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.

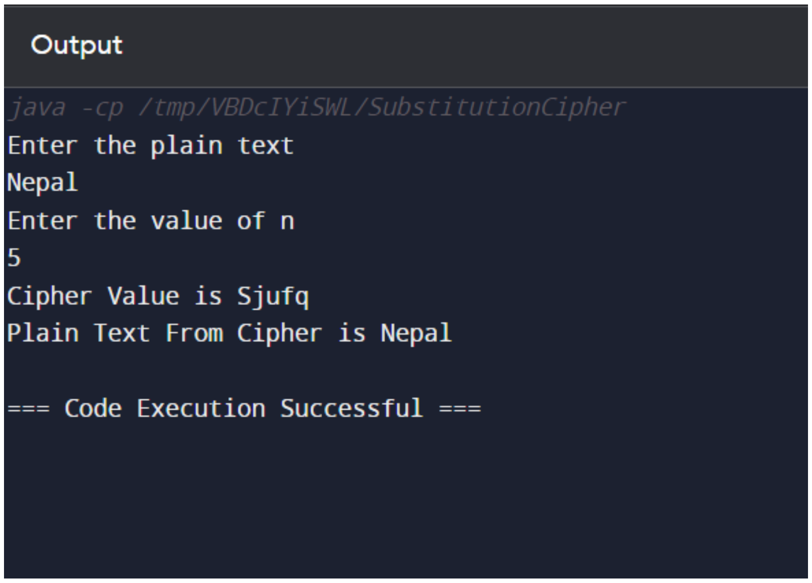
To ensure proper handling of both uppercase and lowercase characters.

Theory:

Substitution Cipher Code Refactoring involves the systematic process of reorganizing and enhancing existing code that implements a substitution cipher algorithm. A substitution cipher is an encryption technique where each letter in the plaintext is replaced with another letter according to a predetermined rule or key. The refactoring process aims to improve various aspects of the code, such as its structure, readability, and maintainability, without altering its fundamental behavior or functionality. This includes making the code more modular, optimizing performance, reducing complexity, and ensuring that it adheres to best coding practices. By doing so, the refactored code becomes easier to understand, test, and modify, which ultimately leads to more efficient and effective software development and maintenance.

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);
}

}
```



```
java -cp /tmp/VBDcIYiSWL/SubstitutionCipher
Enter the plain text
Nepal
Enter the value of n
5
Cipher Value is Sjufq
Plain Text From Cipher is Nepal

=== Code Execution Successful ===
```

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.

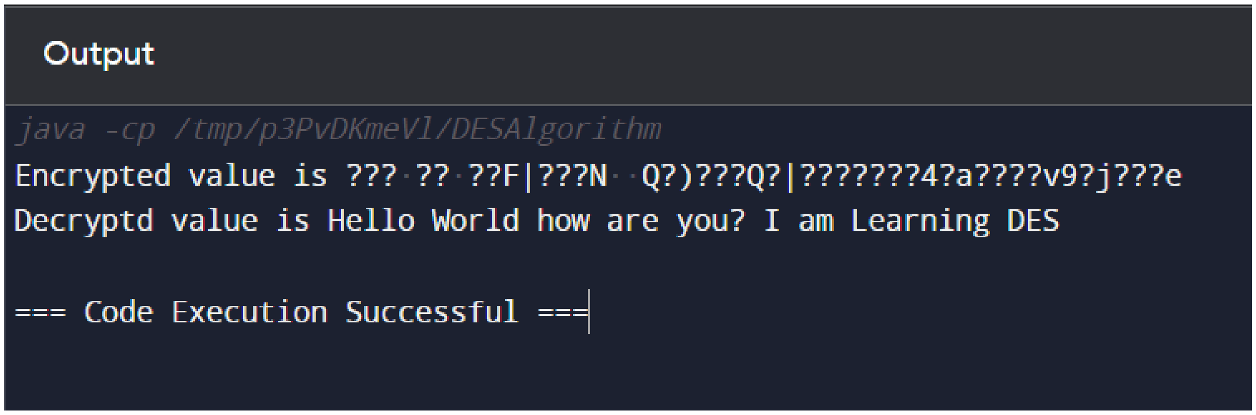
To ensure the correct ation of DES encryption and decryption.

Theory:

DES (Data Encryption Standard) is a symmetric-key algorithm designed for the encryption of digital data. As a symmetric-key algorithm, it uses the same key for both encryption and decryption processes, ensuring efficient and secure data transmission between parties who share the key. DES operates on 64-bit blocks of data and uses a 56-bit key. The algorithm performs 16 rounds of complex permutation and substitution transformations to convert plaintext into ciphertext. Despite its widespread use in the past, DES has been largely superseded by more secure algorithms like AES (Advanced Encryption Standard) due to vulnerabilities that emerged with advances in computational power.

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);
    }
}
```



Output

```
java -cp /tmp/p3PvDKmeVl/DESAlgorithm
Encrypted value is ???·??·??F|???N· ·Q?)???Q?|???????4?a????v9?j???e
Decryptd value is Hello World how are you? I am Learning DES

=== Code Execution Successful ===
```

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:

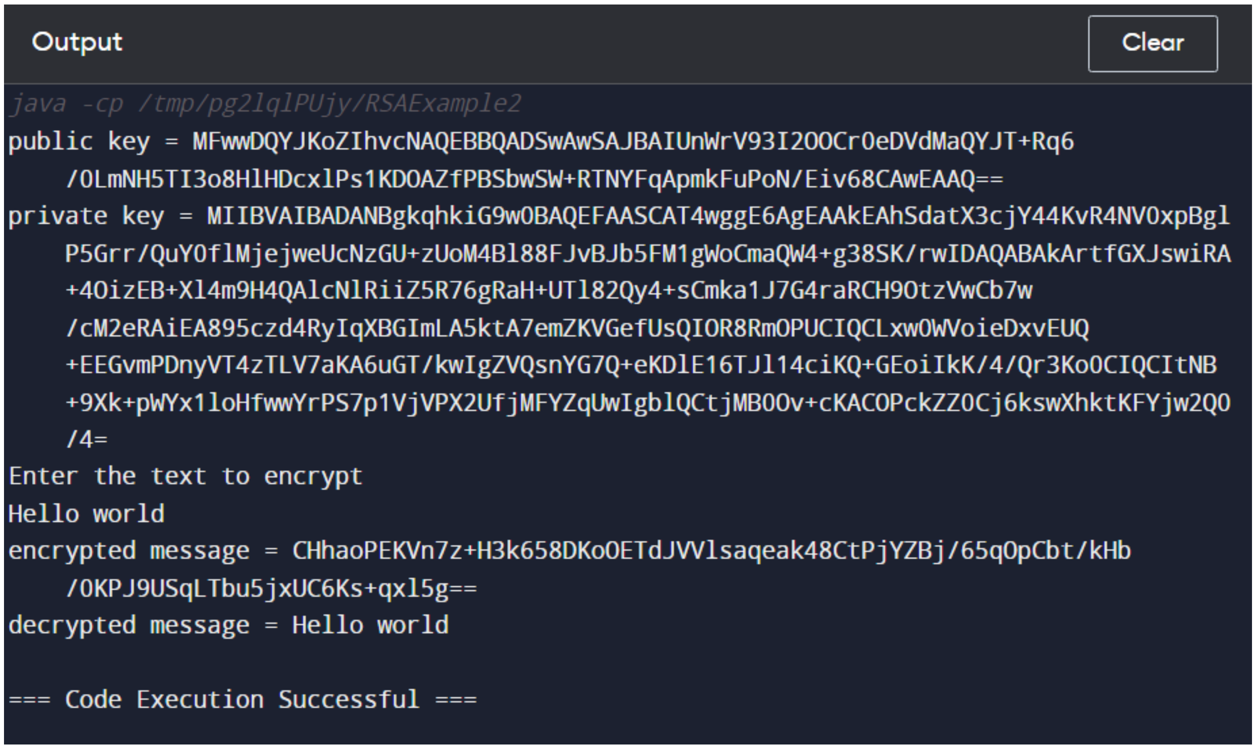
RSA, or Rivest-Shamir-Adleman, is an asymmetric cryptographic algorithm widely used for secure data transmission. It employs a pair of keys, a public key for encryption and a private key for decryption, ensuring that data can be securely exchanged over an insecure network. RSA is based on the mathematical difficulty of factoring large prime numbers, making it a robust choice for securing sensitive information in applications such as online banking, digital signatures, and secure email communication.

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);
    }
}
```



```
java -cp /tmp/pg2lqlPUjy/RSAExample2
public key = MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAIUnWrV93I2OOCr0eDVdMaQYJT+Rq6
    /0LmNH5TI3o8HlHDcxlPs1KD0AZfPBSbwSW+RTNYFqApmkFuPoN/Eiv68CAwEAAQ==
private key = MIIBVAIBADANBgkqhkiG9w0BAQEFAASCAT4wggE6AgEAAkEAhSdatX3cjY44KvR4NV0xpBgl
    P5Grr/QuY0flMjejweUcNzGU+zUoM4Bl88FJvBJb5FM1gWoCmaQW4+g38SK/rwIDAQABAkArtfGXJswiRA
    +4OizEB+Xl4m9H4QAlcNlRiiZ5R76gRaH+UTl82Qy4+sCmka1J7G4raRCH9OtzVwCb7w
    /cM2eRAiEA895czd4RyIqXBGImLA5ktA7emZKVGefUsQIOR8RmOPUCIQCLxw0WVoieDxvEUQ
    +EEGvmPDnyVT4zTLV7aKA6uGT/kwIgZVQsnYG7Q+eKDlE16TJl14ciKQ+GEoiIkK/4/Qr3Ko0CIQCItNB
    +9Xk+pWYx1loHfwwYrPS7p1VjVPX2UfjMFYZqUwIgblQCtjMB0Ov+cKACOPckZZ0Cj6kswXhktKFYjw2Q0
    /4=
Enter the text to encrypt
Hello world
encrypted message = CHhaoPEKVn7z+H3k658DKoOETdJVVlsaqeak48CtPjYZBj/65qOpCbt/kHb
    /0KPJ9USqLTbu5jxUC6Ks+qxl5g==
decrypted message = Hello world

=== Code Execution Successful ===
```

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)

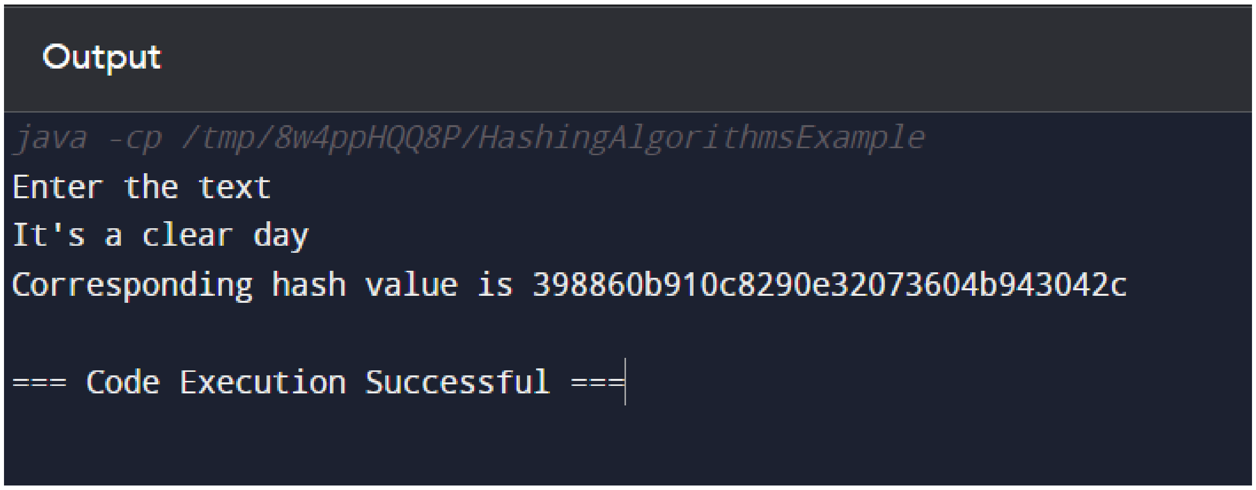
MD5, or Message-Digest Algorithm 5, is a widely used cryptographic hash function that produces a 128-bit hash value. Developed by Ronald Rivest in 1991, MD5 is primarily used to verify data integrity. It takes an input (or 'message') and converts it into a fixed-size, 32-character hexadecimal number. Despite its popularity, MD5 is considered broken for security purposes due to vulnerabilities that allow for collision attacks, where two different inputs produce the same hash value. Consequently, MD5 is now primarily used for checksums and non-security-critical applications.

Refactoring

Refactoring is the process of restructuring existing code without changing its external behavior. The goal is to improve the structure, readability, and maintainability of the code. By refactoring, developers make the code cleaner and easier to understand, which facilitates easier updates and reduces the likelihood of bugs. Common refactoring techniques include renaming variables to more meaningful names, extracting methods to simplify complex code, and removing redundant or duplicated code. This practice is essential for maintaining high-quality software that is both efficient and easy to work with.

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();
    }
}
```



```
java -cp /tmp/8w4ppHQQ8P/HashingAlgorithmsExample
Enter the text
It's a clear day
Corresponding hash value is 398860b910c8290e32073604b943042c

=== Code Execution Successful ===
```

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.

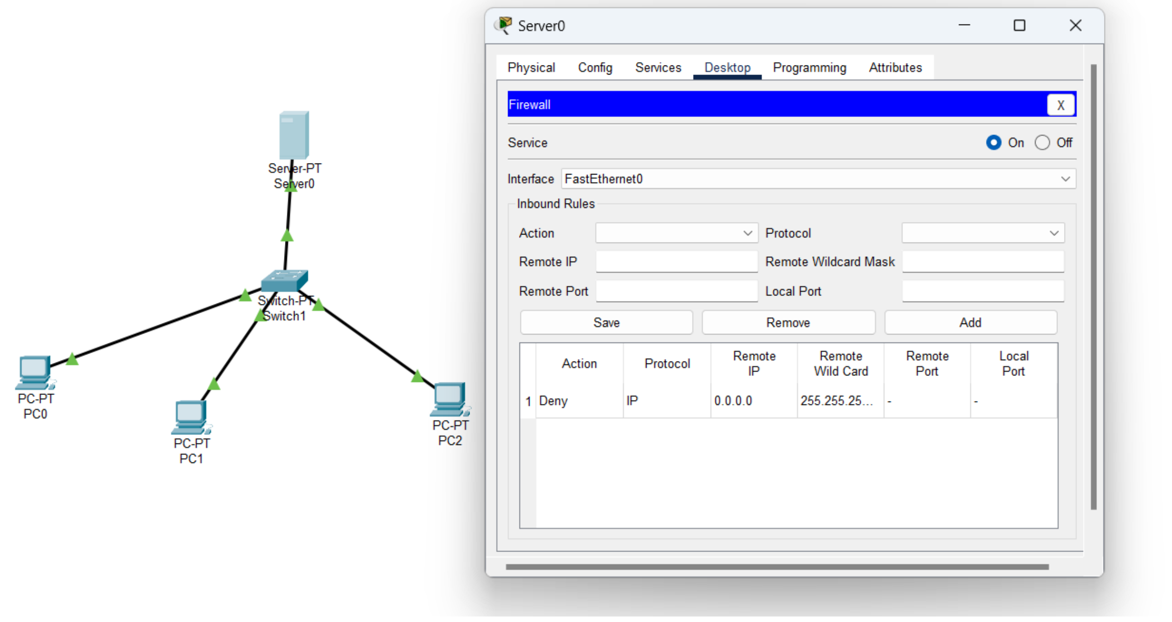
Step 5: Configure the firewall rules to allow or block specific types of traffic.

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:



Service [✓ On] Off

Interface FastEthernet0

Inbound Rules

	Action	Protocol	Remote IP	Remote Wild Card	Remote Port	Local Port
1	Deny	IP	0.0.0.0	255.255.25...	-	-

Discussion and Conclusion:

The lab exercise offers hands-on experience with configuring a firewall using Packet Tracer. Through this activity, students gain a practical understanding of the functionality and importance of firewalls in network security. By observing the behavior of network traffic, students reinforce their knowledge of packet filtering and access control, solidifying key concepts in network protection and management. This practical engagement helps bridge the gap between theoretical knowledge and real-world application, enhancing overall learning and comprehension in network security.