**Part One: Implement the Affine Cipher**

**Compute all possible eligible keys you can use and justify your computation.**

The requirements for the keys are as follows:

* a can be 12 different values: 0 <= a <= 25 where a is co-prime to 26. This limits a to the following values: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25.
* b can be 26 different values: 0 <= b <= 25.

This means that the total number of keys is 12 \* 26 = 312 keys.

**If your code works properly with some selected eligible key, show recovered plaintext after decryption and compare with the original plaintext file; (the test file is given in the unit website). State the possible reasons if your code is not working properly.**

Contents of testfile-Affine.txt:

Inthispaperweconsidertheproblemofrobustfacerecognitionusingcolor

informationinthiscontextsparserepresentationbasedalgorithmsarethe

stateoftheartsolutionsforgrayfacialimageSproposedmodelthecontrolpar

ameterizationTechniquetOgetherwiththeconstrainttranscriptionmethodi

susedbytransformingtheproposedproblemintoasequenceofoptimalparameter

selectionproblemsFinallyapracticalexampleonbeersalesisusedtoshowtheeffectiveness

ofproposedmodelandwepresenttheoptimAladvertisingstrategiescorrespondingtodifferent

competitionsituationS

Contents of decryption file AffineMessage.txt with keys a:5 and b:9:

Inthispaperweconsidertheproblemofrobustfacerecognitionusingcolor

informationinthiscontextsparserepresentationbasedalgorithmsarethe

stateoftheartsolutionsforgrayfacialimageSproposedmodelthecontrolpar

ameterizationTechniquetOgetherwiththeconstrainttranscriptionmethodi

susedbytransformingtheproposedproblemintoasequenceofoptimalparameter

selectionproblemsFinallyapracticalexampleonbeersalesisusedtoshowtheeffectiveness

ofproposedmodelandwepresenttheoptimAladvertisingstrategiescorrespondingtodifferent

competitionsituationS

As we can see the plain text is recovered perfectly.

**Mathematically prove the decrypted message equals to the original message with critical logical reasoning.**

Encryption is achieved with the following equation: c = f(m) = (a m + b) mod 26. Decryption is achieved with the inverse of this function: m = f-1(c) = a-1 (c - b) mod 26. a-1 is the inverse of a and can calculated with the Extended Euclidean Algorithm: a a-1 mod 26 = 1.

**Use your code in the Tutorial 1, print out the letter distribution with a graph chart for the given test file.**

**Submit your code in your hard copy assignment with detailed code comments.**

Code is provided at the end of the report.

**Print out the first page and last page of decrypted file and compare them with the original plaintext, are they the same?**

Comparison shown above to be the same.

**Part Two: Implement DES**

**Mathematically prove DES works (After decryption, you can obtain the original plaintext).**

The formulas for each round of DES encryption and decryption are as follows:

* Encryption: Li=Ri-1, Ri=Li-1 ⊕ f(Ri-1, ki).
* Decryption: Ri-1=Li, Li-1=Ri ⊕ f(Li , ki).

The original encryption uses 16 steps to obtain the final cipher. After the last round the left half of the message and the right half of the message are swapped. Decryption in DES involves the exact same steps as the encryption process only the order that the 16 keys are applied is reversed. After the 16 steps of decryption the right half and the left half of the original message are obtained again. This process means that the original plaintext can be obtained again using the same key applied in reverse.

**State/depict your pseudo code structure based on the three separate required functions.**

The key generation:

function generateKeys(keyString)

initKey = convertToBinary(keyString)

key = PC-1 permutation of initKey

for i = 0 to 15

keyHalfOne = first half of key

keyHalfTwo = second half of key

tempH1One = keyHalfOne[0]

tempH1Two = keyHalfOne[1]

tempH2One = keyHalfTwo[0]

tempH2Two = keyHalfTwo[1]

for j = 0 to 27

if j + shifts[i] is less than 28

keyHalfOne[j] = keyHalfOne[j + shifts[i]]

keyHalfTwo[j] = keyHalfTwo[j + shifts[i]]

keyHalfOne[26] = tempH1One

keyHalfTwo[26] = tempH2One

keyHalfOne[27] = tempH1Two

keyHalfTwo[27] = tempH2Two

for j = 0 to 27

key[j] = keyHalfOne[j]

key[j + 28] = keyHalfTwo[j]

keyArray[i] = PC-2 permutation of key

return keyArray

The switch function:

function switchFunction(intArray, keyArray)

for i = 0 to 31

leftArray[i] = intArray[i]

rightArray[i] = intArray[i + 32]

for i = 0 to 15

tempArray = leftArray

leftArray = rightArray

rightArray = xor(tempArray, functionK(rightArray, keyArray[i]))

for i = 0 to 31

intArray[i] = rightArray[i]

intArray[i+32] = leftArray[i]

return intArray

The F(k) function:

function functionK(rightArray, keyArray)

eArray[i] = E-bit permutation of rightArray

xorArray = xor(eArray, keyArray)

for i = 0 to 7

for j = 0 to 5

subArray[j] = xorArray[(i\*6) + j]

row = subArray[0] + subArray[5]

col = subArray[1] + subArray[2] + subArray[3] + subArray[4]

value = sBox[i][col + (16 \* row)]

valueString = convert value to binary

for j = 0 to valueString.length()

preFinalArray[((4 – valueString.length()) + j) + (i\*4)] = valueString.charAt(j)

finalArray[i] = permutation of preFinalArray

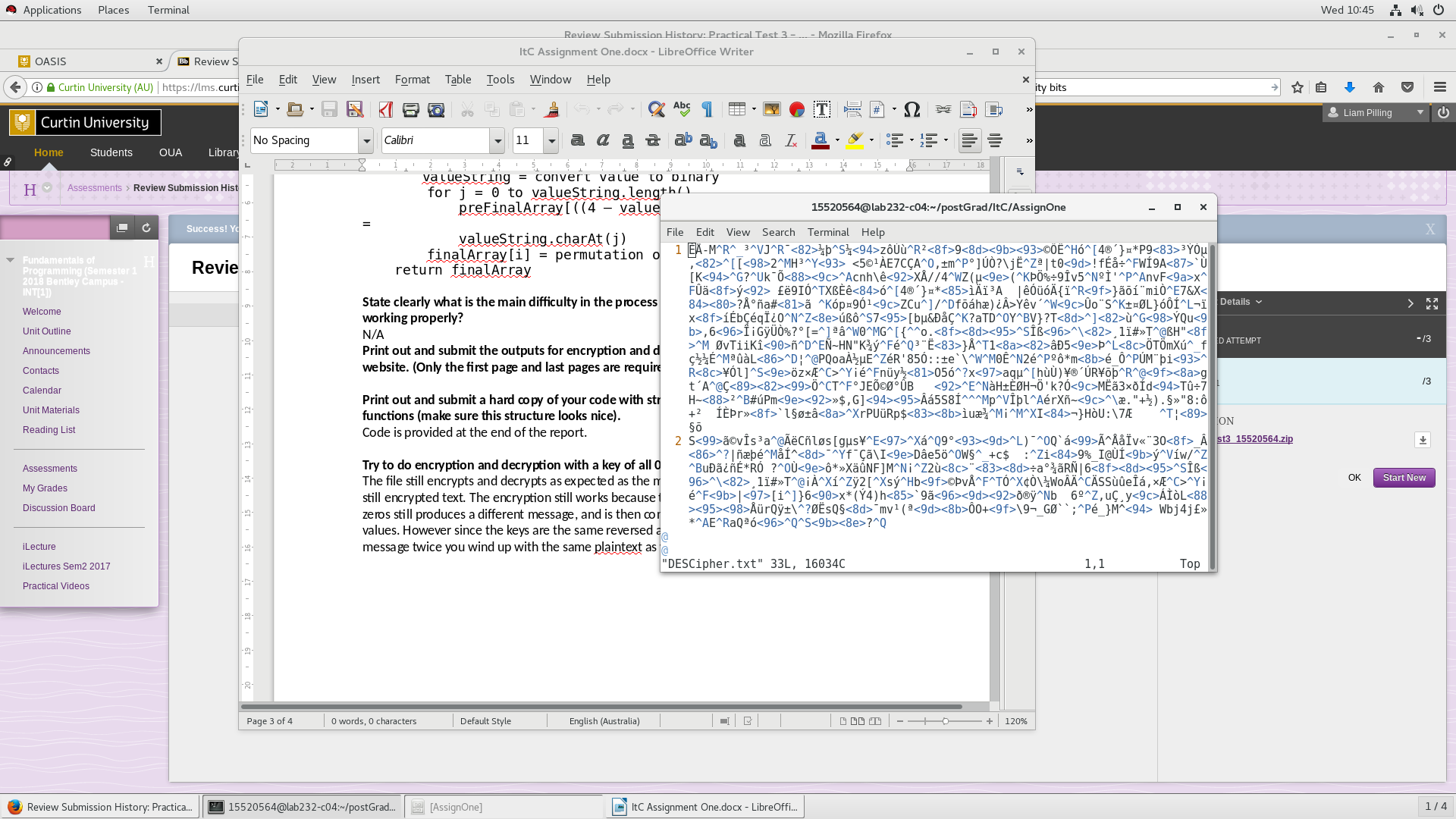
return finalArray

**State clearly what is the main difficulty in the process of your programming if your code is not working properly?**

N/A

**Print out and submit the outputs for encryption and decryption for a test file given in the unit website. (Only the first page and last pages are required for hard copy.)**

Encryption (As seen through vim due to issues with displaying binary values):

Decryption:

***Lines 1 – 31 (first page):***

For each set of fuzzy terms, $A \subseteq M$, $\prod\_{m\in

A}m$ represents a conjunction of the fuzzy terms in $A$. For

instance,

let $A=\{m\_{1,2},m\_{2,1},m\_{4,2}\}\subseteq M$, a new

fuzzy concept ``$m\_{1,2}$ and $m\_{2,1}$ and $m\_{4,2}$" with the linguist

interpretation ``\emph{short sepal and wide sepal and narrow petal}"

can be represented as $\prod\_{m\in

A}m=m\_{1,2}m\_{2,1}m\_{4,2}$. Then the fuzzy rules can be represented as follows:

\bigskip

\textbf{Rule} $R\_1$ : If $x$ is

$m\_{1,2}m\_{2,1}m\_{4,2}$, then $x$ belongs to Class 1;

\textbf{Rule} $R\_2$ : If $x$ is $m\_{2,1}m\_{3,2}$, then $x$ belongs

to Class 1;

\textbf{Rule} $R\_3$ : If $x$ is $m\_{1,2}m\_{4,2}$, then $x$ belongs

to Class 1.

\bigskip

\noindent Then, the antecedent of three fuzzy rules $R\_1, R\_2, R\_3$ for Class

1 can be represented by ``or" as follows:

\bigskip

\textbf{Rule} $R$ : If $x$ is ``$m\_{1,2}m\_{2,1}m\_{4,2}$ or

$m\_{2,1}m\_{3,2}$ or $m\_{1,2}m\_{4,2}$", then $x$ belongs to Class 1.

\bigskip

$\sum^{r}\_{u=1}(\prod\_{m\in A\_u}m)$, which is a formal sum of the

***Lines 184 – 204 (last page):***

A\_{i}}m) \in EM$, if the membership function of $\xi$ is defined in (\ref{miu3}), then $\{\mu\_{\xi}(x) | \xi\in EM \}$ is a set of coherence

membership functions of $(EM, \wedge, \vee)$

\begin{equation}

\mu\_{\xi}(x)=\sup\_{i\in I}\inf\_{\gamma\in A\_i}\frac{\sum\_{u\in

A^\succeq\_{i}(x)}\rho\_{\gamma}(u)}{\sum\_{u\in

X}\rho\_{\gamma}(u)}, \ \ \ \forall x\in X, \label{miu3}

\end{equation}

where $\rho\_{\gamma}(x)$ is a weight function of fuzzy term $\gamma$ and $A^\succeq\_{i}(x)$ is defined in (\ref{under-A}).

In general, the dataset $X$ is assumed to be a finite observations from

a probability

space $(\Omega, \mathcal{F}, \mathcal{P})$. Thus the above membership function (\ref{miu3}) can be extended to whole space $\Omega$ as follows

\begin{equation}

\mu\_{\xi}(x)=\sup\_{i\in I}\inf\_{\gamma\in A\_{i}}\frac{\int\_{

A^\succeq\_{i}(x)}\rho\_{\gamma}(t)d\mathcal{P}(t)}{\int\_{\Omega}\rho\_{\gamma}(t)d\mathcal{P}(t)}, \ \ \ \

\forall x\in \Omega, \label{miu4}

\end{equation}

In \cite{xref:/book/book}, it has been proven that the membership functions defined by (\ref{miu4}) are also coherent membership functions and the

membership function defined by (\ref{miu3}) converges to that defined by (\ref{miu4}), for all $x\in \Omega$

as $|X|$ approaches to infinity. This ensures that the laws discovered based on the membership functions and their

logic operations determined by the finite observed data $X$ (\ref{miu3}) can be extended to the whole space $\Omega$ by membership functions (\ref{miu4}) and analyzed in the framework of probability theory.

**Print out and submit a hard copy of your code with structure explanation by using the three functions (make sure this structure looks nice).**

For this code the structure was done in a way that re used code all had their own functions and that the program separated functionality to improve readability, so no one function was doing more than it should to keep things simple. The only function that doesn't follow this is the main which has to initialise values, start the DES process for the strings and then output results. Code is provided at the end of the report.

**Try to do encryption and decryption with a key of all 0’s, and report your findings.**

The file still encrypts and decrypts as expected as the message is still the same and the cipher text is still encrypted text. The encryption still works because the XoR of the binary message and the key of zeros still produces a different message, and is then converted using the S-Box to give us unique values. However since the keys are the same reversed as they are all zeros if you encrypt the message twice you wind up with the same plaintext as your cipher text.

**Part Three: Additional Questions**

**How many types of threats in information transmission?**

There are two types of threats in information transmission: passive attacks and active attacks. There are two types of passive attacks:

* Cryptanalysis, where an encrypted message is obtained, decrypted and released.
* Traffic analysis, where traffic is analysed based on the information given from transmissions, such as frequency of transmission and where traffic comes from/where it goes.

There are four types of active attacks:

* Replay attacks, where an attacker reuses a message that may be unauthorised or expired in order to influence and disrupt normal traffic.
* Masquerade attacks, where one entity tries to impersonate another entity for illegal purposes.
* Modification attacks, where contents of messages are modified illegally.
* Denial of service attacks, where a system is made to deny requests from traffic due to heavily increased traffic.

**What is source coding in information transmission?**

Source coding is the modifications made to data before it is transmitted across a network. This can be in the form of encryption, where data is encrypted at the source and then decrypted at the destination.

**What is coding/error coding in information transmission?**

Error coding is the modifications made to data in order to lower the probabilities of errors occurring. This is due to transmission channels being noisy which makes the source code vulnerable to unwanted modifications from the transmission. A simple form of this is a parity bit which can set the number of zeros to be even by adding a zero at the end so that the destination can detect if a zero was lost.

**Explain information rate and Hamming distance in coding theory with your own words.**

Information rate is the rate at which a processor can process the information received in a set of binary code. The lower the length of the words and the lower the number of words in a set the faster the binary code can be processed. This can be given a value using the following equation:

(1/n)log2|C| where n is the length of C.

Hamming distance is the distance between two words where they differ. For example, the String Cooler and the string Codder have a distance of 2 because the length of where they differ, the (ol) and the (dd) is 2.

**In your code of DES, which step is source coding? Which step is coding/error coding?**

In the code the source coding step is the switch function because that is where we are applying the modifications in the code in order to prepare it for transmission. The error coding isn't performed in the code, as the 8 parity bits aren't being saved or utilised. If they were then they could be used to check for errors in the transmission of the key with an odd parity check.

**Code For Affine Cipher**

**Key.java**

/\*

This is a file to create the A and B keys into a key object from a user input.

This input is then checked with to make sure that A is a coprime of 26 and

that B is a value between 0 and 25

\*/

import java.util.\*;

public class Key

{

private int aKey;

private int invA;

private int bKey;

// This array stores all of the co primes of 26 and their corresponding

// inverse according to a \* a-1 mod 26 = 1.

private Integer[][] allAValues = {{1, 3, 5, 7, 9,11,15,17,19,21,23,25},

{1, 9,21,15, 3,19, 7,23,11, 5,17,25}};

private Scanner keyboard = new Scanner(System.in);

public Key()

{

setKeys();

}

// This takes user input for the keys and then checks if they are valid. This

// loops until valid integers are entered.

private void setKeys()

{

System.out.println("Please enter your A and B keys:");

aKey = keyboard.nextInt();

bKey = keyboard.nextInt();

List<Integer> aValues = Arrays.asList(Arrays.asList(allAValues).get(0));

while(!aValues.contains(aKey))

{

System.out.println("Please enter valid A key");

aKey = keyboard.nextInt();

}

while((bKey < 0) || (bKey > 25))

{

System.out.println("Enter a valid B key");

bKey = keyboard.nextInt();

}

for(int i = 0; i < allAValues[0].length; i++)

{

if(aKey == allAValues[0][i])

invA = allAValues[1][i];

}

}

public int getA()

{

return aKey;

}

public int getB()

{

return bKey;

}

public int getInvA()

{

return invA;

}

}

**Affine.java**

/\*

Authour: Liam Pilling

Last Date Modified: 15/04/2018

Description: This file takes in a text file with a message to be encrypted

and then takes in two keys which are checked in the file Key.java. The

message is then encrypted with the two keys using the Affine cypher and

written to file. It is then decrypted and printed to written to file.

\*/

import java.util.\*;

import java.io.\*;

public class Affine

{

//The main function first reads the file and then calls the Key.java file

//to input keys. It then encrypts the message and prints the cipher text and

//then decrypts the message and prints the message.

public static void main(String[] args)

{

String filename = null, cipher = null;

Scanner keyboard = new Scanner(System.in);

int c;

StringBuilder message = new StringBuilder();

System.out.println("Enter your filename");

filename = keyboard.nextLine();

//Read in the file.

try

{

FileReader fr = new FileReader(filename);

BufferedReader br = new BufferedReader(fr);

while((c = br.read()) != -1)

message.append((char)c);

}

catch(IOException e)

{

System.out.println(e);

}

Key keys = new Key(); //Takes in two keys and checks validity

cipher = getCipher(message.toString(), keys); //Get the cipher

System.out.println("Cipher text written to file AffineCipher.txt");

writeToFile(cipher, "AffineCipher.txt"); //Write it to a file

System.out.println("Decrypted text written to file AffineMessage.txt");

//Write the message to a file

writeToFile(decryptCipher(cipher,keys), "AffineMessage.txt");

}

//Function to encrypt the message. It takes in a string message and the keys

//and then performs the encryption

public static String getCipher(String message, Key keys)

{

StringBuilder cipher = new StringBuilder();

for (int i = 0; i < message.length(); i++)

{

//Checks first if letter is uppercase or lowercase

if((message.charAt(i) >= 'A') && (message.charAt(i) <= 'Z'))

cipher.append(((char)('A' + (keys.getA() \*

(message.charAt(i) - 'A') + keys.getB()) % 26)));

else if((message.charAt(i) >= 'a') && (message.charAt(i) <= 'z'))

cipher.append(((char)('a' + (keys.getA() \*

(message.charAt(i) - 'a') + keys.getB()) % 26)));

else

cipher.append(message.charAt(i)); //Ignoring other characters

}

return cipher.toString();

}

//Function to decrypt the message. It takes in a string cipher and the keys

//and then performs the decryption

public static String decryptCipher(String cipher, Key keys)

{

StringBuilder message = new StringBuilder();

for (int i = 0; i < cipher.length(); i++)

{

//Checks first if letter is uppercase or lowercase

if((cipher.charAt(i) >= 'A') && (cipher.charAt(i) <= 'Z'))

message.append((char)('A' + ((((keys.getInvA() \*

((cipher.charAt(i) - 'A') - keys.getB())) % 26)) + 26 ) % 26));

else if((cipher.charAt(i) >= 'a') && (cipher.charAt(i) <= 'z'))

message.append((char)('a' + ((((keys.getInvA() \*

((cipher.charAt(i) - 'a') - keys.getB())) % 26)) + 26 ) % 26));

else

message.append(cipher.charAt(i)); //Ignoring other characters

}

return message.toString();

}

// This is a function to write a string to a given filename

public static void writeToFile(String text, String filename)

{

try

{

PrintWriter writer = new PrintWriter(filename, "UTF-8");

writer.print(text);

writer.close();

}

catch(IOException e)

{

System.out.println(e);

}

}

}

**Code For The DES cipher**

**Des.java**

/\*

Authour: Liam Pilling

Last Date Modified: 18/04/2018

Description: This file takes in a text file with a message to be encrypted

and then takes in a key which is cut/padded to 8 characters. It then

performs the DES encryption and key generation to produce a chiphertext. It

then performs the DES function a second time but with the keys in reverse

order to decrypt the cipher text back into the message.

\*/

import java.util.\*;

import java.io.\*;

public class DES

{

private static int[] pCOne = {57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34,

26, 18, 10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52,

44, 36, 63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38,

30, 22, 14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20,

12, 4};

private static int[] shifts = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

private static int[] pCTwo = {14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23,

19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2, 41, 52,

31, 37, 47, 55, 30, 40, 51, 45, 33, 48, 44, 49,

39, 56, 34, 53, 46, 42, 50, 36, 29, 32};

private static int[] initPerm={58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36,

28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64,

56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25,

17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3, 61, 53,

45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23,

15, 7 };

private static int[] finPerm= {40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55,

23, 63, 31, 38, 6, 46, 14, 54, 22, 62, 30, 37, 5,

45, 13, 53, 21, 61, 29, 36, 4, 44, 12, 52, 20,

60, 28, 35, 3, 43, 11, 51, 19, 59, 27, 34, 2, 42,

10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57,

25};

private static int[] eTable = {32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10,

11, 12, 13, 12, 13, 14, 15, 16, 17, 16, 17, 18,

19, 20, 21, 20, 21, 22, 23, 24, 25, 24, 25, 26,

27, 28 ,29, 28, 29, 30, 31, 32, 1};

private static int[][] sBox ={{14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9,

0, 7, 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11,

9, 5, 3, 8, 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9,

7, 3, 10, 5, 0, 15, 12, 8, 2, 4, 9, 1, 7, 5, 11,

3, 14, 10, 0, 6, 13},

{15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5,

10, 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6,

9, 11, 5, 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12,

6, 9, 3, 2, 15, 13, 8, 10, 1, 3, 15, 4, 2, 11, 6,

7, 12, 0, 5, 14, 9},

{10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4,

2, 8, 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12,

11, 15, 1, 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2,

12, 5, 10, 14, 7, 1, 10, 13, 0, 6, 9, 8, 7, 4,

15, 14, 3, 11, 5, 2, 12},

{7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4,

15, 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1,

10, 14, 9, 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3,

14, 5, 2, 8, 4, 3, 15, 0, 6, 10, 1, 13, 8, 9, 4,

5, 11, 12, 7, 2, 14},

{2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0,

14, 9, 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10,

3, 9, 8, 6, 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12,

5, 6, 3, 0, 14, 11, 8, 12, 7, 1, 14, 2, 13, 6,

15, 0, 9, 10, 4, 5, 3},

{12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5,

11, 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0,

11, 3, 8, 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10,

1, 13, 11, 6, 4, 3, 2, 12, 9, 5, 15, 10, 11, 14,

1, 7, 6, 0, 8, 13},

{4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10,

6, 1, 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2,

15, 8, 6, 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6,

8, 0, 5, 9, 2, 6, 11, 13, 8, 1, 4, 10, 7, 9, 5,

0, 15, 14, 2, 3, 12},

{13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0,

12, 7, 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11,

0, 14, 9, 2, 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10,

13, 15, 3, 5, 8, 2, 1, 14, 7, 4, 10, 8, 13, 15,

12, 9, 0, 3, 5, 6, 11}};

private static int[] rightPerm = {16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26,

5, 18, 31, 10, 2, 8, 24, 14, 32, 27, 3, 9, 19,

13, 30, 6, 22, 11, 4, 25};

//The main function is responsible for calling the key generation function,

//reading a text file and the sending each of the 8 characters from the file

//to the DES algorithm to be encrypted. It then prints the cipher text to

//the screen before sending the cipher text back into the algorithm with the

//16 subkeys in reverse order to obtain the original message.

public static void main(String[] args)

{

String plainText;

StringBuilder subString = new StringBuilder();

StringBuilder cipherText = new StringBuilder();

StringBuilder decryptMessage = new StringBuilder();

String filename;

int c, length;

Scanner keyboard = new Scanner(System.in);

int[] intArray = new int[64];

int[][] keyArray;

int[][] decipherKeyArray = new int[16][48];

String key;

System.out.println("Enter your key: ");

key = keyboard.nextLine();

key = key.substring(0, Math.min(key.length(), 8));

keyArray = generateKeys(key);

System.out.println("Enter your filename:");

filename = keyboard.nextLine();

plainText = readFile(filename);

length = plainText.length();

//Loop for every 8 characters in the plaintext

for(int i = 0; i < plainText.length(); i++)

{

subString.append(plainText.charAt(i));

if(((i+1) % 8 == 0) && (i > 0))

{

cipherText.append(startDES(subString.toString(), keyArray));

subString = new StringBuilder();

}

}

if(subString.length() != 0)

cipherText.append(startDES(subString.toString(), keyArray));

writeToFile(cipherText.toString(), "DESCipher.txt");

System.out.println("Cipher text written to DESCipher.txt");

String cipher = readFile("DESCipher.txt");

for(int i = 0; i < 16; i++)

decipherKeyArray[i] = keyArray[15-i];

subString = new StringBuilder();

//Loop for every 8 character string in the cipher

for(int i = 0; i < cipher.length(); i++)

{

subString.append(cipher.charAt(i));

if(((i+1) % 8 == 0) && (i > 0))

{

decryptMessage.append(startDES(subString.toString(),

decipherKeyArray));

subString = new StringBuilder();

}

}

if(subString.length() != 0)

decryptMessage.append(startDES(subString.toString(),

decipherKeyArray));

//We trim this to prevent the file from printing the padded null values

writeToFile(decryptMessage.toString().substring(0, length),

"DESMessage.txt");

System.out.println("Decrypted message written to file DESMessage.txt");

}

//This is the function that converts a string of characters into 16 48-bit

//sub keys. The string is converted to binary before being permutated and

//split in half. Then, for 16 steps, they are shifted left one or two spaces

//and joined and permutated forming each sub key.

public static int[][] generateKeys(String keyString)

{

int[] key = new int[56];

int[][] keyArray = new int[16][48];

int[] keyHalfOne = new int[28];

int[] keyHalfTwo = new int[28];

int[] initKey = new int[64];

//Covert our string to a 64 bit array with padding/chopping

initKey = convertToBinary(keyString);

//Apply the permutation PC-1

for(int i = 0; i < pCOne.length; i++)

key[i] = initKey[(pCOne[i]-1)];

for(int i = 0; i < 16; i++)

{

//Split the key into two halves

for(int j = 0; j < 28; j++)

{

keyHalfOne[j] = key[j];

keyHalfTwo[j] = key[j + 28];

}

int tempH1One = keyHalfOne[0];

int tempH1Two = keyHalfOne[1];

int tempH2One = keyHalfTwo[0];

int tempH2Two = keyHalfTwo[1];

//Here we shift our values according to the shift table

for(int j = 0; j < 28; j++)

{

if(j + shifts[i] < 28)

{

keyHalfOne[j] = keyHalfOne[j + shifts[i]];

keyHalfTwo[j] = keyHalfTwo[j + shifts[i]];

}

}

//Here we fix up our shift for overwritten values

keyHalfOne[26] = tempH1One;

keyHalfTwo[26] = tempH2One;

keyHalfOne[27] = tempH1Two;

keyHalfTwo[27] = tempH2Two;

//Put the two strings back together

for(int j = 0; j < (28); j++)

{

key[j] = keyHalfOne[j];

key[j + 28] = keyHalfTwo[j];

}

//Apply the permutation PC-2 and place it into the key array

for(int j = 0; j < pCTwo.length; j++)

keyArray[i][j] = key[(pCTwo[j]-1)];

}

return keyArray;

}

//This is the function called to start the procedure of encryping each 8

//character long sub strings and then performing the intial permutation.

//They are then send to the switch function before being returned as a

//string again.

public static String startDES(String subString, int[][] keyArray)

{

int[] intArray = new int[64];

int[] initPermArray = new int[64];

int[] finalArray = new int[64];

intArray = convertToBinary(subString);

//Apply the initial permuation IP

for(int i = 0; i < initPerm.length; i++)

initPermArray[i] = intArray[(initPerm[i]-1)];

//Send the array to the switch function

intArray = switchFunction(initPermArray, keyArray);

//Apply the final permutation IP inverse

for(int i = 0; i < finalArray.length; i++)

finalArray[i] = intArray[(finPerm[i]-1)];

return convertToString(finalArray);

}

//This is a function that converts a 8 character long String into a 64-bit

//array, providing padding if less than 8 characters long.

public static int[] convertToBinary(String subString)

{

String tempS = null;

int start = 0;

int[] intArray = new int[64];

for(int i = 0; i < subString.length(); i++)

{

tempS = Integer.toBinaryString(subString.charAt(i));

start = i \* 8;

if(tempS.length() != 8)

start = (8 - tempS.length()) + (i\*8);

for(int j = 0; j < tempS.length(); j++)

intArray[j + start] = Integer.parseInt(String.valueOf(

tempS.charAt(j)));

}

return intArray;

}

//This is the Switch function that takes in the binary array and the 2D

//array of keys. It then splits the binary array, does the 16 steps

//involving the XOR function on the left side and the f(k) on the right side

// and then permutates the final value.

public static int[] switchFunction(int[] intArray, int[][] keyArray)

{

int[] leftArray = new int[32];

int[] rightArray = new int[32];

int[] tempArray;

//Split into a left array and a right array.

for(int i = 0; i < 32; i++)

{

leftArray[i] = intArray[i];

rightArray[i] = intArray[i + 32];

}

//Make the Ln = Rn-1 and Rn = Ln-1 + f(Rn-1, Kn)

for(int i = 0; i < 16; i ++)

{

tempArray = leftArray;

leftArray = rightArray;

rightArray = xor(tempArray, functionK(rightArray, keyArray[i]));

}

//Put the two halves back into one array

for(int i = 0; i < 32; i++)

{

intArray[i] = rightArray[i];

intArray[i+32] = leftArray[i];

}

return intArray;

}

//This is the f(k) function that takes in a binary array and a key array

//and performs the initial permutation, XOR the two arrays, the conversion

//to the corresponding S-Box value and the final Permutation.

public static int[] functionK(int[] rightArray, int[] keyArray)

{

int[] eArray = new int[48];

int[] xorArray;

int[] subArray = new int[6];

int[] preFinalArray = new int[32];

int[] finalArray = new int[32];

int value;

//This is the E-Bit table permutation

for(int i = 0; i < eArray.length; i++)

eArray[i] = rightArray[(eTable[i]-1)];

//We then XOR the values with the given key

xorArray = xor(eArray, keyArray);

//Here we iterate over all the S-Boxes for the 8 subsets from our array

for(int i = 0; i < 8; i++)

{

StringBuilder col = new StringBuilder();

StringBuilder row = new StringBuilder();

//Take the 6 bits needed to find the column and row of the S-Box

for(int j = 0; j < 6; j++)

subArray[j] = xorArray[(i\*6) + j];

row.append(subArray[0]);

row.append(subArray[5]);

col.append(subArray[1]);

col.append(subArray[2]);

col.append(subArray[3]);

col.append(subArray[4]);

//Here we refer to the S-Boxes for our new values

value = sBox[i][(Integer.parseInt(col.toString(), 2)) +

(16 \* Integer.parseInt(row.toString(), 2))];

//Convert the S-Box value to binary and pad if necessary

String valueString = Integer.toBinaryString(value);

for(int j = 0; j < valueString.length(); j++)

preFinalArray[((4 - valueString.length()) + j) + (i\*4)] =

Integer.parseInt(String.valueOf(valueString.charAt(j)));

}

//Apply the permutation for the 32 bit block

for(int i = 0; i < finalArray.length; i++)

finalArray[i] = preFinalArray[(rightPerm[i]-1)];

return finalArray;

}

//This function converts an array of binary values into their ASCII

//equivalents.

public static String convertToString(int[] intArray)

{

StringBuilder binText = new StringBuilder();

StringBuilder text = new StringBuilder();

//This loop takes every 8 bits and converts them to an ASCII character

for(int i = 0; i < 8; i++)

{

binText = new StringBuilder();

for(int j = 0; j < 8; j++)

binText.append(Integer.toString(intArray[(8\*i) + j]));

text.append(Character.toString(

(char)Integer.parseInt(binText.toString(), 2)));

}

return text.toString();

}

// This is a function for performing the XOR operation on two arrays.

public static int[] xor(int[] arrayOne, int[] arrayTwo)

{

int[] returnArray = new int[arrayOne.length];

for(int i = 0; i < arrayOne.length; i++)

{

if(arrayOne[i] == arrayTwo[i])

returnArray[i] = 0;

else

returnArray[i] = 1;

}

return returnArray;

}

//This is a function to read a file to a string

public static String readFile(String filename)

{

StringBuilder plainText = new StringBuilder();

int c;

try

{

BufferedReader br = new BufferedReader(

new InputStreamReader(

new FileInputStream(filename), "UTF8"));

while((c = br.read()) != -1)

plainText.append((char)c);

}

catch(IOException e)

{

System.out.println(e);

}

return plainText.toString();

}

//This is a function to write a string to a given filename

public static void writeToFile(String text, String filename)

{

try

{

PrintWriter writer = new PrintWriter(filename, "UTF-8");

writer.print(text);

writer.close();

}

catch(IOException e)

{

System.out.println(e);

}

}

}