, K_1), K_2)$. Sort the table by K_1, then by K_2 (Notice we sort the table by the keys. Sorting it by r would require a little more work to attack EDE than necessary.)

b. Make a second table with 2^{56} entries where each entry consists of a DES key K and the result $r = D(c_1, K)$. Sort the table by K.

c. To perform the attack, go through the entries in table 1 one by one, use K_1 to index into table 2 (to find $D(c_1, K_1)$), check whether two entries have the same result, i.e., $D(E(m_1, K_1), K_2) = D(c_1, K_1)$. If so, test the key pair K_1, K_2 with other <plaintext, ciphertext> pairs.

d. Extra practise: analyse the number of imposters and the probability to eliminate imposters by using extra pairs of $<m_i, c_i>$.

Notice, this attack is actually the same as the brutal force attack.

3. EDE with three keys is as secure as EDE with two keys. EDE with two keys can be attacked using brutal force with 2^{112} tries. Using meet-in-the-middle attack, EDE with three keys can be attacked with $2^{112} + 2^{56}$ tries (The beauty of meet-in-the-middle attack is turning the number of tries from multiplication to plus.)

   a. Build table 1 as in 2.a, but sort the table according to r.

   b. Build table 2 as in 2.b, but sort the table according to r.

   c. Like the meet-in-the-middle attack, go through these two tables to find entries with the same result ($2^{112} + 2^{56}$ steps). If so, verify these three keys with extra $<m_i, c_i>$pairs.

4. No, this scheme is not more secure than regular DES. Because the same key is used for both encryption rounds, it is not even necessary to perform a man-in-the-middle attack like in regular 2DES. It can be argued that this scheme is slightly more secure because a brute-force attack is required to decrypt twice, thus at most doubling the amount of time required to find the key, but the attacker still only needs to try all 256 keys, or, on average, 255 keys. It can also be argued that this scheme is slightly weaker than normal DES, because it will encrypt plaintext to plaintext if given a weak key, whereas DES will at least give a properly encrypted value with a weak key.

4 Breaking Monoaphabetic Cipher Using Frequence Analysis [30 points. There will be partial credit if you only partially solve it.]

A monoalphabetic cipher is one where each symbol in the input (called the "plaintext") is mapped to a fixed symbol in the output (called the "ciphertext"). Let's go back to the age where we use statistic analysis (or frequence analysis) to break ciphers (esentially the keys). It will be a fun assignment, and you can use modern techniques to break old ciphers.

While you can entirely finish this assignment by using pencil and paper, it is more efficient to use (existing) programs. Please **Find**[1] or **write** a program[2] that reads an input ciphertext, and outputs the (sorted) letter frequencies, diagram (consecutive pairs of letters) frequencies, and trigram (consecutive triplets of letters) frequencies.

[1]For instance, `http://crypto.interactive-maths.com/frequency-analysis-breaking-the-code.html`, which I recommend especially if you don't wanna write code.

[2]Please note that you don't have to write your program in our VM. You can use whatever computing environment convenient to you, e.g., Windows, Linux, Mac, etc., using Java, C, Python, etc.

Using this frequence analysis program, decode the following ciphertext. The original plaintext that produced this ciphertext contains only upper case letters; all punctuation and spaces were removed before encoding. It was encrypted with a mono-alphabetic cipher. Explain how you decoded the ciphertext in your answer.

```
WMTLXJAVQNXBTECJIZMWDMEIGXXVTSXRAWHXHTNHVMTVW
KVMXBTECQGXUCXSWXNDXWYNQZQGNQZTIQGPAHRWNRQGVM
XGAVASXJAVJIVMXCTKVTNHPAHRWNRJIVMXCTKVWZWKMVQ
YNQZZMTVVMXSXMTKJXXNWNVMXDQNHADVQGVMXJSWVWKME
WNWKVSIGQSVMXBTKVVZQIXTSKVQPAKVWGIVMQKXMQCXKZ
WVMZMWDMRXNVBXEXNMTLXJXXNCBXTKXHVQKQBTDXVMXEK
XBLXKTNHVMXMQAKXWKWVVMTVWNKWHWQAKKEWBXZWVMZMW
DMQASCXVWVWQNMTKJXXNBTVXBISXDXWLXH
```

Reference `http://en.wikipedia.org/wiki/Frequency_analysis`

Answer1 (using tools): We can begin by performing the frequency analysis of all characters, digrams, and trigrams on the ciphertext. Immediately we see the most frequent trigram is 'VMX', suggesting that 'V' = 'T', 'M' = 'H', and 'X = E'. We make those substitutions.

Next, we notice that there is an occurrence of 'VMTV' in the text. It is possible that 'VMTV' is 'THAT', so we substitute 'A' for 'T'.

Next, the most frequent letter in the ciphertext is 'W'. We have already used the first three most frequent letters 'E', 'T', and 'A', so it is possible 'W' is 'O', 'I', 'N', or 'S'. Substitution with 'O' leads to some instances of 'TO', but it also creates two instances of 'OTO', which is a very rare trigram in English, so seeing it appear twice is unlikely. Additionally, there seem to be many instances where 'O' appears by itself. The same is true for 'S' and 'N' as well, and we also get an instance of a quadgram 'TSTS' or 'TNTN', which should not appear in English. The reasonable candidate for 'W' seems to be 'I' due to the number of times 'W' appears by itself in addition to 'WV' being one of the most frequent digrams in the ciphertext, which would decrypt to 'IT'.

The next most frequent letter is 'K'. It is possibly 'O', 'N', or 'S', assuming our previous substitutions are correct. The digram 'WK' appears 7 times in the ciphertext and the digram 'KV' appears 5 times. If 'K' is 'O', then that means we have 'IO' and 'OT', which don't appear frequently in English. If 'K' is 'S', then we have 'IS' and 'ST', which both appear frequently. If K is 'N', then we have 'IN' and 'NT', which also appear frequently. 'N', seems promising, however we also see that 'KK' appears in the ciphertext. The probability of 'NN' appearing in English is extremely low, so it seems like 'S' should be substituted instead.

The next most frequent letter is 'Q' in the ciphertext. We should try matching 'O', 'N', or 'R', the next most frequent letter in English. The most frequent digram we have that contains 'Q' with a letter we know is 'VQ'. This gives us 'TO', 'TN', and 'TR' respectively. The only frequent digram among those is 'TO', so 'O' isa likely candidate. We also see that choosing 'O' completes a sequence 'VMQKX' to read 'THOSE', which improves our confidence not only in choosing O, but also in our previous choice of 'S' for 'K'.

We have already matched 'T' with 'A', so the next most frequent ciphertext letter is 'N', which is possibly 'N' or 'R'. We see the digrams 'WN' and 'XN' appear frequently in the ciphertext. This gives us 'IR', 'ER', 'IN', 'EN' for 'R' and 'N' respectively. Although 'ER' is frequent, 'IR' is not, while both 'EN' and 'IN' are frequent. Additionally, a frequent trigram is 'NQZ'. Because 'Q' is 'O', this might be 'FOR', but if 'N' is 'R', then it would be in the wrong place. Because of this, it is likely that 'N' decrypts to itself.

We have a decent number of letters and can try to complete some words in the text now.

First, we notice the sequence 'XVWVWQN' is 'ETITION'. It is likely that the full word is 'PETITION', possibly as part of 'REPETITION' or 'COMPETITION'. The ciphertext letter before the sequence is 'C', so we substitute 'P' for 'C'. This also completes the ciphertext 'CTKV' to 'PAST' and 'MQCX' to 'HOPE', which increases our confidence.

Next, we see the sequence 'JXXN', which is '?EEN', appear frequently, and sometimes with only one undecrypted character before it. The whole word could be 'SEEN', 'BEEN', 'PREEN', 'KEEN', etc. It cannot be 'SEEN', as we have already solved 'S'. 'PREEN' is too large to fit in some places without changing our previous substitutions. It seems 'BEEN' is most likely, since a phrase like 'PETITION HAS KEEN' would not make much sense. We substitute 'B' for 'J'.

Next, there is the phrase 'VMXCTKVWZWKMVQ' which decrypts to 'THE PAST I ?ISH TO'. It is highly unlikely this is part of a word containing 'ISHTO', meaning that '?ISH' is a 4 letter word ending in 'ISH'. There are very few of these, the most common being 'FISH', 'DISH', and 'WISH'. 'WISH' seems the most likely candidate here, so we substitute 'W' for 'Z'. This also completes 'NQZ' to 'NOW'.

We next notice the phrase 'VMXJSWVWKM' which decrypts to 'THEB?ITISH' which is almost certainly 'THE BRITISH', so we substitute 'R' for 'S'.

Near the beginning we see the phrase 'ZMWDM' which is 'WHI?H'. The full word is probably 'WHICH', so substitute 'C' for 'D'.

This reveals the phrase 'XUCXSWXNDX' which is 'E?PERIENCE'. The word is almost certainly 'EXPERIENCE', so substitute 'X' for 'U'.

After 'THE BRITISH', we have the phrase 'EWNWKVSI' which is '?INISTR?'. This is almost certainly 'MINISTRY', so substitute 'M' for 'E' and 'Y' for 'I'.

Next notice the phrase 'VMXEKXBLXK' which is 'THEMSE??ES'. This must be 'THEMSELVES', so replace 'B' with 'L' and 'B' with 'V'.

Near the top is 'JAV', which is 'B?T', which is surely 'BUT'. Thus, we replace 'A' with 'U'.

At this point, enough of the message is decrypted to substitute just by reading it. 'F' for 'G', 'D' for 'H', 'G' for 'R', 'K' for 'Y', 'J' for 'P'. 'O' and 'F' don't appear in the ciphertext, so it is unknown what they decrypt to, but it must be either 'Z' or 'Q'.

This completes the decryption, and we see the plaintext is a quote from Patrick Henry's well-known 'Give me Liberty or Give me Death' speech, although there seems to be an error, because the original quote has ten years instead of two years.

```
IHAVEBUTONELAMPBYWHICHMYFEETAREGUIDEDANDTHATI
STHELAMPOFEXPERIENCEIKNOWOFNOWAYOFJUDGINGOFTH
EFUTUREBUTBYTHEPASTANDJUDGINGBYTHEPASTIWISHTO
KNOWWHATTHEREHASBEENINTHECONDUCTOFTHEBRITISHM
INISTRYFORTHELASTTWOYEARSTOJUSTIFYTHOSEHOPESW
ITHWHICHGENTLEMENHAVEBEENPLEASEDTOSOLACETHEMS
```

```
ELVESANDTHEHOUSEISITTHATINSIDIOUSSMILEWITHWHI
CHOURPETITIONHASBEENLATELYRECEIVED
```

The original quote is:

```
I have but one lamp by which my feet are guided; and that
is the lamp of experience.  I know of no way of judging
of the future but by the past.  And judging by the past,
I wish to know what there has been in the conduct of the
British ministry for the last ten years, to justify those
hopes with which gentlemen have been pleased to solace
themselves, and the House?  Is it that insidious smile
with which our petition has been lately received?
```

Answer2 (writing a program):

```
import java.util.*;
import java.util.Map.Entry;

//A program to output the letter, diagram, and trigram frequencies
public class Mono {
    static void putItem(Map<String, Integer> map, String key) {
        if (map.get(key) == null) {
            map.put(key, new Integer(1));
        } else {
            map.put(key, map.get(key) + 1);
        }
    }

    public static void showFreq(String str) {
        String cipherText = str;
        final int cipherLen = cipherText.length();

        HashMap<String, Integer> letterMap = new HashMap<String, Integer>();
        HashMap<String, Integer> diaMap = new HashMap<String, Integer>();
        HashMap<String, Integer> triMap = new HashMap<String, Integer>();

        String letter;
        String dia;
        String tri;

        for (int i = 0; i < cipherLen; i++) {
            // update letter frequency
            letter = cipherText.substring(i, i + 1);
            putItem(letterMap, letter);

            // update diagram frequency
            if (i > cipherLen -2) {
                continue;
            }
```

```
            dia = cipherText.substring(i, i + 2);
            putItem(diaMap, dia);

            // update trigram frequency
            if (i > cipherLen -3) {
                continue;
            }

            tri = cipherText.substring(i, i + 3);
            putItem(triMap, tri);
        }

        printSorted("Letter Freq:", letterMap);
        printSorted("Digram Freq:", diaMap);
        printSorted("Trigram Freq:", triMap);
    }

    static class ValueComparator implements
            Comparator<Map.Entry<String, Integer>> {

        public ValueComparator() {
        }

        public int compare(Map.Entry<String, Integer> a,
                Map.Entry<String, Integer> b) {
            int result;

            //Return negative of compareTo to reverse the order
            result = -a.getValue().compareTo(b.getValue());

            if (result == 0) {
                result = -a.getKey().compareTo(b.getKey());
            }

            return result;
        }
    }

    static void printSorted(String name, HashMap<String, Integer> map) {
        ValueComparator bvc = new ValueComparator();
        TreeSet<Map.Entry<String, Integer>> sortedSet =
                new TreeSet<Map.Entry<String, Integer>>(bvc);

        sortedSet.addAll(map.entrySet());

        System.out.println(name);

        for (Map.Entry<String, Integer> entry : sortedSet) {
            System.out.print(entry.getKey() + ":" + entry.getValue() + " ");
        }

        System.out.println("\n");
    }

    public static void main(String[] args) {
```

```
String cipherText = "WMTLXJAVQNXBTECJIZMWDMEIGXXVTSXRAWHXHTNHVMTVWKV"
        + "MXBTECQGXUCXSWXNDXWYNQZQGNQZTIQGPAHRWNRQGVMXGAV"
        + "ASXJAVJIVMXCTKVTNHPAHRWNRJIVMXCTKVWZWKMVQYNQZZMT"
        + "VVMXSXMTKJXXNWNVMXDQNHADVQGVMXJSWVWKMEWNWKVSIGQS"
        + "VMXBTKVVXNIXTSKVQPAKVWGIVMQKXMQCXKZWVMZMWDMRXNVB"
        + "XEXNMTLXJXXNCBXTKXHVQKQBTDXVMXEKXBLXKTNHVMXMQAKX"
        + "WKWVVMTVWNKWHWQAKKEWBXZWVMZMWDMQASCXVWVWQNMTKJXX"
        + "NBTVXBISXDXWLXH";

final int cipherLen = cipherText.length();

showFreq(cipherText);
System.out.println("Ciphertext: " + cipherText);

// print the letter frequence as a guide to decode
System.out.println("\nEnglish Freq:");
System.out.println("ETAON RISHD LFCMU GYPWB VKXJQ Z");
System.out.println("TH HE AN RE ER IN ON AT ND ST ES EN OF TE ED OR TI HI AS TO\n");

// Here, replace the letter of ciphertext with inferred plaintext
HashMap<Character, Character> map = new HashMap<Character, Character>();

for (char i = 'A'; i <= 'Z'; i++) {
    map.put(i, i);
}

// This is the key!!
map.put('V', 't');
map.put('M', 'h');
map.put('X', 'e');
map.put('T', 'a');
map.put('W', 'i');
map.put('L', 'v');
map.put('D', 'c');
map.put('K', 's');
map.put('C', 'p');
map.put('J', 'b');
map.put('N', 'n');
map.put('S', 'r');
map.put('Z', 'w');
map.put('U', 'x');
map.put('H', 'd');
map.put('Q', 'o');
map.put('A', 'u');
map.put('G', 'f');
map.put('R', 'g');
map.put('E', 'm');
map.put('I', 'y');
map.put('B', 'l');
map.put('Y', 'k');
map.put('P', 'j');

// output the key
for (Map.Entry<Character, Character> entry : map.entrySet()) {
    if (entry.getKey() != entry.getValue()) {
```

```
                System.out.print(entry.getKey() + "~" + entry.getValue() + " ");
            }
        }

        System.out.println("\n");

        // generate plaintext
        StringBuilder newStr = new StringBuilder();

        for (int i = 0; i < cipherLen; i++) {
            newStr.append(Character.toUpperCase(map.get(cipherText.charAt(i))));
        }

        System.out.println("Plaintext: " + newStr);
    }

}
```

Listing 1: Monoaphabetic cipher

5 Submiting your report

Please write a report describing how you solve each of the problem above. You can use WORD, or Latex to write the report. The Latex source of this assignement is available at `http://web.cse.ohio-state.edu/~lin.3021/file/f18a/hw1.zip`. Regarding how to use Latex, please see this tutorial if you have 0-experience with it `https://www.andy-roberts.net/writing/latex/absolute_beginners`. Please note that latex can be installed in all modern OSes such as Windows, Linux, and Mac.