Description::

1.Spliting the data packets.

2.randomly indexing the data packets either rsa or aes .

3.According to the randomly indexed algorithm for respective data packet,it will encrypt and decrypt the packet.

for eg:

packet1: rsa

\*encrypt the data using rsa agorithm.and save the encrypted file.

\*decrypt the data using rsa algorithm and save the decrypted file.

packet2: aes

\*encrypt the data using aes agorithm.and save the encrypted file.

\*decrypt the data using aes algorithm and save the decrypted file.

and the process goes on...

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

source code::

import java.io.\*;

import java.util.\*;

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;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Path;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.security.SecureRandom;

import java.util.Base64;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.spec.SecretKeySpec;

public class split\_rsa\_aes{

public static void rsa\_conversion(String s) throws Exception {

String z;

int y=1;

Path filePath = Paths.get(s);

String plainText = Files.readString(filePath);

// 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);

if (y <= 9) {

z = s + ".00" + y;

} else {

z = s + ".00" + y;

}

File cipher\_text = new File(z+"cipher.txt");

FileWriter myWriter = new FileWriter(z+"cipher.txt");

myWriter.write(encryptedText);

myWriter.write("\n");

System.out.println("Successfully wrote to the cipher text file.");

myWriter.close();

// Decryption

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

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

File plain\_text = new File(z+"plaintext.txt");

FileWriter Writter = new FileWriter(z+"plaintext.txt");

Writter.write(decryptedText);

Writter.write("\n");

System.out.println("Successfully wrote to the"+z+"plaintext.txt");

Writter.close();

}

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);

}

public static void aes\_conversion(String s) throws Exception {

String j;

int k=1;

Path filePath = Paths.get(s);

String plainText = Files.readString(filePath);

KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");

keyGenerator.init(256);

// Generate Key

SecretKey key = keyGenerator.generateKey();

// Generating IV.

byte[] IV = new byte[16];

SecureRandom random = new SecureRandom();

random.nextBytes(IV);

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

byte[] cipherText = aes\_encrypt(plainText.getBytes(),key, IV);

System.out.println("Encrypted Text : "+Base64.getEncoder().encodeToString(cipherText));

String encrypt\_msg = Base64.getEncoder().encodeToString(cipherText);

if (k <= 9) {

j = s + ".00" + k;

} else {

j = s + ".00" + k;

}

File aes\_cipher\_text = new File(j+"aes\_cipher.txt");

FileWriter myWriter\_aes = new FileWriter(j+"aes\_cipher.txt");

myWriter\_aes.write(encrypt\_msg);

myWriter\_aes.write("\n");

System.out.println("Successfully wrote to the aes\_cipher text file.");

myWriter\_aes.close();

String decrypt\_msg = aes\_decrypt(cipherText,key, IV);

System.out.println("DeCrypted Text : "+decrypt\_msg);

File aes\_plain\_text = new File(j+"aes\_plain.txt");

FileWriter Writer\_aes = new FileWriter(j+"aes\_plain.txt");

Writer\_aes.write(decrypt\_msg);

Writer\_aes.write("\n");

System.out.println("Successfully wrote to the aes\_plain text file.");

Writer\_aes.close();

}

public static byte[] aes\_encrypt (byte[] plaintext,SecretKey key,byte[] IV ) throws Exception

{

//Get Cipher Instance

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

//Create SecretKeySpec

SecretKeySpec keySpec = new SecretKeySpec(key.getEncoded(), "AES");

//Create IvParameterSpec

IvParameterSpec ivSpec = new IvParameterSpec(IV);

//Initialize Cipher for ENCRYPT\_MODE

cipher.init(Cipher.ENCRYPT\_MODE, keySpec, ivSpec);

//Perform Encryption

byte[] cipherText = cipher.doFinal(plaintext);

return cipherText;

}

public static String aes\_decrypt (byte[] cipherText, SecretKey key,byte[] IV) throws Exception

{

//Get Cipher Instance

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

//Create SecretKeySpec

SecretKeySpec keySpec = new SecretKeySpec(key.getEncoded(), "AES");

//Create IvParameterSpec

IvParameterSpec ivSpec = new IvParameterSpec(IV);

//Initialize Cipher for DECRYPT\_MODE

cipher.init(Cipher.DECRYPT\_MODE, keySpec, ivSpec);

//Perform Decryption

byte[] decryptedText = cipher.doFinal(cipherText);

return new String(decryptedText);

}

public static void main(String []args){

try {

HashMap<String, Integer> map = new HashMap<>();

byte b[] = new byte[10];

int x = 1, j = 0;

String s = "";

InputStreamReader ins = new InputStreamReader(System.in);

BufferedReader br = new BufferedReader(ins);

System.out.println("enter the path of source file: ");

String path = br.readLine();

FileInputStream fis = new FileInputStream(path);

int read\_bytes;

while (fis.available() != 0) {

j = 0;

s = "";

if (x <= 9) {

s = path + ".00" + x;

} else {

s = path + ".00" + x;

}

FileOutputStream fos = new FileOutputStream(s);

while (j <= 50 && fis.available() != 0) {

read\_bytes = fis.read(b, 0, 10);

j = j + read\_bytes;

fos.write(b, 0, read\_bytes);

}

System.out.println("packet " + x + " created");

x++;

int min = 5;

int max = 2000;

int random\_int = (int) Math.floor(Math.random() \* (max - min + 1) + min);

if (random\_int >= 1000) {

map.put(s, 2);

rsa\_conversion(s);

} else {

map.put(s, 1);

aes\_conversion(s);

}

}

System.out.println("File splitted successflly!!!!");

Iterator hmIterator = map.entrySet().iterator();

File indexTable = new File("scheudler.txt");

System.out.println("Scheudler file has been created ");

FileWriter myWriter = new FileWriter("scheudler.txt");

while (hmIterator.hasNext()) {

Map.Entry mapElement = (Map.Entry) hmIterator.next();

System.out.println(mapElement.getKey() + " : " + mapElement.getValue());

myWriter.write(mapElement.getKey() + " : " + mapElement.getValue());

myWriter.write("\n");

}

System.out.println("Successfully wrote to the Scheudler file.");

myWriter.close();

}

catch(Exception e){

e.printStackTrace();

}

}

}