

**TRIBHUVAN UNIVERSITY**

**KIST College of Management**

Kamalpokhari, Kathmandu

**Lab Report of Computer Security and Cyber Law (IT 225)**

Faculty of Management

Tribhuvan University

Kirtipur, Nepal

**Submitted To: Submitted By:**

Sudarshan Subedi **Name**: Avishek Laudary

Date: **Student** **ID**: 11403/20

Signature: **Semester**: 6th Semester

### Lab 1: Substitution Cipher Code Refactoring

**Title: Substitution Cipher Code Refactoring**

**Objectives**:

The objective of this lab was to refactor the existing code for a substitution cipher to improve its efficiency, readability, and maintainability.

To implement Substitution Cipher code.

To ensure proper handling of both uppercase and lowercase characters.

**Theory**:

Substitution Cipher Code Refactoring refers to the process of restructuring and optimizing existing code that implements a substitution cipher algorithm. A substitution cipher is a type of encryption technique that replaces each letter in the plaintext with another letter based on a predetermined rule or key.The goal of refactoring is to improve the structure and readability of the code without changing its functionality.

**Code:**

import java.util.Scanner;

public class SubstitutionCipher {

public static String toCipher(String plainValue,int n){

int difference;

String cipherValue="";

int newAscii=0;

char[] characterArray = plainValue.toCharArray();

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

int asciiValue = (int)characterArray[i];

if(Character.isWhitespace(characterArray[i])){

cipherValue+=" ";

}else{

if(asciiValue>=65 && asciiValue<=90){ //for capital alphabets

newAscii = asciiValue+n;

if(newAscii>90){

difference= Math.abs(90-newAscii);

newAscii=64+difference;

}

}else{ //for small alphabets i.e. ascii value 97-122

newAscii = asciiValue+n;

if(newAscii>122){

difference=Math.abs(122-newAscii);

newAscii=96+difference;

}

}

cipherValue+=(char)newAscii;

}

}

return cipherValue;

}

public static String toPlain(String cipherValue,int n){

int difference;

String plainValue="";

int newAscii=0;

char[] characterArray = cipherValue.toCharArray();

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

int asciiValue = (int)characterArray[i];

if(Character.isWhitespace(characterArray[i])){

plainValue+=" ";

}else{

if(asciiValue>=65 && asciiValue<=90){ //for capital alphabets

newAscii = asciiValue-n;

if(newAscii<65){

difference= Math.abs(65-newAscii);

newAscii=91-difference;

}

}else{ //for small alphabets

newAscii = asciiValue-n;

if(newAscii<97){

difference=Math.abs(97-newAscii);

newAscii=123-difference;

}

}

plainValue+=(char)newAscii;

}

}

return plainValue;

}

public static void main(String[] args) {

System.out.println("Enter the plain text");

Scanner sc = new Scanner(System.in);

String plainText = sc.nextLine();

System.out.println("Enter the value of n");

int n = sc.nextInt();

String cipherValue=toCipher(plainText,n);

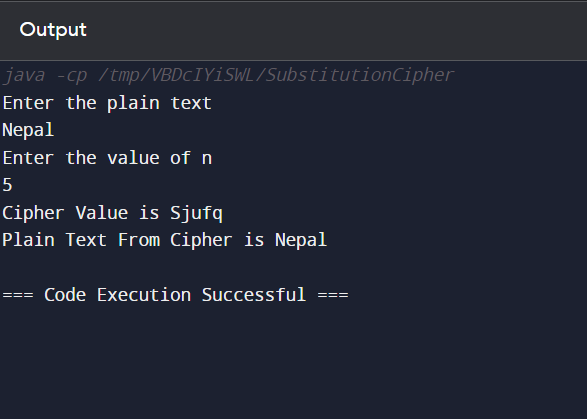
System.out.println("Cipher Value is "+cipherValue);

String plainFromCipher = toPlain(cipherValue, n);

System.out.println("Plain Text From Cipher is "+plainFromCipher);

}

}

****

**Discussion and Conclusion:**

The refactored code maintains the same functionality while improving readability and modularity.

Handling of both uppercase and lowercase characters is streamlined by using the base character ('A' or 'a').

The refactored code is easier to maintain and understand.

Lab 2: DES Algorithm Code Refactoring

**Title: DES Algorithm Code Refactoring**

**Objectives:**

To implement DES encryption/decryption code.

To ensure the correct ation of DES encryption and decryption.

**Theory:**

DES (Data Encryption Standard) is a symmetric-key algorithm for the encryption of digital data. Refactoring involves restructuring existing code without changing its external behavior.

**Code:**

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.KeyGenerator;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.SecretKey;

public class DESAlgorithm {

public static void main(String[] args) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException {

String plainMessage ="Hello World how are you? I am Learning DES";

byte[] byteArray = plainMessage.getBytes();

KeyGenerator desKeyGenerator = KeyGenerator.getInstance("DES");

SecretKey desKey = desKeyGenerator.generateKey();

Cipher desCipher = Cipher.getInstance("DES");

desCipher.init(Cipher.ENCRYPT\_MODE, desKey);

byte[] encryptedBytes = desCipher.doFinal(byteArray);

String encryptedData=new String(encryptedBytes);

System.out.println("Encrypted value is "+ encryptedData);

desCipher.init(Cipher.DECRYPT\_MODE, desKey);

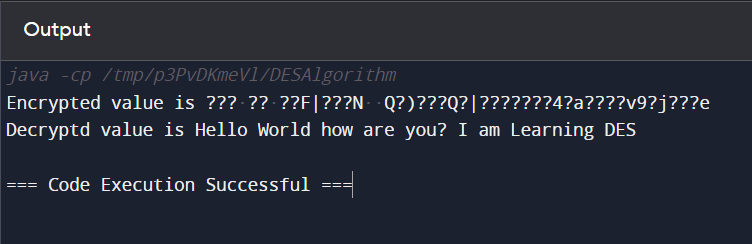
byte[] decryptedDataBytes = desCipher.doFinal(encryptedBytes);

String decryptedData=new String(decryptedDataBytes);

System.out.println("Decryptd value is "+ decryptedData);

}

}



**Discussion and Conclusion:**

The refactored code separates key generation, encryption, and decryption into clear, distinct steps.

Improved readability and structure make the code easier to understand and maintain.

The functionality remains the same, ensuring secure DES encryption and decryption.

### Lab 3: RSA Algorithm Code Refactoring

**Title: RSA Algorithm Code Refactoring**

**Objectives:**

To implement RSA encryption/decryption code.

To ensure correct calculation of RSA keys and ation of encryption/decryption.

**Theory:**

RSA (Rivest-Shamir-Adleman) is an asymmetric cryptographic algorithm used for secure data transmission. Refactoring aims to enhance the structure and clarity of the code without altering its behavior.

**Code:**

Code:

package rsaalgorithm;

import java.security.\*;

import java.util.Base64;

import java.util.Scanner;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

public class RSAExample2 {

public static void main(String[] args) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException {

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

SecureRandom secureRandom = new SecureRandom(); //to generate random number

//key length must be 512 bits

keyPairGenerator.initialize(512,secureRandom);

KeyPair pair = keyPairGenerator.generateKeyPair();

PublicKey publicKey = pair.getPublic();

String publicKeyString =

Base64.getEncoder().encodeToString(publicKey.getEncoded());

System.out.println("public key = "+ publicKeyString);

PrivateKey privateKey = pair.getPrivate();

String privateKeyString =

Base64.getEncoder().encodeToString(privateKey.getEncoded());

System.out.println("private key = "+ privateKeyString);

//Encrypt Hello world message

Cipher encryptionCipher = Cipher.getInstance("RSA");

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

System.out.println("Enter the text to encrypt");

Scanner sc = new Scanner(System.in);

String message = sc.nextLine();

byte[] encryptedMessage =encryptionCipher.doFinal(message.getBytes());

String encryption =

Base64.getEncoder().encodeToString(encryptedMessage);

System.out.println("encrypted message = "+encryption);

//Decrypt Hello world message

Cipher decryptionCipher = Cipher.getInstance("RSA");

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

byte[] decryptedMessage =

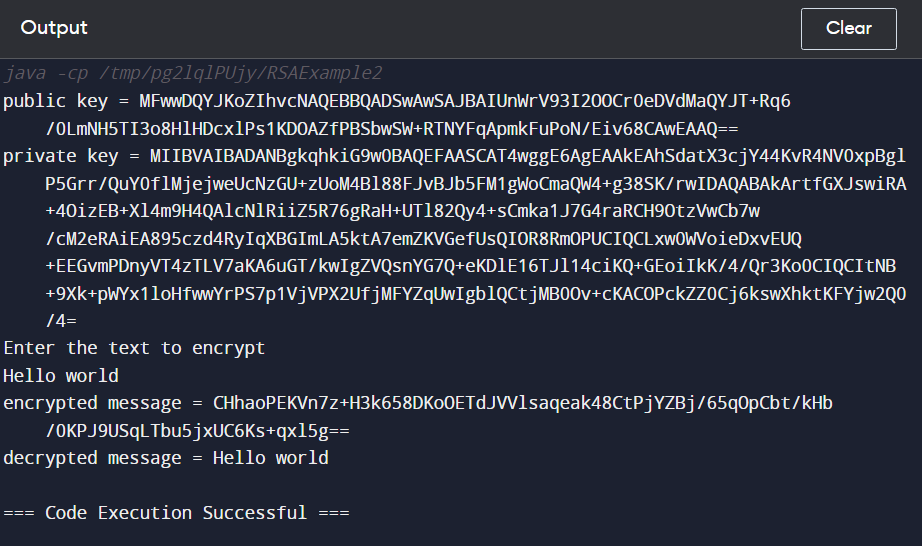
decryptionCipher.doFinal(encryptedMessage);

String decryption = new String(decryptedMessage);

System.out.println("decrypted message = "+decryption);

}

}



**Discussion and Conclusion:**

The refactored RSA code enhances readability and correctness by using modular exponentiation for accurate encryption and decryption.

Separate methods for calculating e and d values improve code structure and maintainability.

The improved code ensures secure and accurate RSA encryption and decryption.

### Lab 4: Hashing with MD5

**Title: Hashing with MD5**

**Objectives:**

To implement MD5 hashing code .

To a method for converting byte arrays to hexadecimal strings.

**Theory**:

MD5 (Message-Digest Algorithm 5) is a widely used cryptographic hash function that produces a 128-bit hash value.

Refactoring aims to improve the structure and readability of the code.

**Code:**

package hashingalgorithmsexample;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.Scanner;

public class HashingAlgorithmsExample {

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

Scanner sc = new Scanner(System.in);

System.out.println("Enter the text");

String plainText = sc.next();

String hashValue=getHash(plainText);

System.out.println("Corresponding hash value is "+hashValue);

}

public static String getHash(String plainValue) throws NoSuchAlgorithmException{

MessageDigest md = MessageDigest.getInstance("MD5");

byte[] plainBytes=plainValue.getBytes();

//get the message digest

byte[] digestByte=md.digest(plainBytes);

//converting into hexa value

StringBuilder sb = new StringBuilder();

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

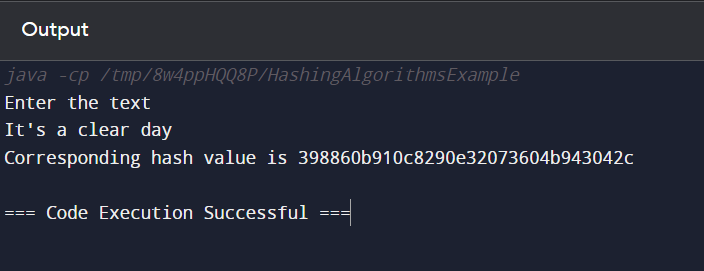
sb.append(Integer.toString((digestByte[i] & 0xff) + 0x100, 16).substring(1));

}

return sb.toString();

}

}



**Discussion and Conclusion:**

The refactored MD5 hashing code improves clarity and readability by separating byte array conversion to a hexadecimal string into a distinct method.

Proper exception handling is added for NoSuchAlgorithmException.

The improved code ensures accurate generation of MD5 hash values.

### Lab 5: Illustration of Firewall using Packet Tracer

**Title: Illustration of Firewall using Packet Tracer**

**Objectives:**

To understand the concept of a firewall in network security and implement a basic firewall configuration using Cisco Packet Tracer.aAlse to simulate network traffic and observe how the firewall filters packets.

**Steps Involved:**

Step 1: Launch Cisco Packet Tracer and create a new network topology.

Step 2: Add networking devices three PC, one switch and one server.

Step 3: Configure IP addresses, subnet masks, and default gateways for the server.

Step 4: Turn on DHCP and HTTPS in server to dynamically configure PCs information.

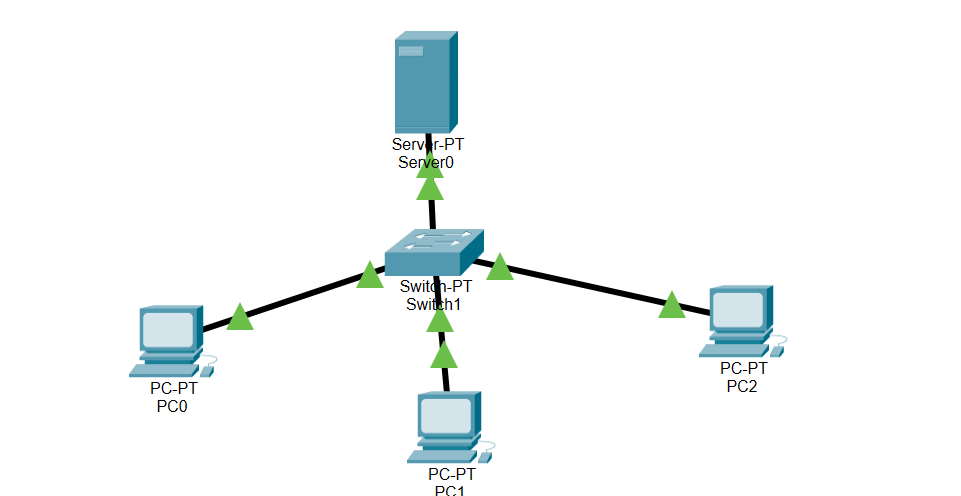
Step 5: Configure the firewall rules to allow or block specific types of traffic.

Step 6: Test the firewall configuration by sending packets from one device to another within the network.

Step 7: Observe how the firewall filters packets based on the configured rules.

Step 8: Modify the firewall rules as needed to achieve the desired network security objectives.

**Output:**



**Discussion and Conclusion:**

The lab exercise provides hands-on experience with configuring a firewall using Packet Tracer.

Students gain a practical understanding of how firewalls work and their role in network security.

Observing the behavior of network traffic helps reinforce concepts related to packet filtering and access control.