1. Implementation of secure/Secret Communication of data

import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.math.BigInteger;

import java.security.Key;

import java.security.KeyFactory;

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.PrivateKey;

import java.security.PublicKey;

import java.security.spec.RSAPrivateKeySpec;

import java.security.spec.RSAPublicKeySpec;

import java.util.Base64;

import javax.crypto.Cipher;

public class RSA\_Read\_Write\_Key

{

static String plainText = "Plain text which need to be encrypted by Java RSA Encryption in ECB Mode";

public static void main(String[] args) throws Exception

{

// Get an instance of the RSA key generator

KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");

keyPairGenerator.initialize(4096);

// Generate the KeyPair

KeyPair keyPair = keyPairGenerator.generateKeyPair();

// Get the public and private key

PublicKey publicKey = keyPair.getPublic();

PrivateKey privateKey = keyPair.getPrivate();

// Get the RSAPublicKeySpec and RSAPrivateKeySpec

KeyFactory keyFactory = KeyFactory.getInstance("RSA");

RSAPublicKeySpec publicKeySpec = keyFactory.getKeySpec(publicKey, RSAPublicKeySpec.class);

RSAPrivateKeySpec privateKeySpec = keyFactory.getKeySpec(privateKey, RSAPrivateKeySpec.class);

// Saving the Key to the file

saveKeyToFile("public.key", publicKeySpec.getModulus(), publicKeySpec.getPublicExponent());

saveKeyToFile("private.key", privateKeySpec.getModulus(), privateKeySpec.getPrivateExponent());

System.out.println("Original Text : " + plainText);

// Encryption

byte[] cipherTextArray = encrypt(plainText, "D:\\sts-3.8.3.RELEASE\\Workspace\\Encryption\\public.key");

String encryptedText = Base64.getEncoder().encodeToString(cipherTextArray);

System.out.println("Encrypted Text : " + encryptedText);

// Decryption

String decryptedText = decrypt(cipherTextArray, "D:\\sts-3.8.3.RELEASE\\Workspace\\Encryption\\private.key");

System.out.println("DeCrypted Text : " + decryptedText);

}

public static void saveKeyToFile(String fileName, BigInteger modulus, BigInteger exponent) throws IOException

{

ObjectOutputStream ObjOutputStream = new ObjectOutputStream(

new BufferedOutputStream(new FileOutputStream(fileName)));

try

{

ObjOutputStream.writeObject(modulus);

ObjOutputStream.writeObject(exponent);

} catch (Exception e)

{

e.printStackTrace();

} finally

{

ObjOutputStream.close();

}

}

public static Key readKeyFromFile(String keyFileName) throws IOException

{

Key key = null;

InputStream inputStream = new FileInputStream(keyFileName);

ObjectInputStream objectInputStream = new ObjectInputStream(new BufferedInputStream(inputStream));

try

{

BigInteger modulus = (BigInteger) objectInputStream.readObject();

BigInteger exponent = (BigInteger) objectInputStream.readObject();

KeyFactory keyFactory = KeyFactory.getInstance("RSA");

if (keyFileName.startsWith("public"))

key = keyFactory.generatePublic(new RSAPublicKeySpec(modulus, exponent));

else

key = keyFactory.generatePrivate(new RSAPrivateKeySpec(modulus, exponent));

} catch (Exception e)

{

e.printStackTrace();

} finally

{

objectInputStream.close();

}

return key;

}

public static byte[] encrypt(String plainText, String fileName) throws Exception

{

Key publicKey = readKeyFromFile("public.key");

// Get Cipher Instance

Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWITHSHA-512ANDMGF1PADDING");

// Initialize Cipher for ENCRYPT\_MODE

cipher.init(Cipher.ENCRYPT\_MODE, publicKey);

// Perform Encryption

byte[] cipherText = cipher.doFinal(plainText.getBytes());

return cipherText;

}

public static String decrypt(byte[] cipherTextArray, String fileName) throws Exception

{

Key privateKey = readKeyFromFile("private.key");

// Get Cipher Instance

Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWITHSHA-512ANDMGF1PADDING");

// Initialize Cipher for DECRYPT\_MODE

cipher.init(Cipher.DECRYPT\_MODE, privateKey);

// Perform Decryption

byte[] decryptedTextArray = cipher.doFinal(cipherTextArray);

return new String(decryptedTextArray);

}

}

2. Experiment on claiming ownership of digital entity

**GenrateDigitalSignature.java**

import java.io.\*;

import java.security.\*;

public class GenerateDigitalSignature

{

public static void main(String args[])

{

/\* Generate a DSA signature \*/

if (args.length != 1)

{

System.out.println("Usage: nameOfFileToSign");

}

else try

{

/\* Generate a key pair \*/

KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA", "SUN");

SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");

keyGen.initialize(1024, random);

KeyPair pair = keyGen.generateKeyPair();

PrivateKey priv = pair.getPrivate();

PublicKey pub = pair.getPublic();

/\* Create a Signature object and initialize it with the private key \*/

Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");

dsa.initSign(priv);

/\* Update and sign the data \*/

FileInputStream fis = new FileInputStream("F:\\Digital Signature Demo\\digital.txt");

BufferedInputStream bufin = new BufferedInputStream(fis);

byte[] buffer = new byte[1024];

int len;

while (bufin.available() != 0)

{

len = bufin.read(buffer);

dsa.update(buffer, 0, len);

};

bufin.close();

/\* Now that all the data to be signed has been read in,

generate a signature for it \*/

byte[] realSig = dsa.sign();

/\* Save the signature in a file \*/

FileOutputStream sigfos = new FileOutputStream("F:\\Digital Signature Demo\\signature.txt");

sigfos.write(realSig);

sigfos.close();

/\* Save the public key in a file \*/

byte[] key = pub.getEncoded();

FileOutputStream keyfos = new FileOutputStream("F:\\Digital Signature Demo\\publickey.txt");

keyfos.write(key);

keyfos.close();

}

catch (Exception e)

{

System.err.println("Caught exception " + e.toString());

}

};

}

**VerifytheSignature.java**

import java.io.\*;

import java.security.\*;

import java.security.spec.\*;

public class VerifyDigitalSignature

{

public static void main(String args[])

{

/\* Verify a DSA signature \*/

if (args.length != 3)

{

System.out.println("Usage: publickeyfile signaturefile datafile");

}

else try

{

/\* import encoded public key \*/

FileInputStream keyfis = new FileInputStream("F:\\Digital Signature Demo\\publickey.txt");

byte[] encKey = new byte[keyfis.available()];

keyfis.read(encKey);

keyfis.close();

X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(encKey);

KeyFactory keyFactory = KeyFactory.getInstance("DSA", "SUN");

PublicKey pubKey = keyFactory.generatePublic(pubKeySpec);

/\* input the signature bytes \*/

FileInputStream sigfis = new FileInputStream("F:\\Digital Signature Demo\\signature.txt");

byte[] sigToVerify = new byte[sigfis.available()];

sigfis.read(sigToVerify );

sigfis.close();

/\* create a Signature object and initialize it with the public key \*/

Signature sig = Signature.getInstance("SHA1withDSA", "SUN");

sig.initVerify(pubKey);

/\* Update and verify the data \*/

FileInputStream datafis = new FileInputStream("F:\\Digital Signature Demo\\digital.txt");

BufferedInputStream bufin = new BufferedInputStream(datafis);

byte[] buffer = new byte[1024];

int len;

while (bufin.available() != 0)

{

len = bufin.read(buffer);

sig.update(buffer, 0, len);

};

bufin.close();

boolean verifies = sig.verify(sigToVerify);

System.out.println("signature verifies: " + verifies);

}

catch (Exception e)

{

System.err.println("Caught exception " + e.toString());

};

}

}

3. Implementation of tracing the digital theft in cyberspace

Phishing

import java.io.IOException;

import java.nio.file.Paths;

import java.util.NoSuchElementException;

import java.util.Scanner;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public final class Phishing {

public static String[] words =

{"win", "instant", "credit card", "income", "amaz", "natural",

"congratulations", "billion", "dirt", "cash", "viagra", "drastic",

"financ", "trial", "fast", "guarantee", "free", "membership",

"secret", "trick", "growth", "dirt", "sex", "prince",

"help", "phone", "partner", "fat", "profit", "invest"};

public static int[] scores = {0, 2, 4, 2, 4, 4, 3, 2, 5, 5,

6, 2, 2, 2, 3, 4, 4, 5, 5, 5,

5, 5, 5, 6, 4, 4, 2, 5, 5, 5};

public static int score = 0;

public static void main(String[] args){

if (args.length < 1)

return;

try (Scanner input = new Scanner(Paths.get(args[0]))) {

String text = input.useDelimiter("\\A").next().toLowerCase(); // or...

Matcher textWords = Pattern.compile("\\w+").matcher(text);

double size = 0;

while (textWords.find())

size++;

// overflow, density vs length, etc. but I'm tired

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

Matcher mat = Pattern.compile(words[i]+"\\w\*").matcher(text);

while (mat.find())

score += scores[i];

}

System.out.printf("File has a score of %.2f/10.%n", 10\*size/score);

} catch (NoSuchElementException e) {

// with one-line, this means empty file

System.out.println("File has a score of 0.");

} catch (IOException e) {

System.err.println("Error opening file. Terminating.");

}

}

}

// ... you can do this

// StringBuilder text = new StringBuilder();

// while (input.hasNext())

// text.append(input.next());

4. Implementation of application in Block Codes

// import required classes and packages

package javaTpoint.JavaExample;

import java.util.\*;

// create HammingCodeExample class to implement the Hamming Code functionality in Java

class HammingCodeExample {

// main() method start

public static void main(String args[])

{

// declare variables and array

int size, hammingCodeSize, errorPosition;

int arr[];

int hammingCode[];

// create scanner class object to take input from user

Scanner sc = new Scanner(System.in);

System.out.println("Enter the bits size for the data.");

size = sc.nextInt();

// initialize array

arr = new int[size];

// get data from user which we want to transfer

for(int j = 0 ; j < size ; j++) {

System.out.println("Enter " + (size - j) + "-bit of the data:");

// fill array with user entered data

arr[size - j - 1] = sc.nextInt();

}

// print the user entered data

System.out.println("The data which you enter is:");

for(int k = 0 ; k < size ; k++) {

System.out.print(arr[size - k - 1]);

}

System.out.println(); // for next line

// call getHammingCode() method and store its return value to the hammingCode array

hammingCode = getHammingCode(arr);

hammingCodeSize = hammingCode.length;

System.out.println("The hamming code generated for your data is:");

for(int i = 0 ; i < hammingCodeSize; i++) {

System.out.print(hammingCode[(hammingCodeSize - i - 1)]);

}

System.out.println(); // for next line

// The added parity bits are the difference b/w the original data and the returned hammingCode

System.out.println("For detecting error at the reciever end, enter position of a bit to alter original data "

+ "(0 for no error):");

errorPosition = sc.nextInt();

// close Scanner class object

sc.close();

// check whether the user entered position is 0 or not.

if(errorPosition != 0) {

// alter bit of the user entered position

hammingCode[errorPosition - 1] = (hammingCode[errorPosition - 1] + 1) % 2;

}

// print sent data to the receiver

System.out.println("Sent Data is:");

for(int k = 0; k < hammingCodeSize; k++) {

System.out.print(hammingCode[hammingCodeSize - k - 1]);

}

System.out.println(); // for next line

receiveData(hammingCode, hammingCodeSize - arr.length);

}

// create getHammingCode() method that returns the hamming code for the data which we want to send

static int[] getHammingCode(int data[]) {

// declare an array that will store the hamming code for the data

int returnData[];

int size;

// code to get the required number of parity bits

int i = 0, parityBits = 0 ,j = 0, k = 0;

size = data.length;

while(i < size) {

// 2 power of parity bits must equal to the current position(number of bits traversed + number of parity bits + 1).

if(Math.pow(2, parityBits) == (i + parityBits + 1)) {

parityBits++;

}

else {

i++;

}

}

// the size of the returnData is equal to the size of the original data + the number of parity bits.

returnData = new int[size + parityBits];

// for indicating an unset value in parity bit location, we initialize returnData array with '2'

for(i = 1; i <= returnData.length; i++) {

// condition to find parity bit location

if(Math.pow(2, j) == i) {

returnData[(i - 1)] = 2;

j++;

}

else {

returnData[(k + j)] = data[k++];

}

}

// use for loop to set even parity bits at parity bit locations

for(i = 0; i < parityBits; i++) {

returnData[((int) Math.pow(2, i)) - 1] = getParityBit(returnData, i);

}

return returnData;

}

// create getParityBit() method that return parity bit based on the power

static int getParityBit(int returnData[], int pow) {

int parityBit = 0;

int size = returnData.length;

for(int i = 0; i < size; i++) {

// check whether returnData[i] contains an unset value or not

if(returnData[i] != 2) {

// if not, we save the index in k by increasing 1 in its value

int k = (i + 1);

// convert the value of k into binary

String str = Integer.toBinaryString(k);

//Now, if the bit at the 2^(power) location of the binary value of index is 1,

// we check the value stored at that location. If the value is 1 or 0,

// we will calculate the parity value.

int temp = ((Integer.parseInt(str)) / ((int) Math.pow(10, pow))) % 10;

if(temp == 1) {

if(returnData[i] == 1) {

parityBit = (parityBit + 1) % 2;

}

}

}

}

return parityBit;

}

// create receiveData() method to detect error in the received data

static void receiveData(int data[], int parityBits) {

// declare variable pow, which we use to get the correct bits to check for parity.

int pow;

int size = data.length;

// declare parityArray to store the value of parity check

int parityArray[] = new int[parityBits];

// we use errorLoc string for storing the integer value of the error location.

String errorLoc = new String();

// use for loop to check the parities

for(pow = 0; pow < parityBits; pow++) {

// use for loop to extract the bit from 2^(power)

for(int i = 0; i < size; i++) {

int j = i + 1;

// convert the value of j into binary

String str = Integer.toBinaryString(j);

// find bit by using str

int bit = ((Integer.parseInt(str)) / ((int) Math.pow(10, pow))) % 10;

if(bit == 1) {

if(data[i] == 1) {

parityArray[pow] = (parityArray[pow] + 1) % 2;

}

}

}

errorLoc = parityArray[pow] + errorLoc;

}

// This gives us the parity check equation values.

// Using these values, we will now check if there is a single bit error and then correct it.

// errorLoc provides parity check eq. values which we use to check whether a single bit error is there or not

// if present, we correct it

int finalLoc = Integer.parseInt(errorLoc, 2);

// check whether the finalLoc value is 0 or not

if(finalLoc != 0) {

System.out.println("Error is found at location " + finalLoc + ".");

data[finalLoc - 1] = (data[finalLoc - 1] + 1) % 2;

System.out.println("After correcting the error, the code is:");

for(int i = 0; i < size; i++) {

System.out.print(data[size - i - 1]);

}

System.out.println();

}

else {

System.out.println("There is no error in the received data.");

}

// print the original data

System.out.println("The data sent from the sender:");

pow = parityBits - 1;

for(int k = size; k > 0; k--) {

if(Math.pow(2, pow) != k) {

System.out.print(data[k - 1]);

}

else {

// decrement value of pow

pow--;

}

}

System.out.println(); // for next line

}

}

5. Implementation of universal steganalysis

**LSBDecode.java**

import javax.imageio.ImageIO;

public class LSB\_decode {

static final String STEGIMAGEFILE = "C:\\steg.png";

static final String DECODEDMESSAGEFILE = "C:\\message\_dec.txt";

public static String b\_msg="";

public static int len = 0;

public static void main(String[] args) throws Exception {

BufferedImage yImage=readImageFile(STEGIMAGEFILE);

DecodeTheMessage(yImage);

String msg="";

//System.out.println("len is "+len\*8);

for(int i=0;i<len\*8;i=i+8){

String sub=b\_msg.substring(i,i+8);

int m=Integer.parseInt(sub,2);

char ch=(char) m;

System.out.println("m "+m+" c "+ch);

msg+=ch;

}

PrintWriter out = new PrintWriter(new FileWriter(DECODEDMESSAGEFILE, true), true);

out.write(msg);

out.close();

}

public static BufferedImage readImageFile(String COVERIMAGEFILE){

BufferedImage theImage = null;

File p = new File (COVERIMAGEFILE);

try{

theImage = ImageIO.read(p);

}catch (IOException e){

e.printStackTrace();

System.exit(1);

}

return theImage;

}

public static void DecodeTheMessage (BufferedImage yImage) throws Exception{

int j=0;

int currentBitEntry=0;

String bx\_msg="";

for (int x = 0; x < yImage.getWidth(); x++){

for ( int y = 0; y < yImage.getHeight(); y++){

if(x==0&&y<8){

//System.out.println("enc "+yImage.getRGB(x, y)+" dec "+yImage.getRGB(x, y)+" "+b\_msg);

int currentPixel = yImage.getRGB(x, y);

int red = currentPixel>>16;

red = red & 255;

int green = currentPixel>>8;

green = green & 255;

int blue = currentPixel;

blue = blue & 255;

String x\_s=Integer.toBinaryString(blue);

bx\_msg+=x\_s.charAt(x\_s.length()-1);

len=Integer.parseInt(bx\_msg,2);

}

else if(currentBitEntry<len\*8){

//System.out.println("enc "+yImage.getRGB(x, y)+" dec "+yImage.getRGB(x, y)+" "+b\_msg);

int currentPixel = yImage.getRGB(x, y);

int red = currentPixel>>16;

red = red & 255;

int green = currentPixel>>8;

green = green & 255;

int blue = currentPixel;

blue = blue & 255;

String x\_s=Integer.toBinaryString(blue);

b\_msg+=x\_s.charAt(x\_s.length()-1);

currentBitEntry++;

//System.out.println("curre "+currentBitEntry);

}

}

}

System.out.println("bin value of msg hided in img is "+b\_msg);

}

}

**LSBEncode.java**

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

//import java.util.Arrays;

import java.util.Scanner;

import javax.imageio.ImageIO;

public class LSB\_encode {

static final String MESSAGEFILE = "C:\\message.txt";

static final String COVERIMAGEFILE = "C:\\cover.jpeg";

static final String STEGIMAGEFILE = "C:\\steg.png";

public static void main(String[] args) throws Exception {

String contentOfMessageFile = (readMessageFile());

int[] bits=bit\_Msg(contentOfMessageFile);

System.out.println("msg in file "+contentOfMessageFile);

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

System.out.print(bits[i]);

System.out.println();

BufferedImage theImage=readImageFile(COVERIMAGEFILE);

hideTheMessage(bits, theImage);

}

public static String readMessageFile () throws FileNotFoundException{

String contentOfMessageFile = "";

File a = new File (MESSAGEFILE);

Scanner scan = new Scanner (a);

while (scan.hasNextLine()){

String next = scan.nextLine();

contentOfMessageFile += next;

if (scan.hasNextLine()){

contentOfMessageFile += "\n";

}

}

scan.close();

return contentOfMessageFile;

}

public static int[] bit\_Msg(String msg){

int j=0;

int[] b\_msg=new int[msg.length()\*8];

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

int x=msg.charAt(i);

String x\_s=Integer.toBinaryString(x);

while(x\_s.length()!=8){

x\_s='0'+x\_s;

}

System.out.println("dec value for "+x +" is "+x\_s);

for(int i1=0;i1<8;i1++) {

b\_msg[j] = Integer.parseInt(String.valueOf(x\_s.charAt(i1)));

j++;

};

}

return b\_msg;

}

public static BufferedImage readImageFile(String COVERIMAGEFILE){

BufferedImage theImage = null;

File p = new File (COVERIMAGEFILE);

try{

theImage = ImageIO.read(p);

}catch (IOException e){

e.printStackTrace();

System.exit(1);

}

return theImage;

}

public static void hideTheMessage (int[] bits, BufferedImage theImage) throws Exception{

File f = new File (STEGIMAGEFILE);

BufferedImage sten\_img=null;

int bit\_l=bits.length/8;

int[] bl\_msg=new int[8];

System.out.println("bit lent "+bit\_l);

String bl\_s=Integer.toBinaryString(bit\_l);

while(bl\_s.length()!=8){

bl\_s='0'+bl\_s;

}

for(int i1=0;i1<8;i1++) {

bl\_msg[i1] = Integer.parseInt(String.valueOf(bl\_s.charAt(i1)));

};

int j=0;

int b=0;

int currentBitEntry=8;

for (int x = 0; x < theImage.getWidth(); x++){

for ( int y = 0; y < theImage.getHeight(); y++){

if(x==0&&y<8){

int currentPixel = theImage.getRGB(x, y);

int ori=currentPixel;

int red = currentPixel>>16;

red = red & 255;

int green = currentPixel>>8;

green = green & 255;

int blue = currentPixel;

blue = blue & 255;

String x\_s=Integer.toBinaryString(blue);

String sten\_s=x\_s.substring(0, x\_s.length()-1);

sten\_s=sten\_s+Integer.toString(bl\_msg[b]);

//j++;

int temp=Integer.parseInt(sten\_s,2);

int s\_pixel=Integer.parseInt(sten\_s, 2);

int a=255;

int rgb = (a<<24) | (red<<16) | (green<<8) | s\_pixel;

theImage.setRGB(x, y, rgb);

//System.out.println("original "+ori+" after "+theImage.getRGB(x, y));

ImageIO.write(theImage, "png", f);

b++;

}

else if (currentBitEntry < bits.length+8 ){

int currentPixel = theImage.getRGB(x, y);

int ori=currentPixel;

int red = currentPixel>>16;

red = red & 255;

int green = currentPixel>>8;

green = green & 255;

int blue = currentPixel;

blue = blue & 255;

String x\_s=Integer.toBinaryString(blue);

String sten\_s=x\_s.substring(0, x\_s.length()-1);

sten\_s=sten\_s+Integer.toString(bits[j]);

j++;

int temp=Integer.parseInt(sten\_s,2);

int s\_pixel=Integer.parseInt(sten\_s, 2);

int a=255;

int rgb = (a<<24) | (red<<16) | (green<<8) | s\_pixel;

theImage.setRGB(x, y, rgb);

//System.out.println("original "+ori+" after "+theImage.getRGB(x, y));

ImageIO.write(theImage, "png", f);

currentBitEntry++;

//System.out.println("curre "+currentBitEntry);

}

}

}

}

}

6. Experiment on target steganalysis

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

public class Steganalysis {

public static void main(String[] args) throws IOException {

String imagePath = "image.jpg";

File imageFile = new File(imagePath);

BufferedImage image = ImageIO.read(imageFile);

int width = image.getWidth();

int height = image.getHeight();

int numPixels = width \* height;

int numHiddenPixels = 0;

for (int y = 0; y < height; y++) {

for (int x = 0; x < width; x++) {

int pixel = image.getRGB(x, y);

int alpha = (pixel >> 24) & 0xff;

int red = (pixel >> 16) & 0xff;

int green = (pixel >> 8) & 0xff;

int blue = pixel & 0xff;

if (alpha < 255) {

numHiddenPixels++;

}

if (red % 2 == 1 || green % 2 == 1 || blue % 2 == 1) {

System.out.println("Hidden data detected!");

return;

}

}

}

System.out.println("No hidden data detected.");

System.out.println("Percentage of image pixels with hidden data: " + (numHiddenPixels \* 100.0 / numPixels) + "%");

}

}

7. Experiment on data hiding in different image types

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

public class ImageSteganography {

public static void main(String[] args) throws IOException {

String originalImagePath = "original.png";

String imagePathWithHiddenData = "hidden.png";

String secretMessage = "This is a secret message!";

// Load the original image

BufferedImage originalImage = ImageIO.read(new File(originalImagePath));

// Embed the secret message in the image using LSB technique

int messageLength = secretMessage.length();

int messageIndex = 0;

for (int y = 0; y < originalImage.getHeight(); y++) {

for (int x = 0; x < originalImage.getWidth(); x++) {

int pixel = originalImage.getRGB(x, y);

if (messageIndex < messageLength) {

char messageChar = secretMessage.charAt(messageIndex);

int messageBit = (messageChar >> (7 - messageIndex % 8)) & 1;

pixel = (pixel & 0xFEFFFFFF) | (messageBit << 24);

messageIndex++;

}

originalImage.setRGB(x, y, pixel);

}

}

// Save the image with hidden data

File outputImageFile = new File(imagePathWithHiddenData);

ImageIO.write(originalImage, "png", outputImageFile);

}

}

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import org.jsteg.\*;

public class SteganographyExperiment {

public static void main(String[] args) throws IOException, JStegException {

// Load the image

BufferedImage image = ImageIO.read(new File("image.jpg"));

// Encode a message within the image

byte[] message = "Hello, world!".getBytes();

JSteg.encode(image, message);

// Save the encoded image to a file

File output = new File("image-encoded.jpg");

ImageIO.write(image, "jpg", output);

// Decode the message from the encoded image

BufferedImage encodedImage = ImageIO.read(output);

byte[] decodedMessage = JSteg.decode(encodedImage);

// Print the decoded message to the console

System.out.println(new String(decodedMessage));

}

}

8. Implementation of statistical steganalysis

import java.awt.image.BufferedImage;

import java.io.File;

import javax.imageio.ImageIO;

Class StatDemo

{

public static void main(String[] args) throws Exception

{

File coverDir = new File("path/to/cover/images/");

File stegoDir = new File("path/to/stego/images/");

double coverMean = 0.0;

double stegoMean = 0.0;

for (File file : coverDir.listFiles()) {

BufferedImage image = ImageIO.read(file);

double mean = calculateMean(image);

coverMean += mean;

}

for (File file : stegoDir.listFiles()) {

BufferedImage image = ImageIO.read(file);

double mean = calculateMean(image);

stegoMean += mean;

}

coverMean /= coverDir.listFiles().length;

stegoMean /= stegoDir.listFiles().length;

System.out.println("Cover mean: " + coverMean);

System.out.println("Stego mean: " + stegoMean);

}

}

public static double calculateMean(BufferedImage image)

{

int width = image.getWidth();

int height = image.getHeight();

double sum = 0.0;

for (int y = 0; y < height; y++) {

for (int x = 0; x < width; x++) {

int pixel = image.getRGB(x, y);

int red = (pixel >> 16) & 0xff;

int green = (pixel >> 8) & 0xff;

int blue = (pixel) & 0xff;

double gray = 0.299 \* red + 0.587 \* green + 0.114 \* blue;

sum += gray;

}

}

double mean = sum / (width \* height);

return mean;

}

9. Implementation of reversible data hiding

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

public class MeanSteganalysis {

public static void main(String[] args) {

String coverPath = "path/to/cover/image.jpg";

String stegoPath = "path/to/stego/image.jpg";

BufferedImage coverImage = loadImage(coverPath);

BufferedImage stegoImage = loadImage(stegoPath);

double coverMean = calculateMean(coverImage);

double stegoMean = calculateMean(stegoImage);

double threshold = (coverMean + stegoMean) / 2;

System.out.println("Cover image mean: " + coverMean);

System.out.println("Stego image mean: " + stegoMean);

System.out.println("Threshold: " + threshold);

if (stegoMean > threshold) {

System.out.println("Stego image detected.");

} else {

System.out.println("No stego image detected.");

}

}

private static BufferedImage loadImage(String path) {

try {

return ImageIO.read(new File(path));

} catch (IOException e) {

e.printStackTrace();

return null;

}

}

private static double calculateMean(BufferedImage image) {

int width = image.getWidth();

int height = image.getHeight();

int totalPixels = width \* height;

long redSum = 0;

long greenSum = 0;

long blueSum = 0;

for (int y = 0; y < height; y++) {

for (int x = 0; x < width; x++) {

int pixel = image.getRGB(x, y);

redSum += (pixel >> 16) & 0xff;

greenSum += (pixel >> 8) & 0xff;

blueSum += pixel & 0xff;

}

}

double redMean = (double) redSum / totalPixels;

double greenMean = (double) greenSum / totalPixels;

double blueMean = (double) blueSum / totalPixels;

return (redMean + greenMean + blueMean) / 3;

}

}

10. Implementation of Steganography in transform domain and Steganography in

Encrypted images