HW1

Yi Xue 02/20/2016

1a. Answer:
Key: 20
Plain text: I LOVE COMPUTERS

1b. Answer:
I LOVE THIS CLASS

1c. Answer:

(i) Plain text: **HE** $\begin{bmatrix} \mathbf{7} \\ \mathbf{4} \end{bmatrix}$

Cipher text: $\begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{1} & \mathbf{3} \end{bmatrix}$ x $\begin{bmatrix} \mathbf{7} \\ \mathbf{4} \end{bmatrix}$ mod 26 = $\begin{bmatrix} \mathbf{15} \\ \mathbf{19} \end{bmatrix}$ mod 26 = $\begin{bmatrix} \mathbf{15} \\ \mathbf{19} \end{bmatrix}$ **PT**

(ii) Cipher text: **PT** $\begin{bmatrix} \mathbf{15} \\ \mathbf{19} \end{bmatrix}$

decryption: $\begin{bmatrix} \mathbf{3} & \mathbf{24} \\ \mathbf{25} & \mathbf{1} \end{bmatrix}$ x $\begin{bmatrix} \mathbf{15} \\ \mathbf{19} \end{bmatrix}$ mod 26 = $\begin{bmatrix} \mathbf{501} \\ \mathbf{394} \end{bmatrix}$ mod 26 = $\begin{bmatrix} \mathbf{7} \\ \mathbf{4} \end{bmatrix}$ **HE**

2a. Answer : CBC

2b. Answer : CBC

2c. Answer : ECB

2d. Answer : OFB

2e. Answer : ECB

3a. Answer :

(I) A: No
B: It would be expensive to let both parties know/agree on the new/different challenge-response protocol each time they communicate.

(ii) A : No
B: Repeated usage of the same protocol provides the chance to break the protocol by BG.

3b. Answer:

(i) No

(ii) If sender and receiver are not in the same time zone, the time difference could cause that they are not in the same day , thus the receiver will get garbage with wrong decryption date.

4.
(1) Self-critique:
The code written in java, it can do encryption and decryption of simplified DES.

(2) Hard copy of the code : two class includes CryptoSDES for main testing, and SDES for the algorithm.

CryptoSDES.java

```
package cryptosdes;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

/**
 * HOMEWORK 1 PROGRAMMING
 * @author yi
 */
public class CryptoSDES {

    private String inputText;
    private String key;
    private SDES sdes;


    public CryptoSDES(){
        sdes = new SDES();
    }

    //convert input String to Integer list to feed into the SDES class
    private List<Integer> strToList(String text) {
        ArrayList<Integer> intList = new ArrayList();
        for(int index=0; index< text.length();index++) {
            intList.add(Integer.valueOf(text.charAt(index)-48));

        }
        return intList;
    }



    public void start() {
        System.out.println("Simplied DES Demo");
```

```
Scanner userInput = new Scanner(System.in);
String choice = "";
while(!choice.equals("4")){
    System.out.println("\n\n1. Encryption");
    System.out.println("2. Decryption");
    System.out.println("3. Encryption with modified S0");
    System.out.println("4. Quit");

    System.out.println("Choose your operation: ");
    choice = userInput.nextLine().trim();


    switch(choice){
        case "1": {
            //input the required info for encryption
            System.out.println("Please input the plain text: ");
            inputText = userInput.nextLine().trim();
            System.out.println("Please input the key: ");
            key = userInput.nextLine().trim();

            //set up SDES parameters
            sdes.setPlainText(strToList(inputText));
            sdes.setSecretKey(strToList(key));
            sdes.encryption();
            break;
        }
        case "2":{
            System.out.println("Please input the cipher text: ");
            inputText = userInput.nextLine().trim();
            System.out.println("Please input the key: ");
            key = userInput.nextLine().trim();

            //set up SDES parameters
            sdes.setCipherText(strToList(inputText));
            sdes.setSecretKey(strToList(key));
            sdes.decryption();
            break;
        }
        case "3" : {

            System.out.println("Please input the plain text: ");
            inputText = userInput.nextLine().trim();
            System.out.println("Please input the key: ");
            key = userInput.nextLine().trim();

            //set up SDES parameters
            sdes.setPlainText(strToList(inputText));
            sdes.setSecretKey(strToList(key));
```

```
                //change the S0, this part can be input by user, for simplicity, it is set in the code
                List<Integer> modS0 = Arrays.asList(1,0,3,2,
                                                    3,1,3,2,
                                                    0,2,1,3,
                                                    3,2,1,0);
                sdes.modifyS0(modS0);

                sdes.encryption();
                sdes.resetS0();  //reset S0 to the published value
                break;

            }

            case "4" : break;
            default: System.out.println("You have to choose from 1-4");
          }


       }

    }
    public static void main(String[] args) {
        CryptoSDES sdesTest = new CryptoSDES();
        sdesTest.start();




    }

}
```

SDES.java

```
package cryptosdes;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
/**
 * Simplified DES algorithms
 * @author yi
 */
public class SDES {

    private List<Integer> plainText;
    private List<Integer> secretKey;
    private List<Integer> cipherText;
    //define the open parameters of SDES algorithms
    private Map<String, List<Integer>> permutations =new HashMap();
    private List<Integer> IP = Arrays.asList(2,6,3,1,4,8,5,7);
    private List<Integer> IPrev = Arrays.asList(4,1,3,5,7,2,8,6);
    private List<Integer> P10 = Arrays.asList(3,5,2,7,4,10,1,9,8,6);
    private List<Integer> P8 = Arrays.asList(6,3,7,4,8,5,10,9);
    private List<Integer> EP = Arrays.asList(4,1,2,3,2,3,4,1);
    private List<Integer> P4 = Arrays.asList(2,4,3,1);
    private List<Integer> S0 = Arrays.asList(1,0,3,2,
                                    3,2,1,0,
                                    0,2,1,3,
                                    3,1,3,2);
    private List<Integer> S1 = Arrays.asList(0,1,2,3,
                                    2,0,1,3,
                                    3,0,1,0,
                                    2,1,0,3);



    public SDES(){
        initMap();
    }



    private void initMap(){
        permutations.put("P10", P10);
        permutations.put("P8", P8);
        permutations.put("IP", IP);
        permutations.put("IPrev", IPrev);
        permutations.put("P4", P4);
        permutations.put("EP", EP);
        permutations.put("S0",S0);
        permutations.put("S1",S1);

    }

   //general permutation function
    private List<Integer> permutation(String pFlag, List<Integer> inputList){
        ArrayList<Integer> permuteList = new ArrayList();
```

```
    for(Integer index : permutations.get(pFlag))
          permuteList.add(inputList.get(index-1));


    return permuteList;

}

//circular left shift 1 bit
private void Lshift(List<Integer> inputList){

    int size = inputList.size();
    Integer elem0 = inputList.get(0);
    for (int index =1; index< size; index++)
        inputList.set(index-1,  inputList.get(index));

    inputList.set(size-1, elem0);

}



//generate the subkey1
private List<Integer> getSubkey1(){

    List<Integer> permuteKey = permutation("P10",secretKey);
    Lshift(permuteKey.subList(0, 5));
    Lshift(permuteKey.subList(5, 10));
    List<Integer> subKey = permutation("P8", permuteKey);

    return subKey;


}

//generate subkey2
private List<Integer> getSubkey2(){

    List<Integer> permuteKey = permutation("P10",secretKey);
    for(int count=0; count<3; count++) {
        Lshift(permuteKey.subList(0, 5));
        Lshift(permuteKey.subList(5, 10));
    }

    List<Integer> subKey = permutation("P8", permuteKey);
    return subKey;
```

```
}

private void XOR(List<Integer> key, List<Integer> inputList){


   for(int index = 0; index < key.size(); index++) {
       if(key.get(index).equals(inputList.get(index)))
           inputList.set(index, 0);
       else
           inputList.set(index, 1);
    }


}

private List<Integer> mapping(String pFlag, List<Integer> inputList){

    List<Integer> Smap = permutations.get(pFlag);

    int rowNumber = inputList.get(0)*2 + inputList.get(3);
    int colNumber = inputList.get(1)*2 + inputList.get(2);
    int index = rowNumber * 4 + colNumber ;

    ArrayList<Integer> mapResult = new ArrayList();
    switch(Smap.get(index)){
       case 0: {
           mapResult.add(0);
           mapResult.add(0);
           break;
       }
       case 1:{
           mapResult.add(0);
           mapResult.add(1);
           break;

       }
       case 2: {
           mapResult.add(1);
           mapResult.add(0);
           break;
       }
       case 3:{
           mapResult.add(1);
           mapResult.add(1);
           break;
       }
    }

    return mapResult;
```

```
}

//swap the first 4 with the last 4
private void swap(List<Integer> inputList) {

    for (int index =0; index<4; index++) {
        Integer tmp = inputList.get(index);
        inputList.set(index, inputList.get(index+4));
        inputList.set(index+4, tmp);
    }


}

//combine operations of EP, key XOR, mapping, P4, XOR together to make it one cycle
//inputList is changed as the result of the function.
private void cycle(List<Integer> key, List<Integer> inputList) {
    //EP permutation
    List<Integer> EPlist= permutation("EP", inputList.subList(4,8));

    //XOR with key
    XOR(key, EPlist);

    //mapping
    List<Integer> S0map = mapping("S0", EPlist.subList(0, 4));
    List<Integer> S1map = mapping("S1", EPlist.subList(4, 8));
    List<Integer> map = new ArrayList();
    map.addAll(S0map);
    map.addAll(S1map);

    //P4 permutation of the map
    List<Integer> P4map = permutation("P4",map);

    //XOR with left 4 bit
    XOR(P4map, inputList.subList(0,4));
}

//make sure all the input parameters are set in valid format
private boolean priorCheck(List<Integer> inputList){
    if(secretKey!=null && inputList!=null&&
            secretKey.size()==10 && inputList.size()==8){
        for(Integer digit: secretKey){
            if (digit.equals(0) || digit.equals(1)) continue;
            else {
                System.out.println("Error: secret key contains invalid number, has to be 0 or 1");
                return false;
            }
```

```
        }

        for(Integer digit: inputList){
            if (digit.equals(0) || digit.equals(1)) continue;
            else {
                System.out.println("Error: secret key contains invalid number, has to be 0 or 1");
                return false;
            }

        }
    }

    else{
        System.out.println("Error: secret key or plain text is not set correctly");
        return false;
    }

    return true;

}

public void encryption(){


    if (priorCheck(plainText)){
        System.out.print("The plain text is: " );
        print(plainText);
        //get the two subkeys
        List<Integer> subKey1 = getSubkey1();
        List<Integer> subKey2 = getSubkey2();

        //IP permutation
        List<Integer> listAfterIP = permutation("IP",plainText);

        //1st cycle
        cycle(subKey1, listAfterIP);

        //swap left and right bits
        swap(listAfterIP);
        System.out.print("After swap : ");
        print(listAfterIP);

        //2nd cycle
        cycle(subKey2, listAfterIP);

        //IPrev permutation
        cipherText = permutation("IPrev",listAfterIP);
        System.out.print("The cipher text is : ");
        print(cipherText);
```

```
    }


}

public void decryption(){

    if (priorCheck(cipherText)){

        System.out.print("The cipher text is : ");
        print(cipherText);
        //get the two subkeys
        List<Integer> subKey1 = getSubkey1();
        List<Integer> subKey2 = getSubkey2();

        //IP permutation
        List<Integer> listAfterIP = permutation("IP",cipherText);

        //1st cycle
        cycle(subKey2, listAfterIP);

        //swap left and right bits
        swap(listAfterIP);
        System.out.print("After swap : ");
        print(listAfterIP);

        //2nd cycle
        cycle(subKey1, listAfterIP);

        //IPrev permutation
        plainText = permutation("IPrev",listAfterIP);
        System.out.print("The plain text is: " );
        print(plainText);
    }


}

private void print(List<Integer> inputList) {
    for(Integer digit : inputList)
        System.out.print(digit);
    System.out.println();

}

public void modifyS0(List<Integer> modS0){
    S0 = modS0;
    permutations.put("S0",S0);
}
```

```
    public void resetS0() {
        S0 = Arrays.asList(1,0,3,2,
                           3,2,1,0,
                           0,2,1,3,
                           3,1,3,2);
        permutations.put("S0",S0);


    }

    //setter for plaintext and secretkey
    public void setPlainText(List<Integer> plainText) {
        this.plainText = plainText;
    }

    public void setSecretKey(List<Integer> secretKey) {
        this.secretKey = secretKey;
    }

    public void setCipherText(List<Integer> cipherText) {
        this.cipherText = cipherText;
    }

    public List<Integer> getPlainText() {
        return plainText;
    }

    public List<Integer> getSecretKey() {
        return secretKey;
    }

    public Map<String, List<Integer>> getPermutations() {
        return permutations;
    }


}
```

---

(3) Code output:

```
run:
Simplied DES Demo


1. Encryption
```

2. Decryption
3. Encryption with modified S0
4. Quit
Choose your operation:
1
Please input the plain text:
10111101
Please input the key:
1010000010
The plain text is: 10111101
After swap : 11101100
The cipher text is : 01110101

1. Encryption
2. Decryption
3. Encryption with modified S0
4. Quit
Choose your operation:
1
Please input the plain text:
11001110
Please input the key:
1001100101
The plain text is: 11001110
After swap : 00110001
The cipher text is : 01100110

1. Encryption
2. Decryption
3. Encryption with modified S0
4. Quit
Choose your operation:
3
Please input the plain text:
00100101
Please input the key:
1001011001
The plain text is: 00100101
After swap : 01000010
The cipher text is : 01001100

1. Encryption
2. Decryption
3. Encryption with modified S0
4. Quit
Choose your operation:

4
BUILD SUCCESSFUL (total time: 3 minutes 0 seconds)