

Department of Computer Engineering

D-19

Lab Manual (2023-24)

Class: B TECH Computer Term: I & II

Computer System Security Lab –(UCOP401)

Faculty Name

Course Detail

UCOP401: Computer System Security Lab

Class: B.TECH
Internal Marks: 25
Credits : 1

Division: A &B
External marks:25
Pattern:2020

Name of Subject: Computer System Security Lab		Semester: VII & VIII
COURSE OUTCOME		
CO1	Identify the need of security and apply various security methods to protect the confidential data	
CO2	Categorize various encryption and decryption algorithms and apply the algorithms as per the requirement.	
CO3	Describe different hashing algorithms and implement the algorithms in order to securing and efficiently transfer the data.	
CO4	Demonstrate the threats posed to information and web security and use various tools to identify the prominent threats.	

List of Experiment

Sr. No.	Experiment List	Mapping With CO
1.	Perform encryption and decryption using Caesar cipher Algorithm.	CO1
2.	Perform encryption and decryption using Play Fair Cipher.	CO2
3.	Perform encryption and decryption using Rail Fence Technique. (Row Transposition Techniques)	CO2
4.	Perform encryption and decryption using (Columnar Transposition Techniques)	CO2
5.	Perform encryption and decryption using One Time Pad Algorithm.	CO2
6.	Write a program to Develop a Secure System by Applying RSA Cryptography Algorithms.	CO3
7.	Implement Digital signature algorithm in Java	CO3
8.	Write a program to implement Diffie–Hellman Key Exchange.	CO3
9.	Implementation of SQL Injection Attack	CO4
10.	Virtual lab Experiment	CO3
11.	Content Beyond Syllabus: Packet Tracer	CO4

Assignment 1

AIM: Perform encryption and decryption using Caesar cipher Algorithm.

Theory:

The Caesar cipher is the simplest and oldest method of cryptography. The Caesar cipher method is based on a mono-alphabetic cipher and is also called a shift cipher or additive cipher. Julius Caesar used the shift cipher (additive cipher) technique to communicate with his officers. For this reason, the shift cipher technique is called the Caesar cipher. The Caesar cipher is a kind of replacement (substitution) cipher, where all letter of plain text is replaced by another letter.

Let's take an example to understand the Caesar cipher, suppose we are shifting with 1, then A will be replaced by B, B will be replaced by C, C will be replaced by D, D will be replaced by E, and this process continues until the entire plain text is finished.

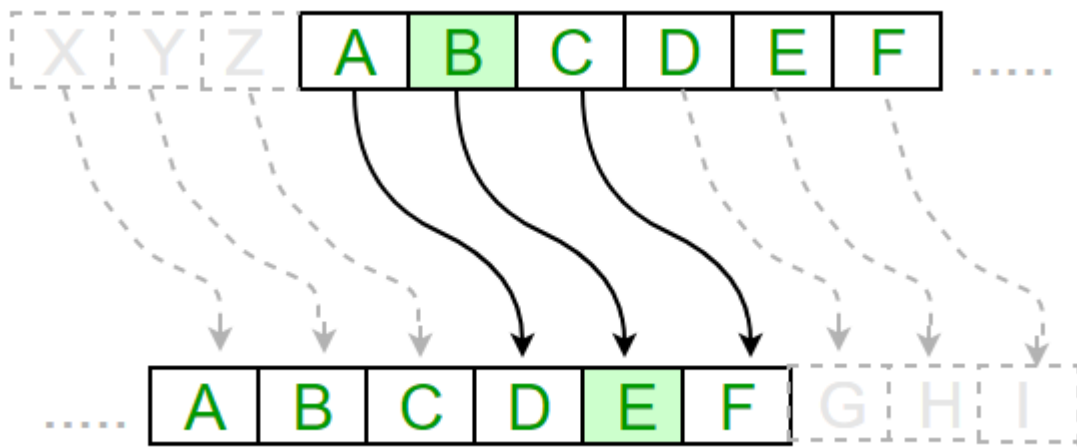
Caesar ciphers is a weak method of cryptography. It can be easily hacked. It means the message encrypted by this method can be easily decrypted.

Rules for the Caesar Cipher:

1. Choose a number between 1 and 25. This will be your "shift" value.
2. Write down the letters of the alphabet in order, from A to Z.
3. Shift each letter of the alphabet by the "shift" value. For example, if the shift value is 3, A would become D, B would become E, C would become F, and so on.
4. Encrypt your message by replacing each letter with the corresponding shifted letter. For example, if the shift value is 3, the word "hello" would become "khood".
5. To decrypt the message, simply reverse the process by shifting each letter back by the same amount. For example, if the shift value is 3, the encrypted message "khood" would become "hello".

Algorithm for Caesar Cipher:

1. Choose a shift value between 1 and 25.
2. Write down the alphabet in order from A to Z.
3. Create a new alphabet by shifting each letter of the original alphabet by the shift value. For example, if the shift value is 3, the new alphabet would be:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
4. Replace each letter of the message with the corresponding letter from the new alphabet. For example, if the shift value is 3, the word "hello" would become "khood".
5. To decrypt the message, shift each letter back by the same amount. For example, if the shift value is 3, the encrypted message "khood" would become "hello".



Advantages:

1. Easy to implement and use thus, making suitable for beginners to learn about encryption.
2. Can be physically implemented, such as with a set of rotating disks or a set of cards, known as a scytale, which can be useful in certain situations.
3. Requires only a small set of pre-shared information.
4. Can be modified easily to create a more secure variant, such as by using a multiple shift values or keywords.

Disadvantages:

1. It is not secure against modern decryption methods.
2. Vulnerable to known-plaintext attacks, where an attacker has access to both the encrypted and unencrypted versions of the same messages.
3. The small number of possible keys means that an attacker can easily try all possible keys until the correct one is found, making it vulnerable to a brute force attack.
4. It is not suitable for long text encryption as it would be easy to crack.
5. It is not suitable for secure communication as it is easily broken.
6. Does not provide confidentiality, integrity, and authenticity in a message.

Code:

```
import java.util.Scanner;
```

```
public class Assignment1E{ public static void main(String...s){
String message, encryptedMessage = "", decryptedMessage=" ";
int key;
char ch;
```

```
Scanner sc = new Scanner(System.in);
System.out.println("*****");
System.out.println("Perform encryption and decryption using Caesar cipher Algorithm. ");
```

```
System.out.println("Enter a message: ");
message = sc.nextLine();
System.out.println("Enter key: ");
key = sc.nextInt();
```

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

```

{
    ch = message.charAt(i);
    if(ch >= 'a' && ch <='z'){ ch = (char)(ch + key);
    if(ch > 'z')
    {
        ch = (char)(ch - 'z' + 'a' - 1);
    }

    encryptedMessage += ch;

}

else if(ch >= 'A' && ch <= 'Z'){ ch = (char)(ch + key);

    if(ch > 'Z'){

        ch = (char)(ch - 'Z' + 'A' - 1);

    }

    encryptedMessage += ch;

}

else {

    encryptedMessage += ch;

}

}

System.out.println("Encrypted Message = " + encryptedMessage);
System.out.println("Decrypted Message");
for(int i = 0; i < encryptedMessage.length();++i)
{
    ch = encryptedMessage.charAt(i);
    if(ch >= 'a' && ch <='z'){ ch = (char)(ch - key);
    if(ch > 'z')
    {
        ch = (char)(ch - 'z' + 'a' - 1);
    }

    decryptedMessage += ch;

}

else if(ch >= 'A' && ch <= 'Z'){ ch = (char)(ch - key);
    if(ch > 'Z'){

        ch = (char)(ch - 'Z' + 'A' - 1);

    }

}

```

```
decryptedMessage += ch;

}

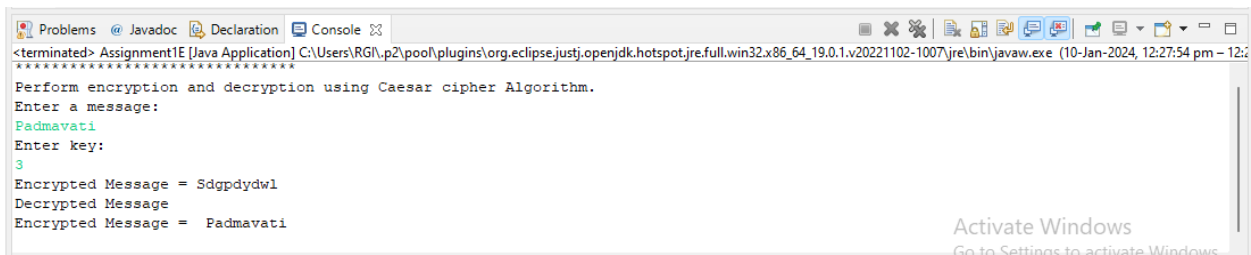
else {

decryptedMessage += ch;
}
}
System.out.println("Encrypted Message = " + decryptedMessage);

}

}
```

Output:



The screenshot shows a Java application window titled "Assignment1E [Java Application]". The console output is as follows:

```
<terminated> Assignment1E [Java Application] C:\Users\RG\p2\poof\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_19.0.1.v20221102-1007\jre\bin\javaw.exe (10-Jan-2024, 12:27:54 pm - 12:27:54 pm)
*****
Perform encryption and decryption using Caesar cipher Algorithm.
Enter a message:
Padmavati
Enter key:
3
Encrypted Message = Sdgpdydwl
Decrypted Message
Encrypted Message = Padmavati
```

An "Activate Windows" watermark is visible in the bottom right corner of the console window.

Conclusion: Thus, we have studied Ceaser Cipher Algorithm and understood that it serves as an excellent introductory algorithm for understanding cryptographic principles such as encryption, decryption, and key management.

Assignment 1 Viva Voce Questions

Viva Questions for Caesar Cipher Practical

1. What is the Caesar cipher? Explain its working.
2. How does the Caesar cipher ensure encryption and decryption?
3. What is the role of the shift key in the Caesar cipher?
4. Can the Caesar cipher encrypt and decrypt numbers or symbols? Why or why not?
5. How can you modify the Caesar cipher to handle case sensitivity?
6. What is the computational complexity of the Caesar cipher algorithm?
7. What are the limitations of the Caesar cipher in modern cryptography?
8. How would you decrypt a Caesar cipher message if the shift key is unknown?
9. What is the difference between symmetric and asymmetric encryption, and where does the Caesar cipher fit?
10. Can the Caesar cipher be considered secure? Justify your answer.

Assignment 2

Aim: To Perform encryption and decryption using Play Fair Cipher.

Theory:

Playfair cipher is proposed by Charles Whetstone in 1889. But it was named for one of his friends Lord Lyon Playfair because he popularized its uses. It is the most popular symmetric encryption technique that falls under the substitution cipher. It is an encoding procedure that enciphers more than one letter at a time.

Playfair cipher is an encryption algorithm to encrypt or encode a message. It is the same as a traditional cipher. The only difference is that it encrypts a digraph (a pair of two letters) instead of a single letter.

It initially creates a key-table of 5*5 matrix. The matrix contains alphabets that act as the key for encryption of the plaintext. Note that any alphabet should not be repeated. Another point to note that there are 26 alphabets and we have only 25 blocks to put a letter inside it. Therefore, one letter is excess so, a letter will be omitted (usually J) from the matrix. Nevertheless, the plaintext contains J, then J is replaced by I. It means treat I and J as the same letter, accordingly.

Since Playfair cipher encrypts the message digraph by digraph. Therefore, the Playfair cipher is an example of a digraph substitution cipher.

Advantages of Playfair Cipher

1. Diverse ciphertext if we scrutinize the Algorithm, we can notice at every Stage we are getting diverse ciphertext, thus more trouble to cryptanalyst.
2. Brute force attack does not affect it.
3. Cryptanalyze (the process of decoding cipher without knowing key) is not possible.
4. Overcomes the limitation of simple Playfair square cipher.
5. Easy to perform the substitution.

Limitations of Playfair Cipher

1. Only 25 alphabets are supported.
2. It does not support numeric characters.
3. Only either upper cases or lower cases are supported.
4. The use of special characters (such as blank space, newline, punctuations, etc.) is prohibited.
5. It does not support other languages, except English.
6. Encryption of media files is also not supported.

Playfair Cipher Encryption Rules

1. First, split the plaintext into **digraphs** (pair of two letters). If the plaintext has the odd number of letters, append the letter **Z** at the end of the plaintext. It makes the plaintext of **even** for example, the plaintext **MANGO** has five letters. So, it is not possible to make a digraph. Since, we will append a letter **Z** at the end of the plaintext, i.e. **MANGOZ**.

2. After that, break the plaintext into **digraphs** (pair of two letters). If any letter appears twice (side by side), put **X** at the place of the second occurrence. Suppose, the plaintext is **COMMUNICATE** then its digraph becomes **CO MX MU NI CA TE**. Similarly, the digraph for the plaintext **JAZZ** will be **JA ZX ZX**, and for plaintext **GREET**, the digraph will be **GR EX ET**.

3. To determine the cipher (encryption) text, first, build a 5*5 key-matrix or key-table and filled it with the letters of alphabets, as directed below:

- Fill the first row (left to right) with the letters of the given keyword (**ATHENS**). If the keyword has duplicate letters (if any) avoid them. It means a letter will be considered only once. After that, fill the remaining letters in alphabetical order. Let's create a 5*5 key-matrix for the keyword ATHENS.

A	T	H	E	N
S	B	C	D	F
G	I/J	K	L	M
O	P	Q	R	U
V	W	X	Y	Z

Note that in the above matrix any letter is not repeated. The letters in the first row (in green color) represent the keyword and the remaining letters sets in alphabetical order.

4. There may be the following three conditions:

- I. If a pair of letters (digraph) appears in the same row

In this case, replace each letter of the digraph with the letters immediately to their right. If there is no letter to the right, consider the first letter of the same row as the right letter. Suppose, Z is a letter whose right letter is required, in such case, T will be right to Z.

X	A	V	I	E
R	B	C	D	F
G	H	K	L	M
N	O	P	Q	S
T	U	W	Y	Z

Right of Z

- II. If a pair of letters (digraph) appears in the same column

In this case, replace each letter of the digraph with the letters immediately below them. If there is no letter below, wrap around to the top of the same column. Suppose, W is a letter whose below letter is required, in such case, V will be below W.

X	A	V	I	E
R	B	C	D	F
G	H	K	L	M
N	O	P	Q	S
T	U	W	Y	Z

Below of W is V

iii) If a pair of letters (digraph) appears in a different row and different column

In this case, select a 3*3 matrix from a 5*5 matrix such that pair of letters appear in the 3*3 matrix. Since they occupy two opposite corners of a square within the matrix. The other corner will be a cipher for the given digraph.

In other words, we can also say that intersection of H and Y will be the cipher for the first letter and suppose, a digraph is HY and we have to find a cipher for it. We observe that both H and Y are placed in different rows and different columns. In such cases, we have to select a 3*3 matrix in such a way that both H and Y appear in the 3*3 matrix (highlighted with yellow color). Now, we will consider only the selected matrix to find the cipher.

X	A	V	I	E
R	B	C	D	F
G	H	K	L	M
N	O	P	Q	S
T	U	W	Y	Z

Now to find the cipher for HY, we will consider the diagonal **opposite** to HY, i.e. **LU**. Therefore, the cipher for **H** will be **L**, and the cipher for **Y** will be **U**.

X	A	V	I	E
R	B	C	D	F
G	H	K	L	M
N	O	P	Q	S
T	U	W	Y	Z

Code:

```
package course;

import java.util.Arrays;

import java.util.Scanner;

public class demo {
```

```
private static char[][] keySquare;

private static void generateKeySquare(String key) {

    key = key.replace("J", "I").toUpperCase();

    key = key.replaceAll("[^A-Z]", "");

    String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    String combinedKey = key + alphabet;

    combinedKey = combinedKey.replaceAll("(.)?(?=.*\\1)", ""); // Remove duplicate characters

    keySquare = new char[5][5];

    int rowIndex = 0;

    int colIndex = 0;

    for (char ch : combinedKey.toCharArray()) {

        keySquare[rowIndex][colIndex] = ch;

        colIndex++;

        if (colIndex == 5) {

            colIndex = 0;

            rowIndex++;

        }

    }

}

private static String preparePlainText(String plainText) {

    plainText = plainText.replace("J", "I").toUpperCase();

    plainText = plainText.replaceAll("[^A-Z]", "");

    StringBuilder preparedText = new StringBuilder(plainText);

    for (int i = 0; i < preparedText.length(); i += 2) {

        if (i + 1 == preparedText.length()) {

            preparedText.append('X');

        } else if (preparedText.charAt(i) == preparedText.charAt(i + 1)) {

            preparedText.insert(i + 1, 'X');
```

```

    }

    }

    return preparedText.toString();

    }

    private static String encrypt(String plainText) {

        StringBuilder encryptedText = new StringBuilder();

        for (int i = 0; i < plainText.length(); i += 2) {

            char ch1 = plainText.charAt(i);

            char ch2 = plainText.charAt(i + 1);

            int row1 = -1, col1 = -1, row2 = -1, col2 = -1;

            for (int row = 0; row < 5; row++) {

                for (int col = 0; col < 5; col++) {

                    if (keySquare[row][col] == ch1) {

                        row1 = row;

                        col1 = col;

                    }

                    if (keySquare[row][col] == ch2) {

                        row2 = row;

                        col2 = col;

                    }

                }

            }

            char encryptedCh1, encryptedCh2;

            if (row1 == row2) {

                encryptedCh1 = keySquare[row1][(col1 + 1) % 5];

                encryptedCh2 = keySquare[row2][(col2 + 1) % 5];

            } else if (col1 == col2) {

                encryptedCh1 = keySquare[(row1 + 1) % 5][col1];

```

```

encryptedCh2 = keySquare[(row2 + 1) % 5][col2];

} else {

encryptedCh1 = keySquare[row1][col2];

encryptedCh2 = keySquare[row2][col1];

}

encryptedText.append(encryptedCh1).append(encryptedCh2);

}

return encryptedText.toString();

}

private static String decrypt(String encryptedText) {

StringBuilder decryptedText = new StringBuilder();

for (int i = 0; i < encryptedText.length(); i += 2) {

char ch1 = encryptedText.charAt(i);

char ch2 = encryptedText.charAt(i + 1);

int row1 = -1, col1 = -1, row2 = -1, col2 = -1;

for (int row = 0; row < 5; row++) {

for (int col = 0; col < 5; col++) {

if (keySquare[row][col] == ch1) {

row1 = row;

col1 = col;

}

if (keySquare[row][col] == ch2) {

row2 = row;

col2 = col;

}

}

}

}

char decryptedCh1, decryptedCh2;

```

```

if (row1 == row2) {

    decryptedCh1 = keySquare[row1][(col1 + 4) % 5];

    decryptedCh2 = keySquare[row2][(col2 + 4) % 5];

    } else if (col1 == col2) {

    decryptedCh1 = keySquare[(row1 + 4) % 5][col1];

    decryptedCh2 = keySquare[(row2 + 4) % 5][col2];

    } else {

    decryptedCh1 = keySquare[row1][col2];

    decryptedCh2 = keySquare[row2][col1];

    }

    decryptedText.append(decryptedCh1).append(decryptedCh2);

}

return decryptedText.toString();

}

public static void main(String[] args) {

    String key = "KEYWORD";

    generateKeySquare(key);

    Scanner scan = new Scanner(System.in); // Take input from user using scanner class

    String plainText = scan.nextLine();

    String preparedText = preparePlainText(plainText);

    String encryptedText = encrypt(preparedText);

    String decryptedText = decrypt(encryptedText);

    System.out.println("Key Square:");

    for (char[] row : keySquare) {

        System.out.println(Arrays.toString(row));

    }

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

    System.out.println("Prepared Text: " + preparedText);

```

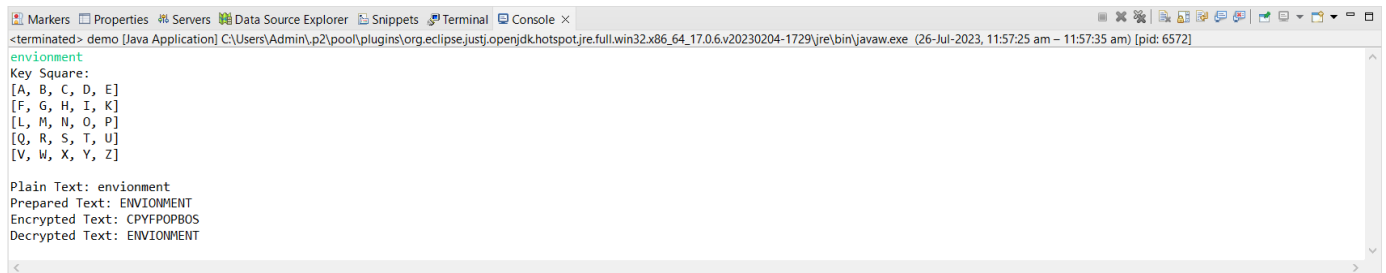
```
System.out.println("Encrypted Text: " + encryptedText);

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

}

}
```

Output:



```
<terminated> demo [Java Application] C:\Users\Admin\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin\javaw.exe (26-Jul-2023, 11:57:25 am - 11:57:35 am) [pid: 6572]
environment
Key Square:
[A, B, C, D, E]
[F, G, H, I, K]
[L, M, N, O, P]
[Q, R, S, T, U]
[V, W, X, Y, Z]

Plain Text: environment
Prepared Text: ENVIRONMENT
Encrypted Text: CPYFPOPBOS
Decrypted Text: ENVIRONMENT
```

Conclusion: Thus, we have studied Playfair Cipher Algorithm. The Playfair Cipher is a manual encryption technique that improves on simple substitution by encrypting digraphs (pairs of letters) instead of single letters, making it more resistant to frequency analysis. While more secure than the Caesar Cipher, it is still vulnerable to modern cryptanalysis due to its reliance on predictable patterns.

Assignment 2 Viva Voce Questions

Viva Questions for Playfair Cipher Practical

1. What is the Playfair cipher, and how does it differ from the Caesar cipher?
2. Explain the significance of the key matrix in the Playfair cipher.
3. How are repeated letters handled in the plaintext pairs?
4. Why are 'I' and 'J' treated as the same letter in the Playfair cipher?
5. What are the encryption rules when the two letters in a pair are in the same row or column?
6. What happens if the letters in a pair form a rectangle in the key matrix?
7. How is a filler character used, and why is it necessary?
8. What are the limitations of the Playfair cipher in modern cryptography?
9. Can the Playfair cipher encrypt and decrypt numeric data? Why or why not?
10. How would you adapt the Playfair cipher to handle special characters or different alphabets?

Assignment 3

AIM: Perform encryption and decryption using Rail Fence Technique. (Row Transposition Techniques)

```
import java.util.Arrays;
```

Theory:

The Rail Fence Technique is a simple transposition cipher that encrypts a message by rearranging the order of the letters in a zigzag pattern. It is one of the oldest and simplest encryption techniques known, and is relatively easy to break, but it can still be useful for simple communication where a high level of security is not required.

To encrypt a message using the Rail Fence Technique, you will need to choose a number of rails. The most common is two rails, but you can use more or less depending on the desired level of security.

Once you have chosen a number of rails, write the message in a zigzag pattern, starting at the top left rail and moving down to the bottom rail, then back up to the top rail, and so on. If you reach the end of a rail before you finish writing the message, simply wrap around to the next rail.

Once you have written the entire message in this zigzag pattern, read off the ciphertext by reading each rail from left to right, in order.

Encryption Algorithm:

1. Key Setup:
 - a. Determine the number of rails or lines to use for the zigzag pattern. This is typically a positive integer.
 - b. Create the rails by setting up as many empty lines as specified by the key.
2. Text Preparation:
 - a. Remove any spaces and punctuation from the plaintext. You may also choose to convert the text to uppercase to simplify the process.
3. Encryption:
 - a. Start at the top rail (rail 1) and write the first letter of the plaintext.
 - b. Move down to the next rail (rail 2) and write the second letter of the plaintext.
 - c. Continue moving down in a zigzag pattern, placing each letter of the plaintext on the appropriate rail.
 - d. When you reach the bottom rail (the last rail), reverse direction and start moving upward.
 - e. Repeat this process until you have placed all the letters of the plaintext on the rails.
4. Ciphertext Formation:
 - a. Read the letters from the rails in order, starting from the top rail to the bottom rail.
 - b. This forms the ciphertext.

Decryption Algorithm:

1. Key Setup:
 - a. Determine the number of rails or lines used for encryption (the same value as used in encryption).
2. Text Preparation:

- a. Remove any spaces and punctuation from the ciphertext. You may also choose to convert the text to uppercase to simplify the process.
3. Decryption:
 - a. Create an empty rail fence with the same number of rails as in the encryption process.
 - b. Filling the Rails:
 - c. Start at the top rail (rail 1) and place the first letter of the ciphertext.
 - d. Move down to the next rail (rail 2) and place the next letter of the ciphertext.
 - e. Continue moving down in a zigzag pattern, placing each letter of the ciphertext on the appropriate rail.
 - f. When you reach the bottom rail (the last rail), reverse direction and start moving upward.
 - g. Repeat this process until you have placed all the letters of the ciphertext on the rails.
4. Recovery of Plaintext:
 - a. Read the letters from the rails in the same zigzag pattern used during encryption.
 - b. This forms the plaintext.

Plaintext	T	H	I	S	I	S	A	S	E	C	R	E	T	M	E	S	S	A	G	E
Rail Fence	T				I				E				T				S			
Encoding		H		S		S		S		C		E		M		S		A		E
key = 3			I				A				R				E				G	
Ciphertext	T	I	E	T	S	H	S	S	S	C	E	M	S	A	E	I	A	R	E	G

Advantages:

1. Ease of Implementation: The Rail Fence Cipher is straightforward to understand and implement, making it accessible to beginners and those new to cryptography. It doesn't require complex mathematical operations or special equipment.
2. Speed: Both encryption and decryption in the Rail Fence Cipher can be performed quickly since they involve simple rearrangements of letters. This makes it suitable for basic and rapid encryption needs.
3. Visual Appeal: The encryption and decryption processes in the Rail Fence Cipher can be visually appealing, as they result in a zigzag pattern. This can make it engaging for educational or recreational purposes.

Disadvantages:

1. Weak Security: The Rail Fence Cipher provides very weak security and is not suitable for protecting sensitive information against modern cryptographic attacks. It can be easily broken using techniques such as frequency analysis, guessing, or manual decryption.
2. Lack of Key: The Rail Fence Cipher lacks a cryptographic key, which means that anyone with knowledge of the algorithm can decrypt the message without requiring a secret key. This lack of key management makes it unsuitable for secure communication.
3. Limited Applicability: Due to its simplicity and weak security, the Rail Fence Cipher is not suitable for most practical encryption scenarios. It may be suitable for simple puzzles, games, or teaching cryptography concepts but should not be used for secure communication.
4. Susceptible to Known-Plaintext Attack: Since the pattern of the rails remains the same for a given number of rails, if an attacker has knowledge of any part of the plaintext and its corresponding ciphertext, they can easily deduce the number of rails and decrypt the entire message.

CODE:

```
class RailFence {  
    // function to encrypt a message  
    public static String encryptRailFence(String text, int key)  
    {  
        // create the matrix to cipher plain text  
        // key = rows , length(text) = columns  
        char[][] rail = new char[key][text.length()];  
        // filling the rail matrix to distinguish filled  
        // spaces from blank ones  
        for (int i = 0; i < key; i++)  
            Arrays.fill(rail[i], '\n');  
        boolean dirDown = false;  
        int row = 0, col = 0;  
        for (int i = 0; i < text.length(); i++) {  
            // check the direction of flow  
            // reverse the direction if we've just  
            // filled the top or bottom rail  
            if (row == 0 || row == key - 1)  
                dirDown = !dirDown;  
            // fill the corresponding alphabet  
            rail[row][col++] = text.charAt(i);  
            // find the next row using direction flag  
            if (dirDown)  
                row++;  
            else  
                row--;  
        }  
        // now we can construct the cipher using the rail  
        // matrix
```

```

StringBuilder result = new StringBuilder();

for (int i = 0; i < key; i++)
    for (int j = 0; j < text.length(); j++)
        if (rail[i][j] != '\n')
            result.append(rail[i][j]);

return result.toString();
}

// This function receives cipher-text and key
// and returns the original text after decryption

public static String decryptRailFence(String cipher, int key)
{
    // create the matrix to cipher plain text
    // key = rows , length(text) = columns
    char[][] rail = new char[key][cipher.length()];

    // filling the rail matrix to distinguish filled
    // spaces from blank ones
    for (int i = 0; i < key; i++)
        Arrays.fill(rail[i], '\n');

    // to find the direction
    boolean dirDown = true;

    int row = 0, col = 0;

    // mark the places with '*'
    for (int i = 0; i < cipher.length(); i++) {

        // check the direction of flow
        if (row == 0)
            dirDown = true;

        if (row == key - 1)
            dirDown = false;

        // place the marker
        rail[row][col++] = '*';
    }
}

```

```
// find the next row using direction flag

if (dirDown)

    row++;

else

    row--;

}

// now we can construct the fill the rail matrix

int index = 0;

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

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

        if (rail[i][j] == '*'

            && index < cipher.length())

            rail[i][j] = cipher.charAt(index++);

StringBuilder result = new StringBuilder();

row = 0;

col = 0;

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

    // check the direction of flow

    if (row == 0)

        dirDown = true;

    if (row == key - 1)

        dirDown = false;

    if (rail[row][col] != '*')

        result.append(rail[row][col++]);

    // find the next row using direction flag

    if (dirDown)

        row++;

    else

        row--;

}
```

```

        return result.toString();
    }

    public static void main(String[] args)
    {
        // Encryption

        System.out.println("Encrypted Message: ");

        System.out.println(encryptRailFence("attack at once", 2));

        System.out.println(encryptRailFence("GeeksforGeeks ", 3));

        System.out.println(encryptRailFence("defend the east wall", 3));

        // Now decryption of the same cipher-text

        System.out.println("\nDecrypted Message: ");

        System.out.println(decryptRailFence("atc toctaka ne", 2));

        System.out.println(decryptRailFence("GsGsekfrek eoe", 3));

        System.out.println(decryptRailFence("dnhaweedtees alf tl", 3));

    }
}

```

OUTPUT:

```

atc toctaka ne
GsGsekfrek eoe
dnhaweedtees alf tl

```

```

Decrypted Message:
attack at once
GeeksforGeeks
defend the east wall

```

Conclusion: The Rail Fence Technique is a transposition cipher that rearranges the characters of plaintext to create ciphertext, demonstrating the basics of rearrangement in cryptography. It is simple to implement and provides a foundation for understanding more complex transposition ciphers. However, it offers minimal security as the pattern can be easily identified and reversed using basic cryptanalysis. Despite its limitations, the Rail Fence Technique is valuable for introducing the concept of transposition in encryption.

Assignment 3 Viva Voce Questions

1. What is the Rail Fence Cipher, and how does it work?
2. Explain the encryption and decryption process using the zigzag pattern.
3. What are the main steps in encrypting and decrypting a message using the Rail Fence Cipher?
4. How does the number of rails affect the security of the Rail Fence Cipher?
5. What happens if the length of the plaintext is not a multiple of the number of rails?
6. Explain the concept of "zigzag" writing in the Rail Fence Cipher. How does it contribute to encryption?
7. Can the Rail Fence Cipher be used for longer or more complex messages? Why or why not?
8. How would you adapt the Rail Fence Cipher to handle spaces and punctuation marks in the plaintext?
9. What are the main weaknesses of the Rail Fence Cipher? How does it compare to modern cryptographic methods?
10. Can the Rail Fence Cipher be broken easily by an attacker? Justify your answer with reasoning.

Assignment 4

AIM: Perform encryption and decryption using (Columnar Transposition Techniques)

Theory:

Columnar transposition techniques are a type of transposition cipher that encrypts a message by rearranging the order of the letters in columns. The columns are then read in a different order to produce the ciphertext.

Columnar transposition techniques can be used with any number of columns, but the more columns you use, the more difficult the cipher will be to break. However, even with a large number of columns, columnar transposition techniques are still relatively weak ciphers, and can be broken by a skilled cryptanalyst.

Columnar transposition techniques can be made more difficult to break by using a longer keyword, or by using a keyword with repeated letters. However, even the most complex columnar transposition techniques can be broken by a skilled cryptanalyst.

Here are some additional techniques that can be used to improve the security of columnar transposition techniques:

1. Use a random number generator to generate the keyword.
2. Use a keyword that is not related to the plaintext message.
3. Use a keyword that is long enough to make it difficult to guess.
4. Use a keyword with repeated letters.
5. Use a combination of columnar transposition and other encryption techniques, such as substitution or polyalphabetic substitution.

Despite their weaknesses, columnar transposition techniques are still useful for simple communication where a high level of security is not required. For example, columnar transposition techniques can be used to encrypt messages on social media or in email.

Encryption:

In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

1. The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
2. Width of the rows and the permutation of the columns are usually defined by a keyword.
3. For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be "3 1 2 4".
4. Any spare spaces are filled with nulls or left blank or placed by a character (Example: _).
5. Finally, the message is read off in columns, in the order specified by the keyword.

Encryption

Given text = Geeks for Geeks

Keyword = HACK

Length of Keyword = 4 (no of rows)

Order of Alphabets in HACK = 3124

	H	A	C	K
	3	1	2	4
	G	e	e	k
	s	_	f	o
	r	_	G	e
	e	k	s	_

Print Characters of column 1,2,3,4

Encrypted Text = e kefGsGsrekoe_

Decryption:

1. To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length.
2. Then, write the message out in columns again, then re-order the columns by reforming the key word.

Advantages of columnar transposition techniques:

1. Simple to implement and understand.
2. Can be used with any number of columns.
3. Can be made more difficult to break by using a longer keyword or a keyword with repeated letters.
4. Can be combined with other encryption techniques, such as substitution or polyalphabetic substitution, to improve security.

Disadvantages of columnar transposition techniques:

1. Relatively weak ciphers that can be broken by a skilled cryptanalyst.
2. Can be time-consuming to encrypt and decrypt long messages.
3. Errors in encryption or decryption can make the message unreadable.

Code:

```
public class ColumnarTranspositionCipher {  
  
    public static String encrypt(String message, String keyword) {  
  
        // Create a matrix to store the plaintext message.  
  
        int keyLength = keyword.length();  
  
        char[][] matrix = new char[keyLength][message.length()];  
  
        // Write the plaintext message to the matrix.  
  
        int row = 0;  
  
        int col = 0;  
  
        for (int i = 0; i < message.length(); i++) {  
  
            matrix[row][col] = message.charAt(i);
```

```

        row++;

        if (row == keyLength) {

            row = 0;

            col++;

        }

    }

    // Order the columns by the alphabetical order of the keyword.

    int[] columnOrder = new int[keyLength];

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

        columnOrder[i] = i;

    }

    // Sort the column order.

    Arrays.sort(columnOrder, (o1, o2) -> Character.compare(keyword.charAt(o1),
keyword.charAt(o2)));

    // Read the ciphertext off column by column, in the order specified by the column order.

    String ciphertext = "";

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

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

            ciphertext += matrix[j][columnOrder[i]];

        }

    }

    return ciphertext;

}

public static String decrypt(String ciphertext, String keyword) {

    // Create a matrix to store the ciphertext.

    int keyLength = keyword.length();

    char[][] matrix = new char[keyLength][ciphertext.length()];

    // Order the columns by the alphabetical order of the keyword.

    int[] columnOrder = new int[keyLength];

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

```

```

        columnOrder[i] = i;
    }

    // Sort the column order.

    Arrays.sort(columnOrder, (o1, o2) -> Character.compare(keyword.charAt(o1),
keyword.charAt(o2)));

    // Write the ciphertext to the matrix, in the order specified by the column order.

    int row = 0;

    int col = 0;

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

        matrix[row][columnOrder[col]] = ciphertext.charAt(i);

        col++;

        if (col == keyLength) {

            col = 0;

            row++;

        }

    }

    // Read the plaintext off row by row, from left to right.

    String plaintext = "";

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

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

            plaintext += matrix[j][i];

        }

    }

    return plaintext;

}

public static void main(String[] args) {

    String message = "SECRET MESSAGE";

    String keyword = "ZEBRAS";

    String ciphertext = encrypt(message, keyword);

    System.out.println("Ciphertext: " + ciphertext);

```

```
        String plaintext = decrypt(ciphertext, keyword);  
        System.out.println("Plaintext: " + plaintext);  
    }  
}
```

Output:

Ciphertext: SECMRETESSAGE

Plaintext: SECRET MESSAGE

Conclusion: The Columnar Transposition Technique is a classical cipher that rearranges the characters of plaintext into columns based on a key, adding complexity to the ciphertext compared to simpler methods. It effectively demonstrates the principle of transposition in cryptography, emphasizing the importance of key-based rearrangement. While it provides more security than basic ciphers, it remains vulnerable to frequency analysis and known-plaintext attacks. Overall, it serves as an essential step in understanding the role of structured patterns in cryptographic systems.

Assignment 4 Viva Voce Questions

Viva Questions for Rail Fence Cipher Practical

1. What is the Rail Fence Cipher, and how does it work?
2. Explain the encryption and decryption process using the zigzag pattern.
3. What are the main steps in encrypting and decrypting a message using the Rail Fence Cipher?
4. How does the number of rails affect the security of the Rail Fence Cipher?
5. What happens if the length of the plaintext is not a multiple of the number of rails?
6. Explain the concept of "zigzag" writing in the Rail Fence Cipher. How does it contribute to encryption?
7. Can the Rail Fence Cipher be used for longer or more complex messages? Why or why not?
8. How would you adapt the Rail Fence Cipher to handle spaces and punctuation marks in the plaintext?
9. What are the main weaknesses of the Rail Fence Cipher? How does it compare to modern cryptographic methods?
10. Can the Rail Fence Cipher be broken easily by an attacker? Justify your answer with reasoning.
11. What is the computational complexity of the Rail Fence Cipher encryption and decryption processes?

Assignment 5

Aim: Perform encryption and decryption using One Time Pad Algorithm.

Theory:

One Time Pad algorithm (OTP) is a symmetric key type algorithm. In the encryption process, this algorithm uses a stream cipher derived from the XOR result between the plaintext bit and the key bit.

It is the only available algorithm that is unbreakable (completely secure). It is one of the Substitution techniques which converts plain text into ciphertext

The two requirements for the One-Time pad are

- The key should be randomly generated & should be as long as the size of the message.
- The key is to be used to encrypt and decrypt a single message, and then it is discarded.

So, encrypting every new message requires a new key of the same length as the new message in one-time pad. The ciphertext generated by the One-Time pad is random, so it does not have any statistical relation with the plain text.

A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9
K	L	M	N	O	P	Q	R	S	T
10	11	12	13	14	15	16	17	18	19
U	V	W	X	Y	Z				
20	21	22	23	24	25				

Examples:

Input: Message = HELLO, Key = MONEY, Output: Cipher = TSYPM,

Explanation: Part 1: Plain text to Ciphertext

Plain text = H E L L O \rightarrow 7 4 11 11 14

Key = M O N E Y \rightarrow 12 14 13 4 24

Plain text + key \rightarrow 19 18 24 15 38 \rightarrow 19 18 24 15 12 (because $38 - 26 = 12$)

Cipher Text \rightarrow T S Y P M

Part 2: Ciphertext to Message

Cipher Text = 19 18 24 15 12 \rightarrow T S Y P M

Key = M O N E Y \rightarrow 12 14 13 4 24

Cipher text - key \rightarrow 7 4 11 11 -12 \rightarrow 7 4 11 11 14 ($-12+26 = 14$)

Plain text = 7 4 11 11 14 \rightarrow H E L L O

Security of One-Time Pad

- If any way cryptanalyst finds these two keys using which two plaintext are produced but if the key was produced randomly, then the cryptanalyst cannot find which key is more likely than the other. In fact, for any plaintext as the size of ciphertext, a key exists that produces that plaintext.
- So if a cryptanalyst tries the brute force attack(try using all possible keys), he would end up with many legitimate plaintexts, with no way of knowing which plaintext is legitimate. Therefore, the code is unbreakable.
- The security of the one-time pad entirely depends on the randomness of the key. If the characters of the key are truly random, then the characters of the ciphertext will be truly random. Thus, there are no patterns or regularities that a cryptanalyst can use to attack the ciphertext.

Advantages

- One-Time Pad is the only algorithm that is truly unbreakable and can be used for low-bandwidth channels requiring very high security (ex. for military uses).

Disadvantages

- There is the practical problem of making large quantities of random keys. Any heavily used system might require millions of random characters on a regular basis.
- For every message to be sent, a key of equal length is needed by both sender and receiver. Thus, a mammoth key distribution problem exists.

Code:

```
package course;

import java.util.Random;

import java.util.Scanner;

public class OneTimePad {

    // Function to generate a random key (pad) of the same length as the plaintext

    public static String generateRandomKey(int length) {

        Random random = new Random();

        StringBuilder keyBuilder = new StringBuilder();

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

            char randomChar = (char) (random.nextInt(26) + 'A'); // Generates a random uppercase letter

            keyBuilder.append(randomChar);

        }

        return keyBuilder.toString();

    }

    // Function to perform one-time pad encryption
```



```
public static String encrypt(String plaintext, String key) {  
  
    if (plaintext.length() != key.length()) {  
  
        throw new IllegalArgumentException("Plaintext and key must have the same length.");  
  
    }  
  
    StringBuilder ciphertextBuilder = new StringBuilder();  
  
    for (int i = 0; i < plaintext.length(); i++) {  
  
        char encryptedChar = (char) ((plaintext.charAt(i) + key.charAt(i)) % 26 + 'A');  
  
        ciphertextBuilder.append(encryptedChar);  
  
    }  
  
    return ciphertextBuilder.toString();  
  
    }  
  
    // Function to perform one-time pad decryption  
  
    public static String decrypt(String ciphertext, String key) {  
  
        if (ciphertext.length() != key.length()) {  
  
            throw new IllegalArgumentException("Ciphertext and key must have the same length.");  
  
        }  
  
        StringBuilder decryptedBuilder = new StringBuilder();  
  
        for (int i = 0; i < ciphertext.length(); i++) {  
  
            char decryptedChar = (char) ((ciphertext.charAt(i) - key.charAt(i) + 26) % 26 + 'A');  
  
            decryptedBuilder.append(decryptedChar);  
  
        }  
  
        return decryptedBuilder.toString();  
  
    }  
  
    public static void main(String[] args) {  
  
        // Input string from user  
  
        Scanner scan = new Scanner(System.in);  
  
        String randomtext = scan.nextLine();  
  
        String plaintext = randomtext.toUpperCase();
```

```
String key = generateRandomKey(plaintext.length());

System.out.println("Plaintext: " + plaintext);

System.out.println("Key: " + key);

String ciphertext = encrypt(plaintext, key);

System.out.println("Ciphertext: " + ciphertext);

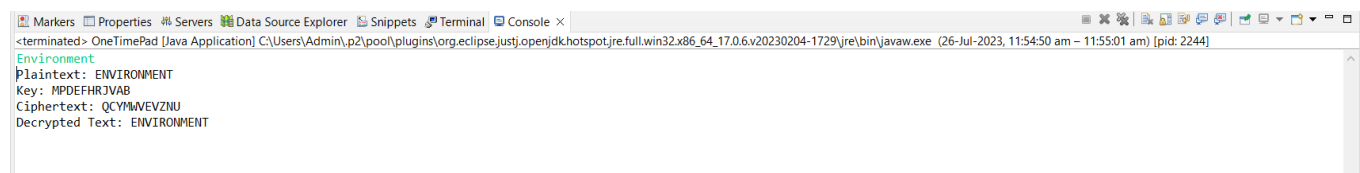
String decryptedText = decrypt(ciphertext, key);

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

}

}
```

Output:

A screenshot of an IDE's console window. The title bar shows various tabs like 'Markers', 'Properties', 'Servers', 'Data Source Explorer', 'Snippets', 'Terminal', and 'Console'. The active console tab displays the following output:

```
<terminated> OneTimePad [Java Application] C:\Users\Admin\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin\javaw.exe (26-Jul-2023, 11:54:50 am - 11:55:01 am) [pid: 2244]
Environment
Plaintext: ENVIRONMENT
Key: MPDEFHRJVAB
Ciphertext: QCYMVEVZNU
Decrypted Text: ENVIRONMENT
```

Conclusion: The **One-Time Pad (OTP)** algorithm is a symmetric encryption technique that offers perfect secrecy when used correctly. It involves combining plaintext with a random key (pad) of the same length using bitwise XOR. Each key is used only once, ensuring that the ciphertext is unbreakable as long as the key remains secret and truly random.

Assignment 5 Viva Voce Questions

Viva Questions for One Time Pad Algorithm

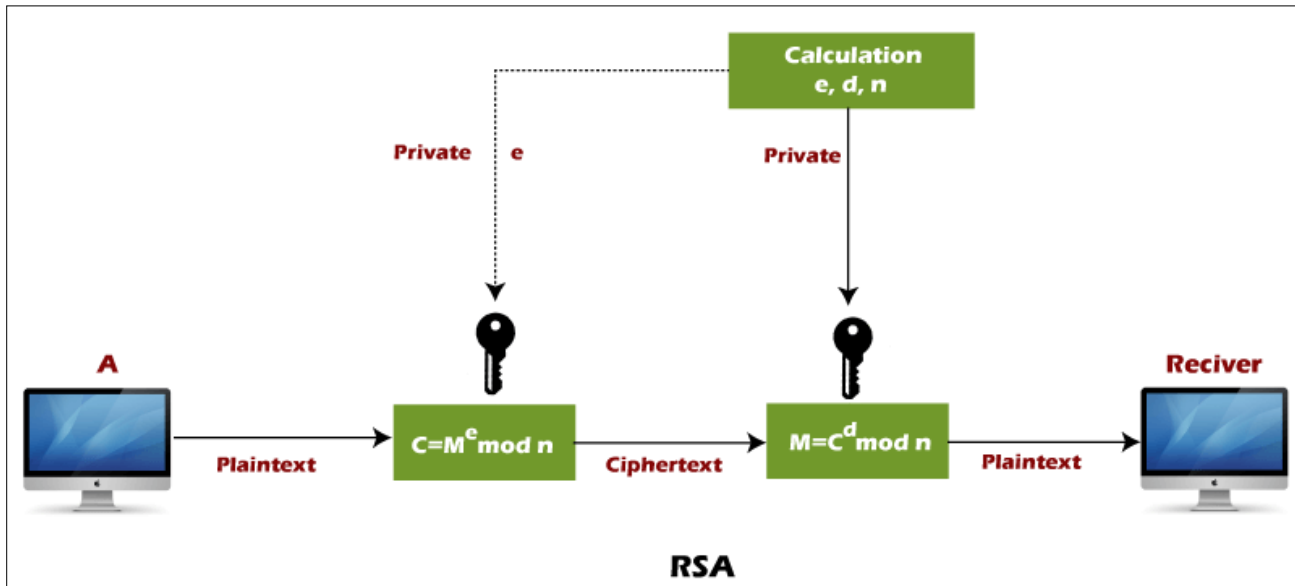
1. What is the One Time Pad (OTP) cipher, and how does it differ from other symmetric ciphers like the Caesar Cipher or Vigenère Cipher?
2. Explain the concept and how OTP uses a random key to achieve unbreakable encryption.
3. Why is the key in the OTP cipher required to be truly random, and how does this impact the security of the encryption?
4. Discuss the importance of key randomness and how any pattern in the key makes the cipher insecure.
5. How does the XOR operation work in the OTP encryption and decryption process?
6. Explain how XOR is used for both encryption and decryption in OTP.
7. What would happen if the key used in the OTP encryption is reused?
8. Discuss the security risks of using the same key for multiple messages and why it's called a "one-time" pad.
9. Can the OTP cipher be broken using brute-force attacks?
10. Why is the OTP cipher considered unbreakable when the key is truly random and only used once?

Assignment 6

AIM: Write a program to Develop a Secure System by Applying RSA Cryptography Algorithms.

Theory:

RSA encryption algorithm is a type of public-key encryption algorithm. The **Public key** is used for encryption, and the **Private Key** is used for decryption. RSA is the most common public-key algorithm, named after its inventors **Rivest, Shamir, and Adelman (RSA)**.



RSA algorithm uses the following procedure to generate public and private keys:

- Select two large prime numbers, p and q .
- Multiply these numbers to find $n = p \times q$, where n is called the modulus for encryption and decryption.
- Choose a number e less than n , such that n is relatively prime to $(p - 1) \times (q - 1)$. It means that e and $(p - 1) \times (q - 1)$ have no common factor except 1. Choose "e" such that $1 < e < \phi(n)$, e is prime to $\phi(n)$,
 $\gcd(e, \phi(n)) = 1$
- If $n = p \times q$, then the public key is $\langle e, n \rangle$. A plaintext message m is encrypted using public key $\langle e, n \rangle$. To find ciphertext from the plain text following formula is used to get ciphertext C .
 $C = m^e \bmod n$
Here, m must be less than n . A larger message ($> n$) is treated as a concatenation of messages, each of which is encrypted separately.
- To determine the private key, we use the following formula to calculate the d such that:
 $D_e \bmod \{(p-1) \times (q-1)\} = 1$
Or
 $D_e \bmod \phi(n) = 1$
- The private key is $\langle d, n \rangle$. A ciphertext message c is decrypted using private key $\langle d, n \rangle$. To calculate plain text m from the ciphertext c following formula is used to get plain text m .
 $m = c^d \bmod n$

- **Let us learn the mechanism behind the RSA algorithm :**

>> Generating Public Key:

- Select two prime no's. Suppose **P = 53** and **Q = 59**.
- **Now First part of the Public key : $n = P * Q = 3127$.**
- We also need a small exponent say **e** :
- **But e Must be An integer.**
- **Not be a factor of $\Phi(n)$.**
- **$1 < e < \Phi(n)$ [$\Phi(n)$ is discussed below],**
- **Let us now consider it to be equal to 3.**

- Our Public Key is made of n and e

>> Generating Private Key:

- We need to calculate $\Phi(n)$:
- Such that **$\Phi(n) = (P-1)(Q-1)$**
- **so, $\Phi(n) = 3016$**
- Now calculate Private Key, **d** :
- **$d = (k * \Phi(n) + 1) / e$ for some integer k**
- **For k = 2, value of d is 2011.**
- Now we are ready with our – Public Key (n = 3127 and e = 3) and Private Key(d = 2011) Now we will encrypt **“HI”**:
- Convert letters to numbers : H = 8 and I = 9
- Thus **Encrypted Data $c = (89e) \bmod n$**
- **Thus our Encrypted Data comes out to be 1394**
- Now we will decrypt **1394** :
- **Decrypted Data = $(cd) \bmod n$**
- **Thus our Encrypted Data comes out to be 89**
- **8 = H and I = 9 i.e. "HI".**

Code:

```
import java.io.*;
```

```
import java.math.*;

class rsa

{

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

{

int q,p,n,pn,publickey=0,d=0,msg;

double cipher,ptext;

int check,check1;

DataInputStream in=new DataInputStream(System.in);

System.out.println("ENTER NO");

p=Integer.parseInt(in.readLine());

q=Integer.parseInt(in.readLine());

check=prime(p);

check1=prime(q);

if(check!=1||check1!=1)

{

System.exit(0);

}

n=p*q;

pn=(p-1)*(q-1);

for(int e=2;e<pn;e++)

{

if(gcd(e,pn)==1)

{

publickey=e;

System.out.println("PUBLIC KEY :"+e);

break;

}

}

for(int i=0;i<pn;i++)
```

```

{
d=i;
if(((d*publickey)%pn)==1)
break;
}

System.out.println("PRIVATE KEY :"+d);
System.out.println("ENTER MESSAGE ");
msg=Integer.parseInt(in.readLine());
cipher=Math.pow(msg,publickey);
cipher=cipher%n;
System.out.println("ENCRYPTED :"+cipher);
ptext=Math.pow(cipher,d);
ptext=ptext%n;
System.out.println("DECRYPTED :"+ptext);
}

static int prime(int a)
{
int flag=0;
for(int i=2;i<a;i++)
{
if(a%i==0)
{
System.out.println(a+" is not a Prime Number");
flag = 1;
return 0;
}
}
if(flag==0)
return 1;
return 1;
}

```

```
}  
  
static int gcd(int number1, int number2)  
{  
    if(number2 ==  
    0){ return  
    number1;  
    }  
    return gcd(number2, number1%number2);  
}  
}
```

Output:

ENTER NO

3

11

PUBLIC KEY :3

PRIVATE KEY :7

ENTER MESSAGE

20

ENCRYPTED :14.0

DECRYPTED :20.0

Conclusion: The RSA Cryptography Algorithm is a widely used asymmetric encryption method that ensures secure communication through the use of public and private keys. Its foundation on the mathematical difficulty of factoring large prime numbers makes it highly secure for modern applications. RSA enables both encryption and digital signatures, proving its versatility in cryptographic systems.

Assignment 6 Viva Voce Questions

Viva Questions for RSA Cryptography Algorithms

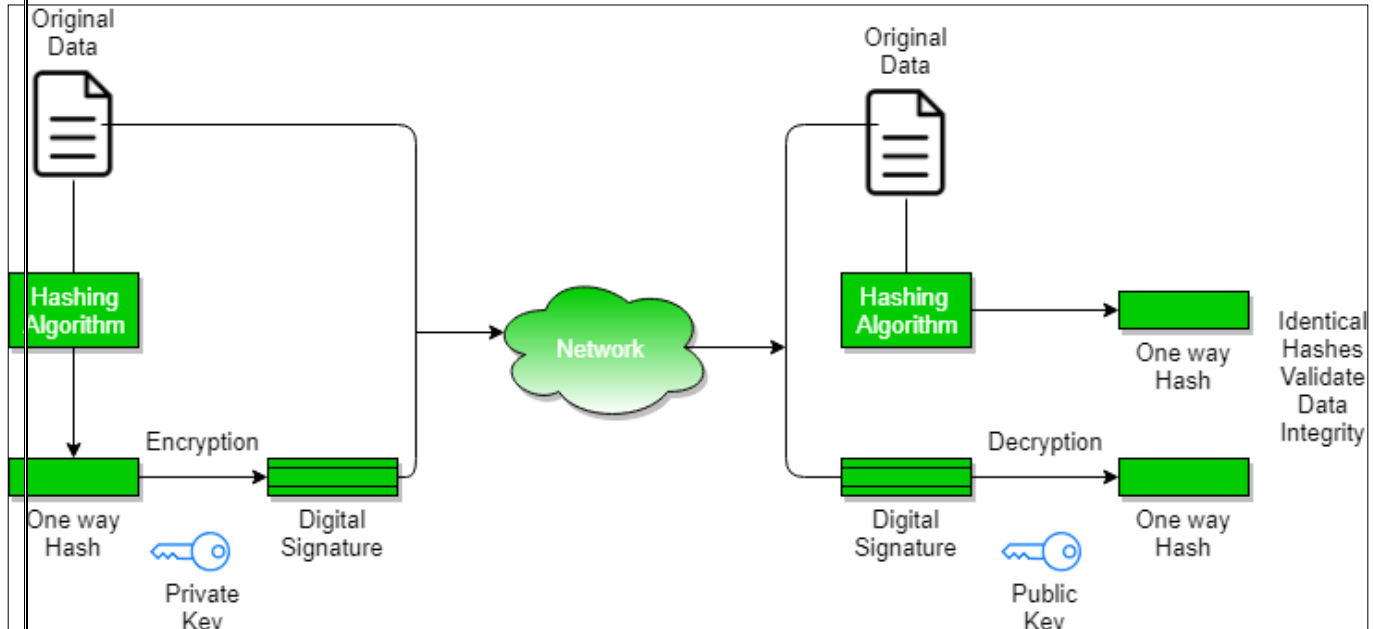
1. What is RSA encryption, and how does it differ from symmetric encryption algorithms like AES?
2. Explain the differences between symmetric and asymmetric encryption.
3. What is the significance of the public and private keys in the RSA algorithm?
4. Discuss how public and private keys are used for encryption and decryption.
5. How do you generate the public and private keys in RSA?
6. Explain the process of selecting prime numbers, calculating the totient function, and choosing the exponents.
7. What is the role of the Euler's Totient Function $\phi(n)$ in RSA key generation?
8. Describe how the totient function is calculated and its importance in the key generation process.
9. Why must the public exponent e be coprime with $\phi(n)$?
10. Explain the significance of selecting a public exponent e that is coprime to $\phi(n)$.

Assignment 7

Aim: Implement Digital signature algorithm in Java

Theory:

A digital signature is a mathematical technique used to validate the authenticity and integrity of a message, software, or digital document.



Digital Signature aids in:-

Authentication-Authentication is any process by which a system verifies the identity of a user who wishes to access it.

Non- repudiation– Non-repudiation means to ensure that a transferred message has been sent and received by the parties claiming to have sent and received the message. Non-repudiation is a way to guarantee that the sender of a message cannot later deny having sent the message and that the recipient cannot deny having received the message.

Integrity– to ensure that the message was not altered during the transmission

Signing Algorithms:

To create a digital signature, signing algorithms like email programs create a one-way hash of the electronic data which is to be signed. The signing algorithm then encrypts the hash value using the private key (signature key).

This encrypted hash along with other information like the hashing algorithm is the digital signature. This digital signature is appended with the data and sent to the verifier.

Benefits of Digital Signatures

- **Legal documents and contracts:** Digital signatures are legally binding. This makes them ideal for any legal document that requires a signature authenticated by one or more parties and guarantees that the record has not been altered.
- **Sales contracts:** Digital signing of contracts and sales contracts authenticates the identity of the seller and the buyer, and both parties can be sure that the signatures are legally binding and that the terms of the agreement have not been changed.

- **Financial Documents:** Finance departments digitally sign invoices so customers can trust that the payment request is from the right seller, not from a bad actor trying to trick the buyer into sending payments to a fraudulent account.

Drawbacks of Digital Signatures

- **Dependence on Key Management:** Digital signatures rely on the secure management of cryptographic keys. This means that the sender must keep their private key safe and secure from unauthorized access, while the recipient must verify the sender's public key to ensure its authenticity. Any failure in key management can compromise the security of the digital signature.
- **Complexity:** Digital signatures require a complex process of key generation, signing, and verification. This can make them difficult to implement and use for non-technical users.
- **Compatibility:** Different digital signature algorithms and formats may not be compatible with each other, making it difficult to exchange signed messages across different systems and applications.
- **Legal Recognition:** Although digital signatures have legal recognition in many countries, their legal status may not be clear in all jurisdictions. This can limit their usefulness in legal or regulatory contexts.
- **Revocation:** In case of key compromise or other security issues, digital signatures must be revoked to prevent their misuse. However, the revocation process can be complex and may not be effective in all cases.
- **Cost:** Digital signatures may involve additional costs for key management, certificate issuance, and other related services, which can make them expensive for some users or organizations.
- **Limited Scope:** Digital signatures provide authentication and integrity protection for a message, but they do not provide confidentiality or protection against other types of attacks, such as denial-of-service attacks or malware.

Feature	Digital Signature
Basics / Definition	Digital signature is like a fingerprint or an attachment to a digital document that ensures its authenticity and integrity.
Process / Steps	Hashed value of original message is encrypted with sender's secret key to generate the digital signature.
Security Services	Authenticity of Sender, integrity of the document and non-repudiation.
Standard	It follows Digital Signature Standard (DSS).

Code:

package practical;

```
import java.security.*;

import java.util.Base64;

public class DigitalSignature {

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

    // Generate a key pair

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

    keyPairGenerator.initialize(2048);

    KeyPair keyPair = keyPairGenerator.generateKeyPair();

    // Get the private key

    PrivateKey privateKey = keyPair.getPrivate();

    // Get the message to be signed

    String message = "This is a message to be signed.";

    // Create a signature object

    Signature signature = Signature.getInstance("SHA256withRSA");

    // Initialize the signature object with the private key

    signature.initSign(privateKey);

    // Add the message to the signature object

    signature.update(message.getBytes());

    // Calculate the signature

    byte[] signatureBytes = signature.sign();

    // Save the signature

    String signatureString = Base64.getEncoder().encodeToString(signatureBytes);

    System.out.println("Signature: " + signatureString);

    // Verify the signature

    Signature verificationSignature = Signature.getInstance("SHA256withRSA");

    // Initialize the verification signature object with the public key

    verificationSignature.initVerify(keyPair.getPublic());

    // Add the message to the verification signature object
```

```
verificationSignature.update(message.getBytes());

// Verify the signature

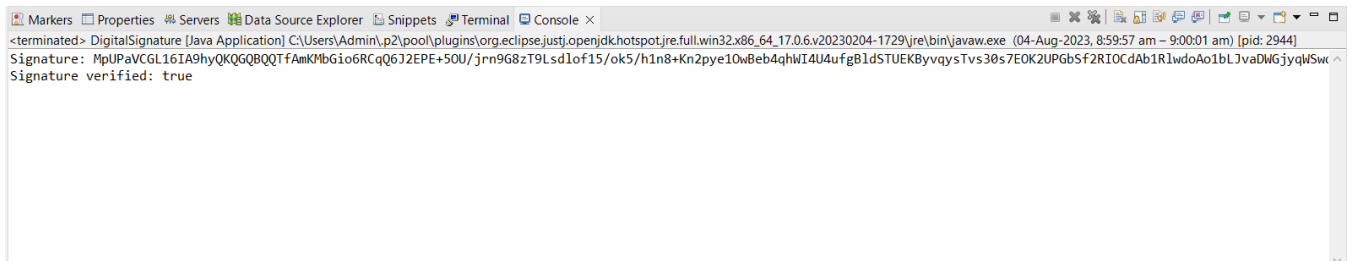
boolean isVerified = verificationSignature.verify(signatureBytes);

System.out.println("Signature verified: " + isVerified);

}

}
```

Output:

A screenshot of a Java application's console output. The window title is "DigitalSignature [Java Application]". The output shows a long alphanumeric signature string followed by the message "Signature verified: true". The console window includes standard IDE tabs like "Markers", "Properties", "Servers", "Data Source Explorer", "Snippets", "Terminal", and "Console". The terminal output is as follows:
<terminated> DigitalSignature [Java Application] C:\Users\Admin\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin\javaw.exe (04-Aug-2023, 8:59:57 am - 9:00:01 am) [pid: 2944]
Signature: MpUPaVcGL16IA9hyQKQG8QQTfAmKMbGio6RCqQ6J2EPE+5OU/jrn9G8zT9Lsd1of15/ok5/h1n8+Kn2pye10wBeb4qhI4U4ufgB1dSTUEKByvqysTvs30s7EOK2UPGbSf2RI0CdAb1RlwoAo1bLJvaDWGjyqWSw
Signature verified: true

Conclusion: The Digital Signature Algorithm (DSA) provides a robust method for verifying the authenticity and integrity of digital messages using asymmetric cryptography. It ensures non-repudiation by allowing only the holder of the private key to sign data, while the public key enables verification. DSA is efficient for secure digital communication, particularly in government and financial sectors.

Assignment 7 Viva Voce Questions

Viva Questions for Digital signature algorithm

1. What is a Digital Signature, and why is it important?
2. Explain the role of digital signatures in ensuring authenticity, integrity, and non-repudiation.
3. How does the Digital Signature Algorithm (DSA) work?
4. Discuss the key components of DSA, including key generation, signing, and verification.
5. What is the significance of hashing in the digital signature process?
6. Explain how hashing ensures data integrity and reduces computational overhead.
7. Why are private and public keys used in DSA, and how do they differ?
8. Describe the role of each key and the principle of asymmetric encryption.
9. What happens if a message is altered after being signed?
10. Explain the verification process and how it detects tampering.
11. Why do digital signatures depend on the security of the private key?

Assignment 8

Aim: Write a program to implement Diffie–Hellman Key Exchange.

Theory:

Diffie-Hellman key exchange, also called exponential key exchange, is a method of digital encryption that uses numbers raised to specific powers to produce decryption keys on the basis of components that are never directly transmitted, making the task of a would-be code breaker mathematically overwhelming.

To implement Diffie-Hellman, the two end users Alice and Bob, while communicating over a channel they know to be private, mutually agree on positive whole numbers p and q , such that p is a prime number and q is a generator of p . The generator q is a number that, when raised to positive whole-number powers less than p , never produces the same result for any two such whole numbers. The value of p may be large but the value of q is usually small.

The most serious limitation of Diffie-Hellman in its basic or "pure" form is the lack of authentication. Communications using Diffie-Hellman all by itself are vulnerable to man in the middle attacks. Ideally, Diffie-Hellman should be used in conjunction with a recognized authentication method such as digital signatures to verify the identities of the users over the public communications medium. Diffie-Hellman is well suited for use in data communication but is less often used for data stored or archived over long periods of time.

Steps in the algorithm:

1 Alice and Bob agree on a prime number p and a base g .

2 Alice chooses a secret number a , and sends Bob $(g^a \bmod p)$. 3 Bob chooses a secret number b , and sends Alice $(g^b \bmod p)$. 4 Alice computes $((g^b \bmod p)^a \bmod p)$.

5 Bob computes $((g^a \bmod p)^b \bmod p)$.

Both Alice and Bob can use this number as their key. Notice that p and g need not be protected. 1 Alice and Bob agree on $p = 23$ and $g = 5$.

2 Alice chooses $a = 6$ and sends $5^6 \bmod 23 = 8$.

3 Bob chooses $b = 15$ and sends $5^{15} \bmod 23 = 19$.

4 Alice computes $19^6 \bmod 23 = 2$.

5 Bob computes $8^{15} \bmod 23 = 2$. Then 2 is the shared secret.

Clearly, much larger values of a , b , and p are required. An eavesdropper cannot discover this value even if she knows p and g and can obtain each of the messages.

Code:

```
import java.io.*;
```

```

import java.math.BigInteger; class Diffie
{
public static void main(String[] args) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter prime number:");
BigInteger p=new BigInteger(br.readLine());
System.out.print("Enter primitive root of "+p+":");
BigInteger g=new BigInteger(br.readLine());
System.out.println("Enter value for x less than "+p+":");
BigInteger x=new BigInteger(br.readLine());
BigInteger R1=g.modPow(x,p);
System.out.println("R1="+R1);
System.out.print("Enter value for y less than "+p+":");
BigInteger y=new BigInteger(br.readLine());
BigInteger R2=g.modPow(y,p);
System.out.println("R2="+R2);
BigInteger k1=R2.modPow(x,p);
System.out.println("Key calculated at Alice's side:"+k1);
BigInteger k2=R1.modPow(y,p);
System.out.println("Key calculated at Bob's side:"+k2);
System.out.println("diffie hellman secret key Encryption has Taken");
}
}

```

Output:

```

Enter prime number:
11
Enter primitive root of 11:7 Enter value for x less than 11: 3
R1=2
Enter value for y less than 11:6 R2=4
Key calculated at Alice's side:9
Key calculated at Bob's side:9
diffie hellman secret key Encryption has Taken

```

Conclusion:

This way we have implemented Diffie hellman key exchange algorithm. The Diffie–Hellman Key Exchange protocol allows two parties to securely exchange cryptographic keys over an unsecured channel, ensuring a shared secret without directly transmitting it.

Assignment 8 Viva Voce Questions

Viva Questions for Diffie hellman key exchange algorithm

- 1.What is the primary purpose of the Diffie–Hellman Key Exchange?
2. Discuss how the protocol enables secure key exchange over an insecure channel.
3. What are the public and private components in the Diffie–Hellman protocol?
4. Explain the roles of p , g , and the private keys.
5. Why is the Diffie–Hellman protocol considered secure?
6. Discuss the difficulty of solving the discrete logarithm problem.
7. What is a primitive root modulo p and why is it important in Diffie–Hellman?
8. Explain the significance of g and its selection.
9. Can the shared secret be transmitted directly in Diffie–Hellman? Why or why not?
10. What would happen if the private keys a and b are not kept secret?

Assignment 9

AIM: Implementation of SQL Injection Attack

Theory:

What is a SQL Injection Attack (SQLi)?

SQL Injection attacks (or SQLi) alter SQL queries, injecting malicious code by exploiting application vulnerabilities.

Successful SQLi attacks allow attackers to modify database information, access sensitive data, execute admin tasks on the database, and recover files from the system. In some cases attackers can issue commands to the underlying database operating system.

The severe impact of these attacks makes it critical for developers to adopt practices that prevent SQL injection, such as parameterized queries, stored procedures, and rigorous input validation.

Impact of a Successful SQL Injection Attack

The consequences of a successful SQL injection attack can include:

Stolen credentials—attackers can obtain credentials via SQLi and then impersonate users and use their privileges.

Unauthorized access to databases—attackers can gain access to the sensitive data in database servers.

Data alteration—attackers can alter or add new data to the accessed database.

Data deletion—attackers can delete database records or drop entire tables.

Lateral movement—attackers can access database servers with operating system privileges, and use these permissions to access other sensitive systems.

How Does a SQL Injection Attack Work?

A SQL Injection attack involves inserting or “injecting” a SQL query via the input data from the client to the application. A successful attack allows an attacker to manipulate the SQL queries that an application makes to its database. It typically involves the following steps:

Identification of vulnerable inputs: Attackers first identify inputs within the web application that are vulnerable to SQL injection. These inputs could be text fields in a form, URL parameters, or any other input mechanisms.

Crafting the malicious SQL query: Once a vulnerable input is identified, attackers craft a SQL statement intended to be inserted into the query executed by the application. This statement is designed to modify the original SQL query to perform actions unintended by the application developers.

Bypassing application security measures: Attackers often have to bypass security measures like input validation or escaping special characters. They achieve this through techniques like string concatenation or utilizing SQL syntax to comment out parts of the original query.

Executing the malicious query: When the application executes the SQL query, it includes the attacker's malicious input. This modified query can perform actions such as unauthorized viewing of data, deletion of data, or even database schema alterations.

Extracting or manipulating data: Depending on the attack, the outcome might be the extraction of sensitive information (like user credentials), altering existing data, adding new data, or even deleting significant portions of the database.

Exploiting database server vulnerabilities: Advanced SQL injections may exploit vulnerabilities in the database server, extending the attack beyond the database to the server level. This can include executing commands on the operating system or accessing other parts of the server's file system.

This process leverages the dynamic execution of SQL in applications where user inputs are directly included in SQL statements without proper validation or escaping. It exploits the way SQL queries are constructed, often in a way that the developers did not anticipate.

Real-Life SQL Injection Attack Examples

Over the past 20 years, many SQL injection attacks have targeted large websites, business and social media platforms. Some of these attacks led to serious data breaches. A few notable examples are listed below.

Breaches Enabled by SQL Injection

GhostShell attack—hackers from APT group Team GhostShell targeted 53 universities using SQL injection, stole and published 36,000 personal records belonging to students, faculty, and staff.

Turkish government—another APT group, RedHack collective, used SQL injection to breach the Turkish government website and erase debt to government agencies.

7-Eleven breach—a team of attackers used SQL injection to penetrate corporate systems at several companies, primarily the 7-Eleven retail chain, stealing 130 million credit card numbers.

HBGary breach—hackers related to the Anonymous activist group used SQL Injection to take down the IT security company's website. The attack was a response to HBGary CEO publicizing that he had names of Anonymous organization members.

Notable SQL Injection Vulnerabilities

Tesla vulnerability—in 2014, security researchers publicized that they were able to breach the website of Tesla using SQL injection, gain administrative privileges and steal user data.

Cisco vulnerability—in 2018, a SQL injection vulnerability was found in Cisco Prime License Manager. The vulnerability allowed attackers to gain shell access to systems on which the license manager was deployed. Cisco has patched the vulnerability.

Fortnite vulnerability—Fortnite is an online game with over 350 million users. In 2019, a SQL injection vulnerability was discovered which could let attackers access user accounts. The vulnerability was patched.

Types of SQL Injection Attacks

There are several types of SQL injection:

Union-based SQL Injection – Union-based SQL Injection represents the most popular type of SQL injection and uses the UNION statement. The UNION statement represents the combination of two select statements to retrieve data from the database.

Error Based SQL Injection – this method can only be run against MS-SQL Servers. In this attack, the malicious user causes an application to show an error. Usually, you ask the database a question and it returns an error message which also contains the data they asked for.

Blind SQL Injection – in this attack, no error messages are received from the database; We extract the data by submitting queries to the database. Blind SQL injections can be divided into boolean-based SQL Injection and time-based SQL Injection. Learn more in our guide to Blind SQL injection.

SQLi attacks can also be classified by the method they use to inject data:

SQL injection based on user input – web applications accept inputs through forms, which pass a user's input to the database for processing. If the web application accepts these inputs without sanitizing them, an attacker can inject malicious SQL statements.

SQL injection based on cookies – another approach to SQL injection is modifying cookies to “poison” database queries. Web applications often load cookies and use their data as part of database operations. A malicious user, or malware deployed on a user's device, could modify cookies, to inject SQL in an unexpected way.

SQL injection based on HTTP headers – server variables such HTTP headers can also be used for SQL injection. If a web application accepts inputs from HTTP headers, fake headers containing arbitrary SQL can inject code into the database.

Second-order SQL injection – these are possibly the most complex SQL injection attacks, because they may lie dormant for a long period of time. A second-order SQL injection attack delivers poisoned data, which might be considered benign in one context, but is malicious in another context. Even if developers sanitize all application inputs, they could still be vulnerable to this type of attack.

SQL Injection Code Examples

Let's look at two common examples of SQL injection attacks.

Example 1: Using SQLi to Authenticate as Administrator

This example shows how an attacker can use SQL injection to circumvent an application's authentication and gain administrator privileges.

Consider a simple authentication system using a database table with usernames and passwords. A user's POST request will provide the variables user and pass, and these are inserted into a SQL statement:

```
sql = "SELECT id FROM users WHERE username='" + user + "' AND password='" + pass + "'"
```

The problem here is that the SQL statement uses concatenation to combine data. The attacker can provide a string like this instead of the pass variable:

password' OR 5=5

The resulting SQL query will be run against the database:

```
SELECT id FROM users WHERE username='user' AND password='pass' OR 5=5'
```

Because 5=5 is a condition that always evaluates to true, the entire WHERE statement will be true, regardless of the username or password provided.

The WHERE statement will return the first ID from the users table, which is commonly the administrator. This means the attacker can access the application without authentication, and also has administrator privileges.

A more advanced form of this attack is where the attacker adds a code comment symbol at the end of the SQL statement, allowing them to further manipulate the SQL query. The following will work in most databases including MySQL, PostgreSQL, and Oracle:

```
' OR '5'='5' /*
```

Learn more in our detailed guide to sql injection test.

Example 2: Using SQLi to Access Sensitive Data

In this example, the following code obtains the current username, and searches for items matching a certain item name, where the owner is the current user.

```
...
string userName = ctx.GetAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '"
    + userName + "' AND itemname = '"
    + ItemName.Text + "'";
...
```

This code has the same weakness as in the previous example – the use of concatenation. After combining the username and item name, the code creates the following query:

```
SELECT * FROM items
WHERE owner =
AND itemname = ;
```

If the attacker provides the following string for itemname:

Widget' OR 5=5

The SQL statement becomes:

```
SELECT * FROM items
```

```
WHERE owner = 'John'
```

```
AND itemname = 'Widget' OR 5=5';
```

Which is the same as: `SELECT * FROM items;`

This means the query will return the data of the entire table, giving the attacker unauthorized access to sensitive data.

Conclusion: SQL Injection is a critical security vulnerability that allows attackers to manipulate database queries by injecting malicious SQL code. It can lead to unauthorized access, data leakage, and even complete database compromise. Preventing SQL injection requires careful input validation, parameterized queries, and secure coding practices.

Assignment 9 Viva Voce Questions

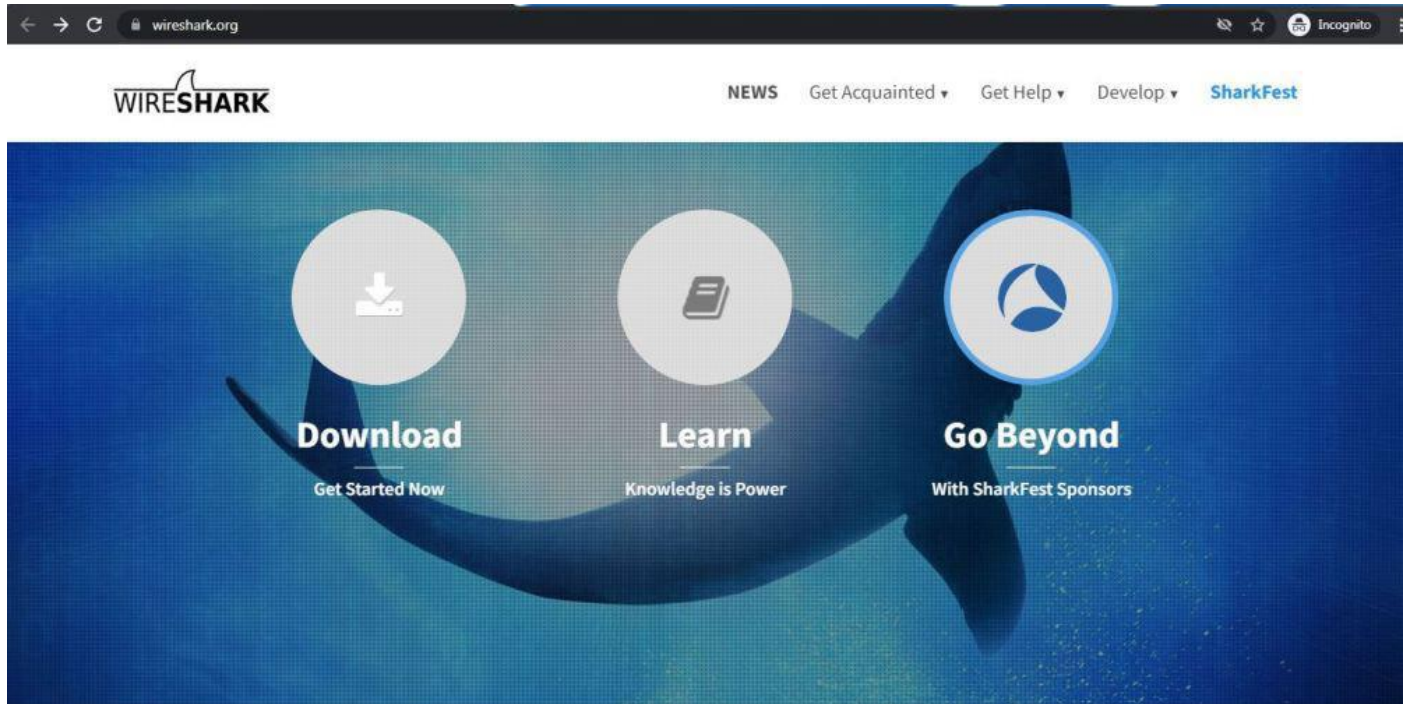
Viva Questions for SQL Injection attacks

1. What are the common types of SQL Injection attacks?
2. Discuss types like Classic SQL Injection, Blind SQL Injection, and Union-based SQL Injection.
How does the example query `SELECT * FROM users WHERE username = 'admin' AND password = " OR '1'='1';` exploit a SQL Injection vulnerability?
3. Analyze the query and explain how it bypasses authentication.
4. What is the role of user input sanitization in preventing SQL Injection?
5. Discuss how sanitizing inputs helps avoid injecting malicious SQL.
6. What are parameterized queries, and how do they prevent SQL Injection?
7. Explain the concept and provide examples of using prepared statements.
8. Why is error handling important in preventing SQL Injection?
9. Discuss how exposing detailed errors can help attackers craft more effective SQL Injection attacks.
10. What is the difference between authentication bypass and data exfiltration in SQL Injection?

Assignment 10

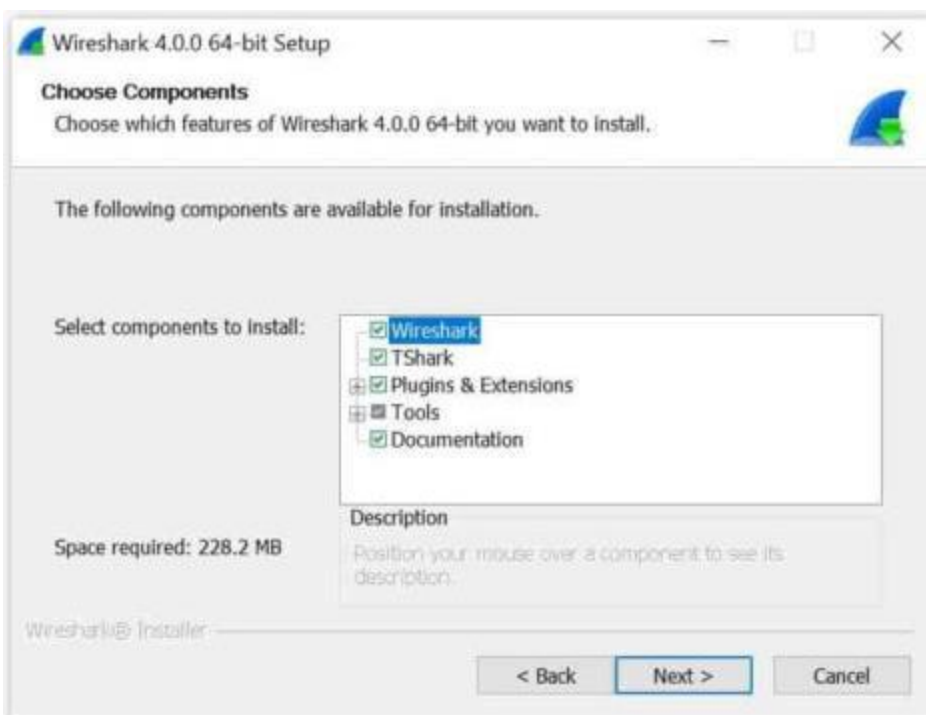
AIM: Implementation of Virtual Lab Experiment

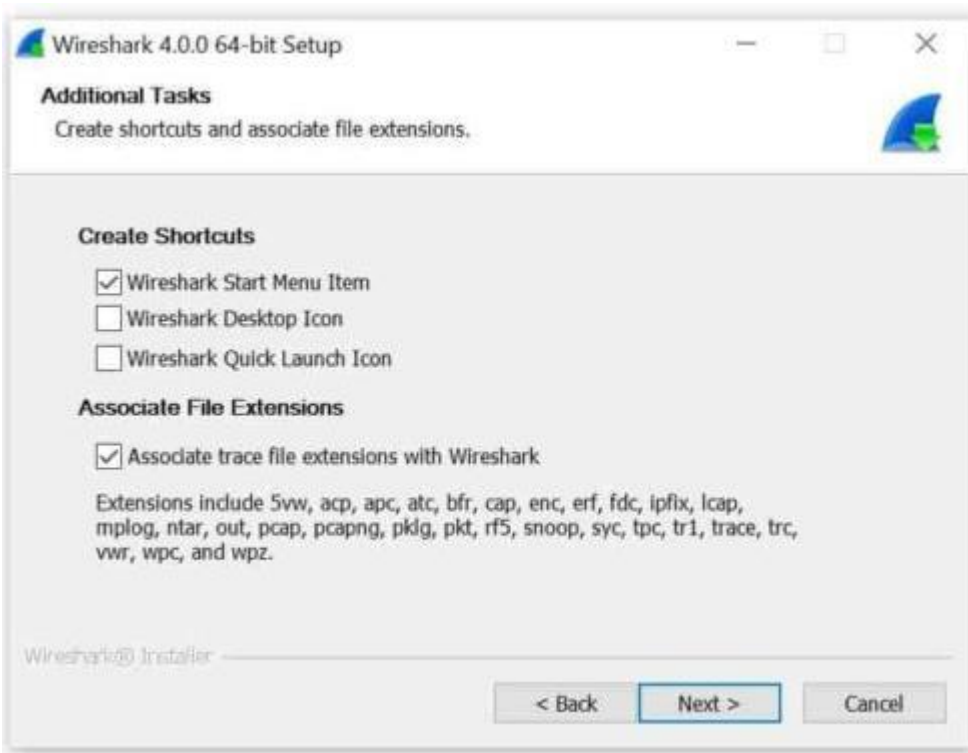
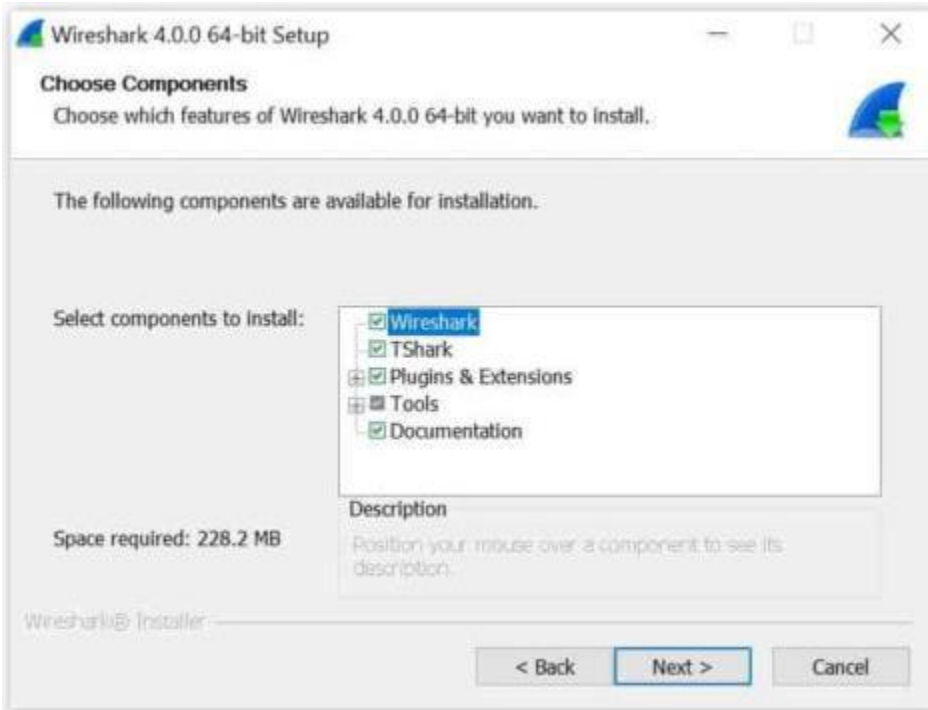
Content Beyond Syllabus: Packet Tracer

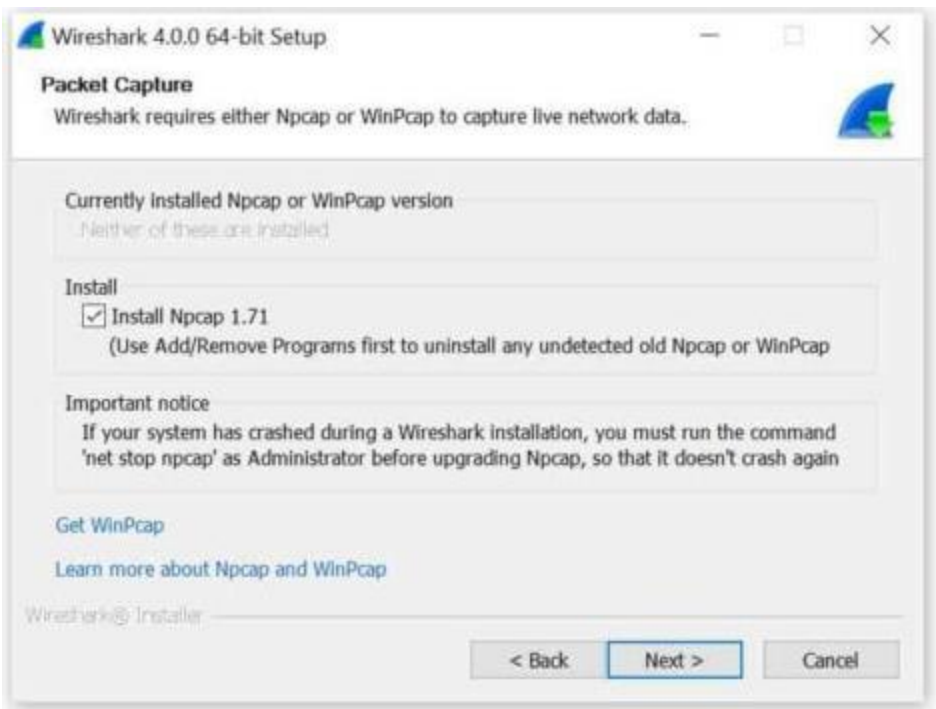
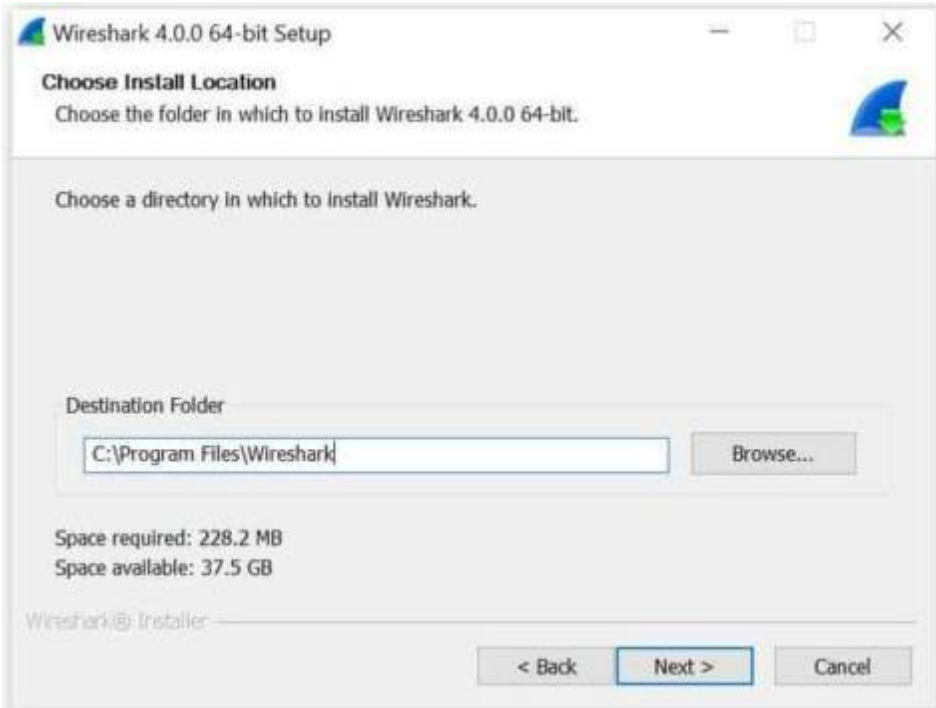


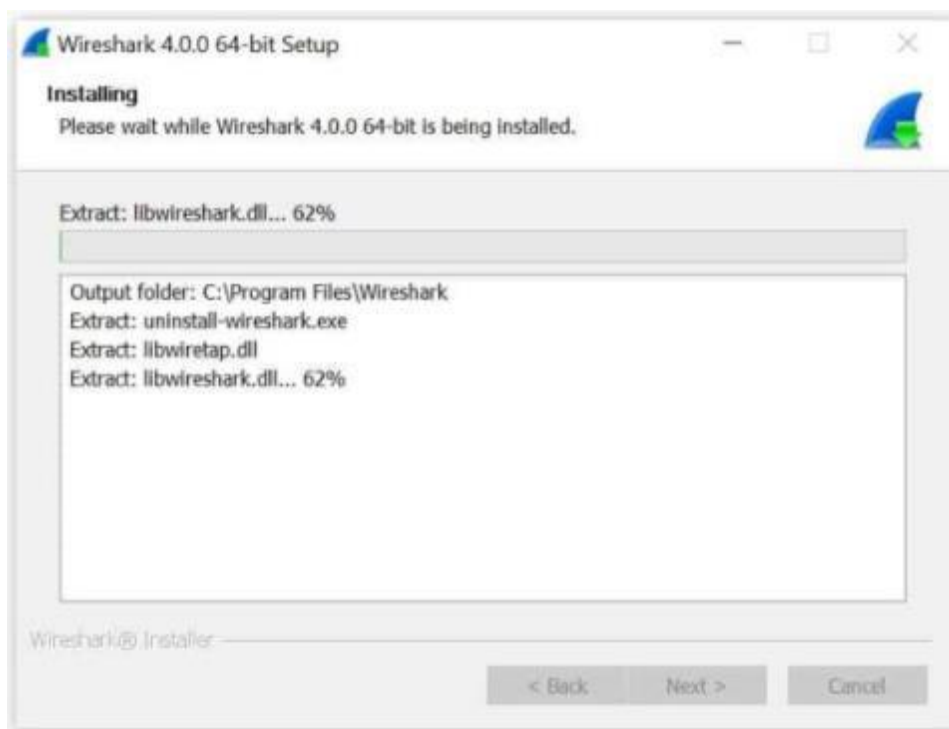
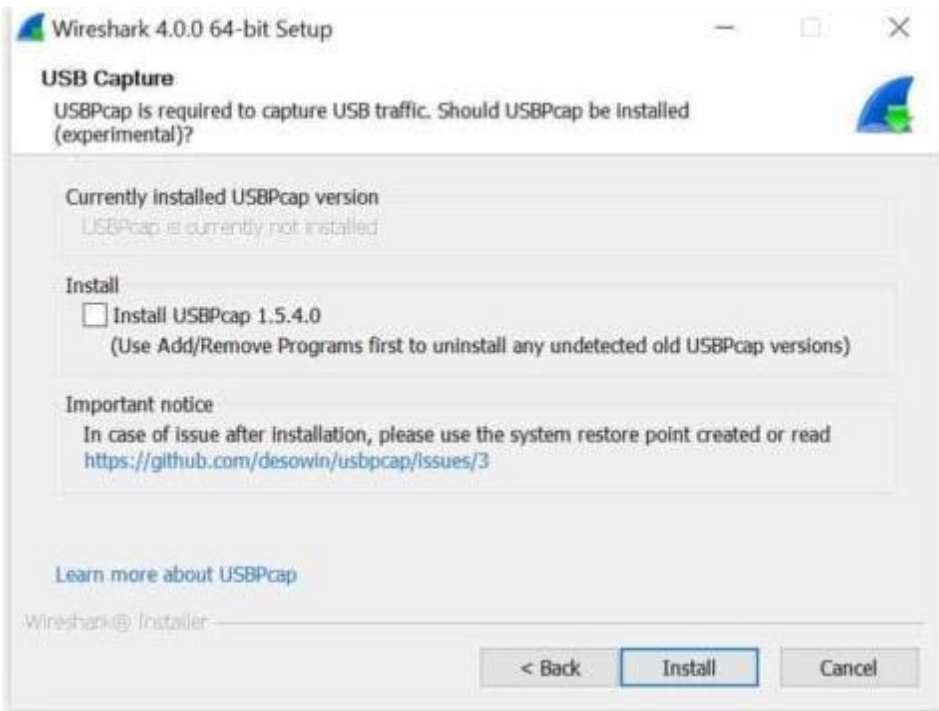


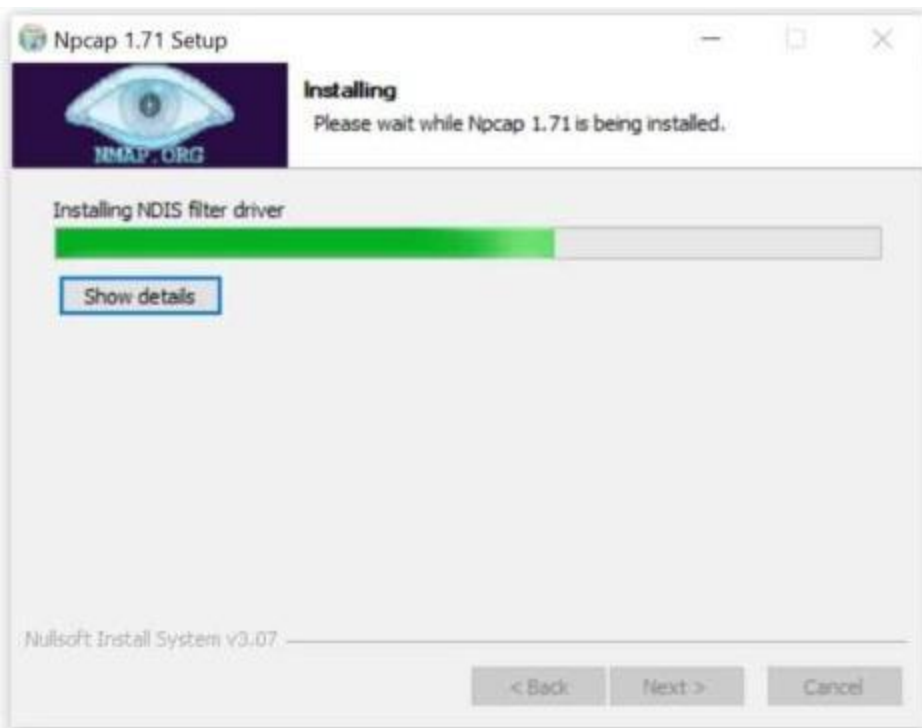
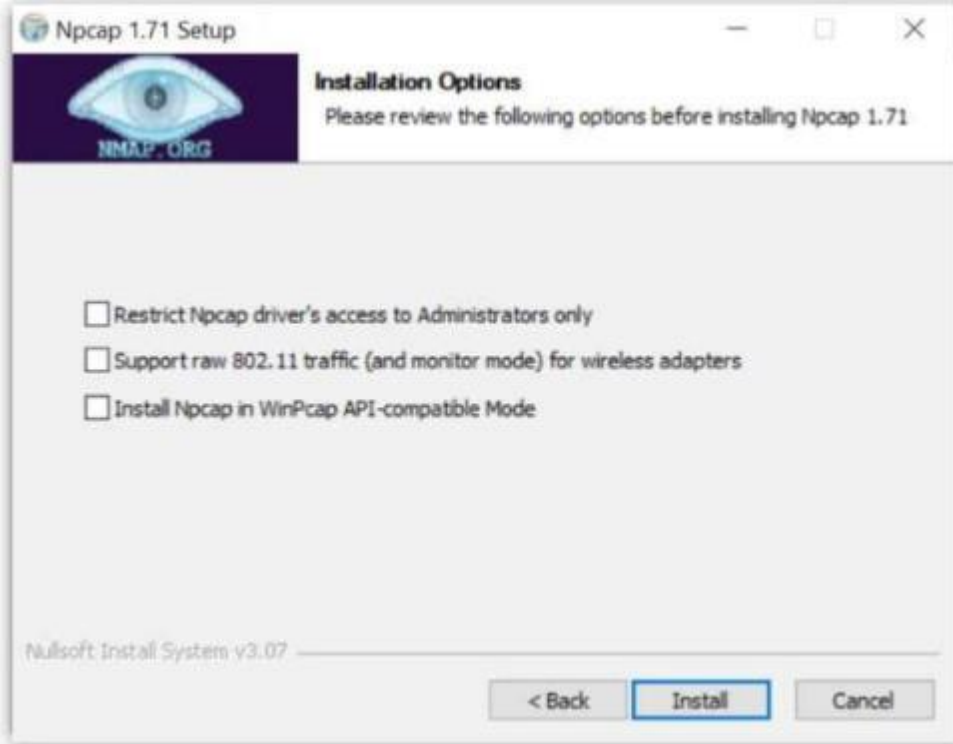
Wireshark-win64-4....exe

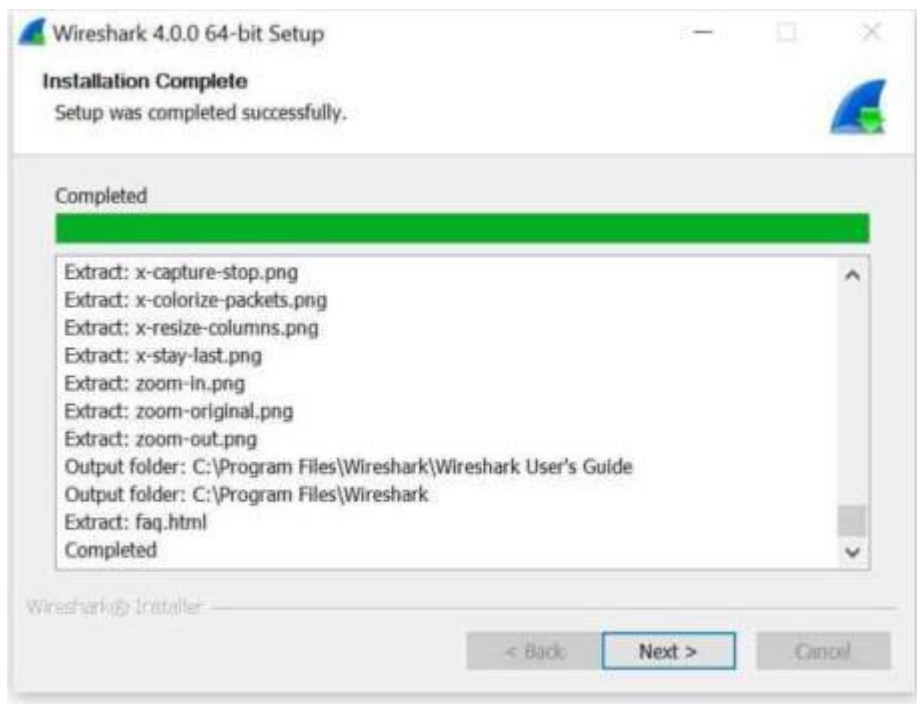


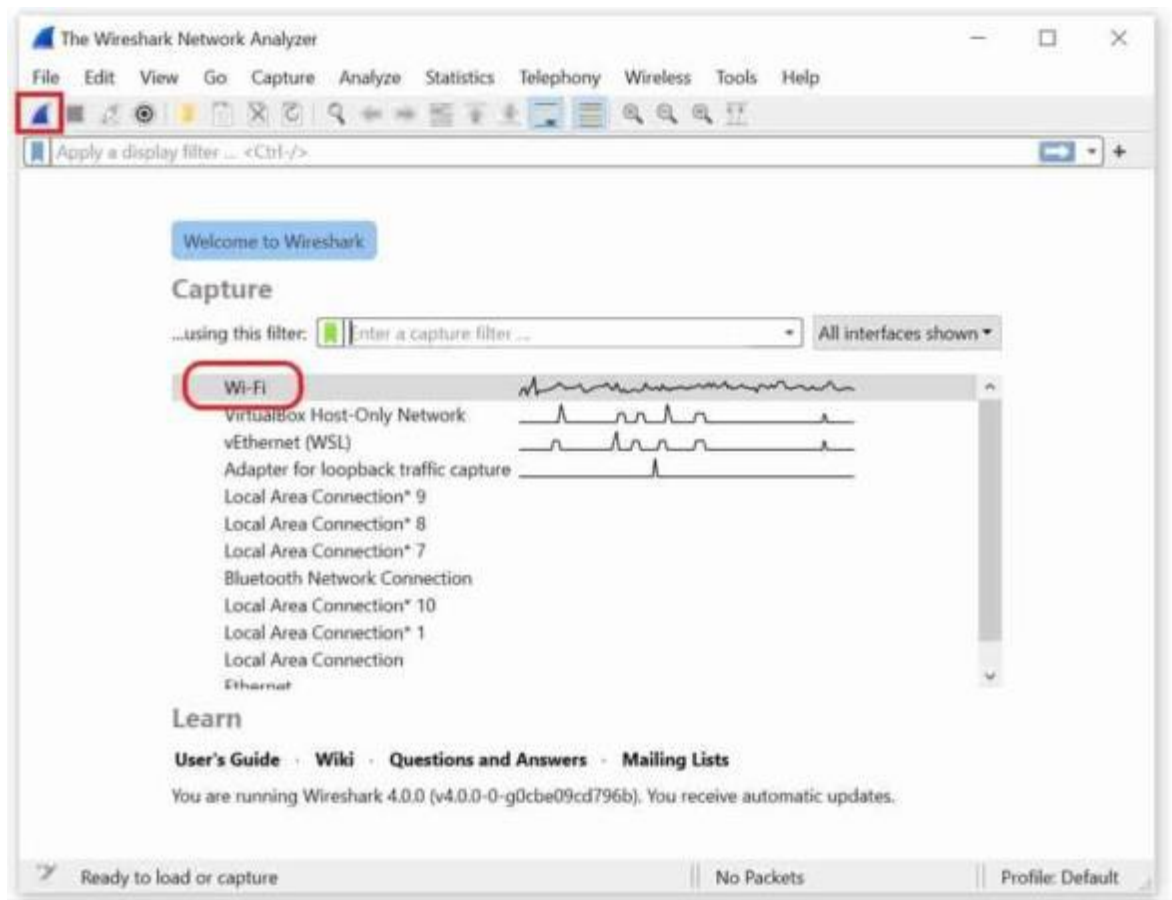
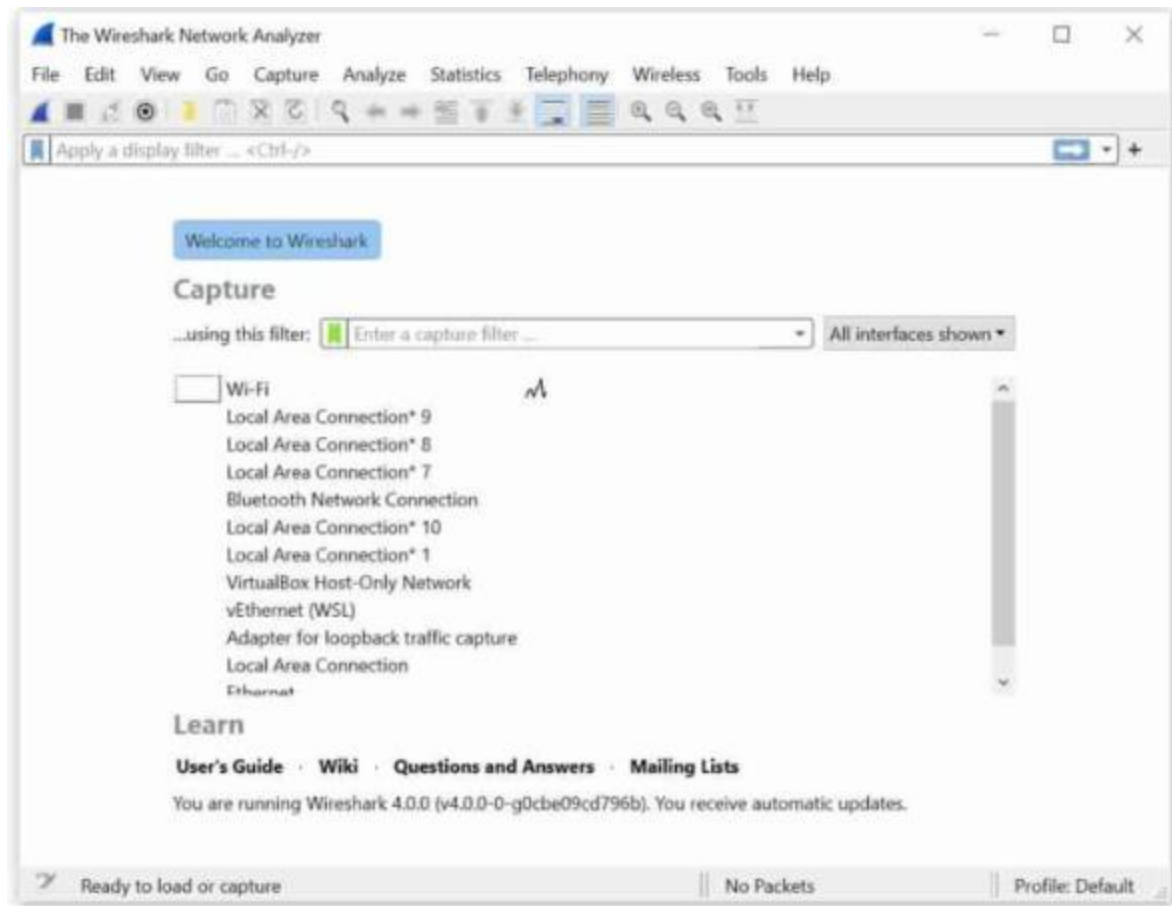












Capturing from Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter: <Ctrl-J>

No.	Time	Source	Destination	Protocol	Length	Info
555	6.496336	192.168.0.103	172.217.166.106	TLSv1.2	282	Applicati
556	6.497688	142.250.195.74	192.168.0.103	TCP	54	443 → 542
557	6.502495	31.47.196.154	192.168.0.103	TCP	1494	443 → 505
558	6.502546	192.168.0.103	31.47.196.154	TCP	54	50510 → 4
559	6.502866	31.47.196.154	192.168.0.103	TCP	1494	443 → 505
560	6.503195	31.47.196.154	192.168.0.103	TCP	1494	443 → 505
561	6.503223	192.168.0.103	31.47.196.154	TCP	54	50510 → 4
562	6.504245	172.217.166.106	192.168.0.103	TCP	54	443 → 543
563	6.577437	31.47.196.154	192.168.0.103	SSLv2	1494	Encrypted

> Frame 1: 1494 bytes on wire (11952 bits),
 > Ethernet II, Src: Tp-LinkT_49:44:7a (c4:6e
 > Internet Protocol Version 4, Src: 31.47.19
 > Transmission Control Protocol, Src Port: 4
 Transport Layer Security

0000 40 5b d8 32 31 f9 c4 6e 1f 49 44 7a ~
 0010 05 c8 fe 15 40 00 32 06 a0 21 1f 2f
 0020 00 67 01 bb c5 4e ae fd 3a 6a 6d 4c
 0030 01 f5 b5 29 00 00 0f d7 e5 49 f2 b8
 0040 77 9b d1 06 88 54 87 81 a3 d4 21 bc
 0050 c7 99 cf 68 60 c9 84 d8 e9 81 f6 54
 0060 a3 f9 ca f5 1a e3 0b d6 50 c4 3d a1
 0070 79 e7 57 3c e6 84 34 c5 c5 d7 03 fa
 0080 20 94 63 83 65 53 53 5e 52 26 f6 8b
 0090 a3 38 b9 64 e5 51 f6 35 86 a8 d9 27 ~

Wi-Fi: <live capture in progress> | Packets: 563 - Displayed: 563 (100.0%) | Profile: Default