D.Y. Patil College of Engineering, Akurdi, Pune – 44

Department of Electronics And Telecommunication

Submission Date :_____

Name of the student:_____

Roll No:_____

Experiment No. 01

Insecurity of different types of passwords

Aim : Design and implement for the insecurity of default passwords, printed passwords and password transmitted in plain text.

Software Requirement:

C / C++ / Java/Python or equivalent compiler

Theory:

In today's growing world of Internet & technology, its very critical, important & mandatory to have strong password while creating any user account on various internet platforms such as websites, applications, interfaces etc. *Strong Password* is reasonably difficult to guess in a short period of time either through human guessing or the use of specialized software.

The following are general recommendations for creating a Strong Password:

A Strong Password **should** - ○ Be at least 8 characters in length ○ Contain both upper and lowercase alphabetic characters (e.g. A-Z, a-z) ○ Have at least one numerical character (e.g. 0-9) ○ Have at least one special character (e.g. ~!@#$%^&*()_-+=)

A Strong Passwords **do not** - ○ Spell a word or series of words that can be found in a standard dictionary ○ Spell a word with a number added to the beginning and the end ○ Be based on any personal information such as user id, family name, pet, birthday, etc.

The following are several recommendations for maintaining a Strong Password:

Do not share your password with anyone for any reason

Passwords should not be shared with anyone. In situations where someone requires access to another individual's protected resources, delegation of permission options should be explored. For example, Microsoft Exchange calendar will allow a user to delegate control of his or her calendar to another user without sharing any passwords. This type of solution is encouraged. Passwords should not be shared even for the purpose of computer repair. An alternative to doing this is to create a new account with an appropriate level of access for the repair person.

Change your password upon indication of compromise

If you suspect someone has compromised your account, change your password immediately. Be sure to change your password from a computer you do not typically.

Consider using a passphrase instead of a password

A passphrase is a password made up of a sequence of words with numeric and/or symbolic characters inserted throughout. A passphrase could be a lyric from a song or a favorite quote. Passphrases typically have additional benefits such as being longer and easier to remember. For example, the passphrase "My passw0rd is $uper str0ng!" is 28 characters long and includes alphabetic, numeric and special characters. It is also relatively easy to remember. It is important to note the placement of numeric and symbolic characters in this example as they prevent multiple words from being found in a standard dictionary. The use of blank spaces also makes a password more difficult to guess.

Do not write your password down or store it in an insecure manner

As a general rule, you should avoid writing down your password. In cases where it is necessary to write down a password, that password should be stored in a secure location and properly destroyed when no longer needed (see Guidelines for Data Protection). Using a password manager to store your passwords is not recommended unless the password manager leverages strong encryption and requires authentication prior to use. The ISO has vetted some password managers that meets these requirements.

Avoid reusing a password

When changing an account password, you should avoid reusing a previous password. If a user account was previously compromised, either knowingly or unknowingly, reusing a password could allow that user account to, once again, become compromised. Similarly, if a password was shared for some reason, reusing that password could allow someone unauthorized access to your account.

Avoid using the same password for multiple accounts

While using the same password for multiple accounts makes it easier to remember your passwords, it can also have a chain effect allowing an attacker to gain unauthorized access to multiple systems. This is particularly important when dealing with more sensitive accounts such as your Andrew account or your online banking account. These passwords should differ from the password you use for instant messaging, webmail and other web-based accounts.

Do not use automatic logon functionality

Using automatic logon functionality negates much of the value of using a password. If a malicious user is able to gain physical access to a system that has automatic logon configured, he or she will be able to take control of the system and access potentially sensitive information.

The following are Guidelines for individuals responsible for provisioning and support of user accounts:

Enforce strong passwords

Many systems and applications include functionality that prevents a user from setting a password that does not meet certain criteria. Functionality such as this should be leveraged to ensure only Strong Passwords are being set.

Require a change of initial or "first-time" passwords

Forcing a user to change their initial password helps ensure that only that user knows his or her password. Depending on what process is being used to create and distribute the password to the user, this practice can also help mitigate the risk of the initial password being guessed or intercepted during transmission to the user. This guidance also applies to situations where a password must be manually reset.

Force expiration of initial or "first-time" passwords

In certain situations, a user may be issued a new account and not access that account for a period of time. As mentioned previously, initial passwords have a higher risk of being guessed or intercepted depending on what process is being used to create and distribute passwords. Forcing an initial password to expire after a period of time (e.g. 72 hours) helps mitigate this risk. This may also be a sign that the account is not necessary.

Do not use Restricted data for initial or "first-time" passwords

The Guidelines for Data Classification defines Restricted data in its data classification scheme. Restricted data includes, but is not limited to, social security number, name, date of birth, etc. This type of data should not be used wholly or in part to formulate an initial password. See Appendix A for a more comprehensive list of data types.

Always verify a user's identity before resetting a password

A user's identity should always be validated prior to resetting a password. If the request is inperson, photo identification is a sufficient means of doing this. If the request is by phone, validating an identity is much more difficult. One method of doing this is to request a video conference with the user (e.g. Skype) to match the individual with their photo id. However, this can be a cumbersome process. Another option is to have the person's manager call and confirm the request. For obvious reasons, this would not work for student requests. If available, a self-service password reset solution that prompts a user with a series of customized questions is an effective approach to addressing password resets.

Never ask for a user's password

As stated above, individual user account passwords should not be shared or any reason. A natural correlation to this guidance is to never ask others for their passwords. Once again,

delegation of permission is one alternative to asking a user for their password. Some applications include functionality that allows an administrator to impersonate another user, without entering that user's password, while still tying actions back to the administrator's user account. This is also an acceptable alternative. In computer repair situations, requesting that a user create a temporarily account on their system is one alternative.

The following are several additional Guidelines for individuals responsible for the design and implementation of systems and applications:

Change default account passwords

Default accounts are often the source of unauthorized access by a malicious user. When possible, they should be disabled completely. If the account cannot be disabled, the default passwords should be changed immediately upon installation and configuration of the system or application.

Implement strict controls for system-level and shared service account passwords Shared service accounts typically provide an elevated level of access to a system. Systemlevel accounts, such as root and Administrator, provide complete control over a system. This makes these types of accounts highly susceptible to malicious activity. As a result, a more lengthy and complex password should be implemented. System-level and shared service accounts are typically critical to the operation of a system or application. Because of this, these passwords are often known by more than one administrator. Passwords should be changed anytime someone with knowledge of the password changes job responsibilities or terminates employment. Use of accounts such as root and Administrator should also be limited as much as possible. Alternatives should be explored such as using Psudo in place of root and creating unique accounts for Windows administration instead of using default accounts.

Do not use the same password for multiple administrator accounts

Using the same password for multiple accounts can simplify administration of systems and applications. However, this practice can also have a chain effect allowing an attacker to break into multiple systems as a result of compromising a single account password.

Do not allow passwords to be transmitted in plain-text

Passwords transmitted in plain-text can be easily intercepted by someone with malicious intent. Protocols such as FTP, HTTP, SMTP and Telnet all natively transmit data (including your password) in plain-text. Secure alternatives include transmitting passwords via an encrypted tunnel (e.g. IPSec, SSH or SSL), using a one-way hash or implementing a ticket based authentication scheme such as Kerberos.

Do not store passwords in easily reversible form

Passwords should not be stored or transmitted using weak encryption or hashing algorithms. For example, the DES encryption algorithm and the MD-4 hash algorithm both have known security weaknesses that could allow protected data to be deciphered. Encryption algorithms such as 3DES or AES and hashing algorithms such as SHA-1 or SHA-256 are stronger alternatives to the previously mentioned algorithms.

Implement automated notification of a password change or reset

When a password is changed or reset, an email should be automatically sent to the owner of that user account. This provides a user with a confirmation that the change or reset was successful and also alerts a user if his or her password to unknowingly changed or reset.

If above guideline not followed for creating the password of any account, then security of that password will be getting compromised.

Problem Statement:

Write a Program which implements Insecurity of Default & Printed Passwords. A) Create user ID & Password. Store them.

B) Get user ID & password from user, verify them.

C) Comment on insecurity associated with them in conclusion

Program (attach):

```
import java.util.*;
class HelloWorld {
public static void main(String[] args) {
   Scanner sc= new Scanner(System.in);
   System.out.print("Create user id: ");
   String userId=sc.nextLine();
   System.out.print("Create password:");
   String password=sc.nextLine();


   System.out.print("Enter your user id ");
   String checkuserId=sc.nextLine();
   System.out.print("Enter your password: ");
   String checkpassword=sc.nextLine();


  if((userId.equals(checkuserId)) && (password.equals(checkpassword))){
   System.out.println("your entered userid and password are correct");
```

```
    }
    else{
        System.out.println("your entered userid and password is incorrect");
    }
  }
}
```

Output:

```
java -cp /tmp/TEOgbMQJ9p HelloWorld
Create user id: DYP@gmail.com
Create password:12345
Enter your user id DYP@gmail.com
Enter your password: 12345
your entered userid and password are correct
```

Conclusion:

Submission Date :_____

Name of the student:_____

Roll No:_____

Experiment No. 02

Encryption & Decryption using Rail Fence , Row & Column Transformation Technique

Aim : Implementation of Rail Fence & Row-Column Transformation Technique for Encryption & Decryption of Plain Text

Software Requirement:

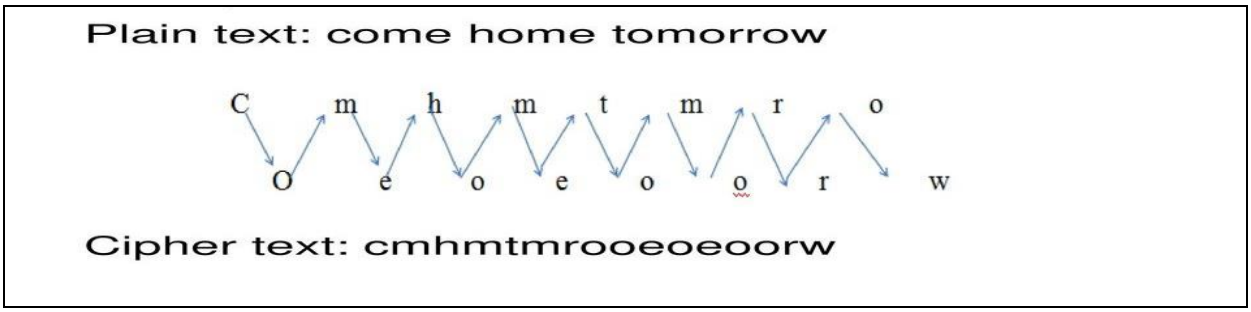
C / C++ / Java/Python or equivalent compiler

Theory:

A) Rail Fence Cipher

In the rail fence cipher, the plain text is written downwards and diagonally on successive "rails" of an imaginary fence, then moving up when we reach the bottom rail. When we reach the top rail, the message is written downwards again until the whole plaintext is written out. The message is then read off in rows.

EXAMPLE:



B) Row-Columnar Transposition Cipher:

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 .

5. Finally, the message is read off in columns, in the order specified by the keyword

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.

EXAMPLE:

A	U	T	H	O	R
1	6	5	2	3	4
W	E	A	R	E	D
I	S	C	O	V	E
R	E	D	S	A	V
E	Y	O	U	R	S
E	L	F	A	B	C

yields the cipher

W I R E E R O S U A E V A R B D E V S C A C D O F E S E Y L .

ALGORITHM:

STEP-1: Read the Plain text.

STEP-2: Arrange the plain text in row columnar matrix format.

STEP-3: Now read the keyword depending on the number of columns of the plain text.

STEP-4: Arrange the characters of the keyword in sorted order and the corresponding columns of the plain text.

STEP-5: Read the characters row wise or column wise in the former order to get the cipher text.

Problem Statement:

A) Write a Program to implement Rail Fence Cipher for Encryption & Decryption of Plain Text
Plain text: SEND ME ONE THOUSAND RUPEE

Depth: 02 & 03

B) Write a Program to implement Row & Column Transformation Technique for Encryption & Decryption of Plain Text

Plain text: SEND ME ONE THOUSAND RUPEE

Keyword: ZEBRA

Program:

A) Rail Fence Cipher for Encryption & Decryption:-

```
import java.util.Arrays;
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;
                // place the marker
                if (rail[row][col] != '*')
                        result.append(rail[row][col++]);
                // find the next row using direction flag
                if (dirDown)
                        row++;
                else
                        row--;
        }
        return result.toString();
}
// driver program to check the above functions
public static void main(String[] args)
{
        // Encryption
        System.out.println("Encrypted Message: ");
        System.out.println(encryptRailFence("SEND ME ONE THOUSAND RUPEE", 2));
        System.out.println(encryptRailFence("SEND ME ONE THOUSAND RUPEE", 3));
        // Now decryption of the same cipher-text
```

```
		System.out.println("\nDecrypted Message: ");
		System.out.println(decryptRailFence("SN EOETOSN UEEDM N HUADRPE", 2));
		System.out.println(decryptRailFence("S OTS EEDM N HUADRPENEEONU", 3));
	}
}
// This code is contributed by Jay
```

B) Row and Column Transformation Encrption and Decryption:-

```
import java.io.*;
import java.util.*;
// Class
// For transposition cipher
public class GFG {
	public static String selectedKey;
	public static char sortedKey[];
	public static int sortedKeyPos[];
	// Constructor 1 of this class
	// Default constructor defining the default key
	public GFG()
	{
		selectedKey = "zebra";
		sortedKeyPos = new int[selectedKey.length()];
		sortedKey = selectedKey.toCharArray();
	}
	// Constructor 2 of this class
	// Parameterized constructor defining the custom key
	public GFG(String GeeksForGeeks)
	{
		selectedKey = "sendmeonethousandrupee";
		sortedKeyPos = new int[selectedKey.length()];
		sortedKey = selectedKey.toCharArray();
	}
  // Method 1 - doProcessOnKey()
	// To reorder data do the sorting on selected key
	public static void doProcessOnKey()
	{
		// Find position of each character in selected key
		// and arranging it in alphabetical order
		int min, i, j;
		char orginalKey[] = selectedKey.toCharArray();
		char temp;
		// Step 1: Sorting the array of selected key
		// using nested for loops
		for (i = 0; i < selectedKey.length(); i++) {
			min = i;
			for (j = i; j < selectedKey.length(); j++) {
				if (sortedKey[min] > sortedKey[j]) {
					min = j;
				}
			}
			if (min != i) {
				temp = sortedKey[i];
				sortedKey[i] = sortedKey[min];
```

```
                        sortedKey[min] = temp;
                }
        }
        // Step 2: Filling the position of array
        // according to alphabetical order
        // using nested for loops
        for (i = 0; i < selectedKey.length(); i++) {
                for (j = 0; j < selectedKey.length(); j++) {
                        if (orginalKey[i] == sortedKey[j])
                                sortedKeyPos[i] = j;
                }
        }
}
// Method 2 - doEncryption()
// To encrypt the targeted string
public static String doEncryption(String plainText)
{
        int min, i, j;
        char orginalKey[] = selectedKey.toCharArray();
        char temp;
        doProcessOnKey();
        // Step 3: Generating the encrypted message by
        // doing encryption using Transpotion Cipher
        int row = plainText.length() / selectedKey.length();
        int extrabit= plainText.length() % selectedKey.length();
        int exrow = (extrabit == 0) ? 0 : 1;
        int rowtemp = -1, coltemp = -1;
        int totallen = (row + exrow) * selectedKey.length();
        char pmat[][] = new char[(row + exrow)][(selectedKey.length())];
        char encry[] = new char[totallen];
        int tempcnt = -1;
        row = 0;
        for (i = 0; i < totallen; i++) {
                coltemp++;
                if (i < plainText.length()) {
                        if (coltemp == (selectedKey.length())) {
                                row++;
                                coltemp = 0;
                        }
                        pmat[row][coltemp] = plainText.charAt(i);
                }
                else {
                        // Padding can be added between two
                        // consecutive alphabets or a group of
                        // alphabets of the resultant cipher text
                        pmat[row][coltemp] = '-';
                }
        }
        int len = -1, k;
        for (i = 0; i < selectedKey.length(); i++) {
                for (k = 0; k < selectedKey.length(); k++) {
                        if (i == sortedKeyPos[k]) {
```

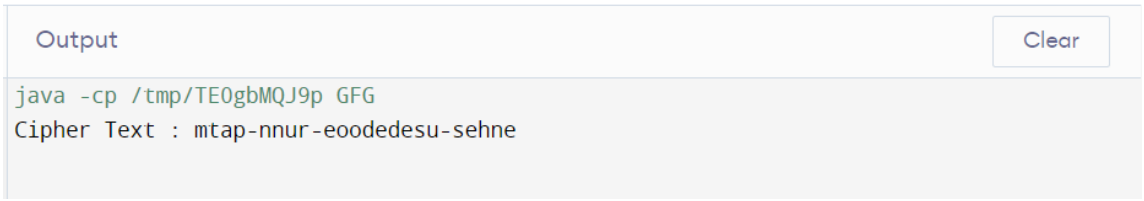
```
                                        break;
                                }
                        }
                        for (j = 0; j <= row; j++) {
                                len++;
                                encry[len] = pmat[j][k];
                        }
                }
                String p1 = new String(encry);
                return (new String(p1));
        }
        // Method 3 - doEncryption()
        // To decrypt the targeted string
        public static String doDecryption(String s)
        {
                int min, i, j, k;
                char key[] = selectedKey.toCharArray();
                char encry[] = s.toCharArray();
                char temp;
                doProcessOnKey();
                // Step 4: Generating a plain message
                int row = s.length();
                selectedKey.length();
                char pmat[][]= new char[row][(selectedKey.length())];
                int tempcnt = -1;
                for (i = 0; i < selectedKey.length(); i++) {
                        for (k = 0; k < selectedKey.length(); k++) {
                                if (i == sortedKeyPos[k]) {
                                        break;
                                }
                        }
                        for (j = 0; j < row; j++) {
                                tempcnt++;
                                pmat[j][k] = encry[tempcnt];
                        }
                }
                // Step 5: Storing matrix character in
                // to a single string
                char p1[] = new char[row * selectedKey.length()];
                k = 0;
                for (i = 0; i < row; i++) {
                        for (j = 0; j < selectedKey.length(); j++) {
                                if (pmat[i][j] != '*') {
                                        p1[k++] = pmat[i][j];
                                }
                        }
                }
                p1[k++] = '\0';
                return (new String(p1));
        }
        // Method 4 - main()
        // Main driver method
```

```
        public static void main(String[] args)
        {
                // Creating object of class in main method
                GFG tc = new GFG();
                // Printing the ciphere text
                // Custom input - Hello Geek
                System.out.println("Cipher Text : "          + tc.doEncryption("sendmeonethousandrupee"));
        }
}
```

Output:

A). Rail Fence Cipher for Encryption & Decryption:-

B). Row and Column Transformation Encrption and Decryption:-



Conclusion:

D.Y. Patil College of Engineering, Akurdi, Pune – 44

Department Of Electronics And Telecommunication

Submission Date :_____

Name of the student:_____

Roll No: _____

Experiment No. 03

Implementation of Ceaser Cipher for encryption and decryption

Aim : To implement a program for encrypting a plain text and decrypting a cipher text using Caesar Cipher (shift cipher) substitution technique

Software Requirement:

C / C++ / Java/Python or equivalent compiler

Theory:

It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on.

The method is named after Julius Caesar, who used it in his private correspondence.

The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions.

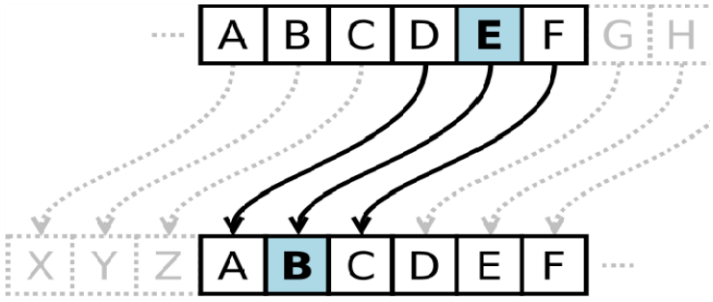
The encryption can also be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1, Z = 25.

Encryption of a letter x by a shift n can be described mathematically as, $E_n(x)$

$$= (x + n) \bmod 26$$

Decryption is performed similarly,

$$D_n(x) = (x - n) \bmod 26$$



ALGORITHM:

STEP-1: Read the plain text from the user.

STEP-2: Read the key value from the user.

STEP-3: If the key is positive then encrypt the text by adding the character in the plain text.

STEP-4: Else subtract the key from the plain text.

STEP-5: Display the cipher text obtained above.

Problem Statement: Write programs to perform encryption and decryption using: Ceaser Cipher

Plain Text: SEND ME ONE THOUSAND RUPEE

Program:

```
import java.util.Scanner;
public class New {
	public static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";
	public static String encryptData(String inputStr, int shiftKey)
	{
	 inputStr = inputStr.toLowerCase(); // convert inputStr into lower case
	   String encryptStr = ""; // encryptStr to store encrypted data
	   for (int i = 0; i < inputStr.length(); i++)  // use for loop for traversing each character of the input string
	   {
	      int pos = ALPHABET.indexOf(inputStr.charAt(i)); // get position of each character of inputStr in ALPHABET
	      int encryptPos = (shiftKey + pos) % 26;   // get encrypted char for each char of inputStr
	      char encryptChar = ALPHABET.charAt(encryptPos);
	      encryptStr += encryptChar; // add encrypted char to encrypted string
	   }
	   return encryptStr; // return encrypted string
	}

	// create decryptData() method for decrypting user input string with given shift key
	public static String decryptData(String inputStr, int shiftKey)
	{
	 inputStr = inputStr.toLowerCase();  // convert inputStr into lower case
	   String decryptStr = "";// decryptStr to store decrypted data
	   for (int i = 0; i < inputStr.length(); i++) // use for loop for traversing each character of the input string
	   {
	      int pos = ALPHABET.indexOf(inputStr.charAt(i));// get position of each character of inputStr in ALPHABET
	      int decryptPos = (pos - shiftKey) % 26;// get decrypted char for each char of inputStr
	      if (decryptPos < 0){ // if decryptPos is negative
	         decryptPos = ALPHABET.length() + decryptPos;
	      }
	      char decryptChar = ALPHABET.charAt(decryptPos);
	      decryptStr += decryptChar;// add decrypted char to decrypted string
	   }
	   return decryptStr; // return decrypted string
	}

	// main() method start
	public static void main(String[] args)
	{
	   // create an instance of Scanner class
	   Scanner sc = new Scanner(System.in);

	   // take input from the user
	   System.out.println("Enter a string for encryption using Caesar Cipher: ");
```

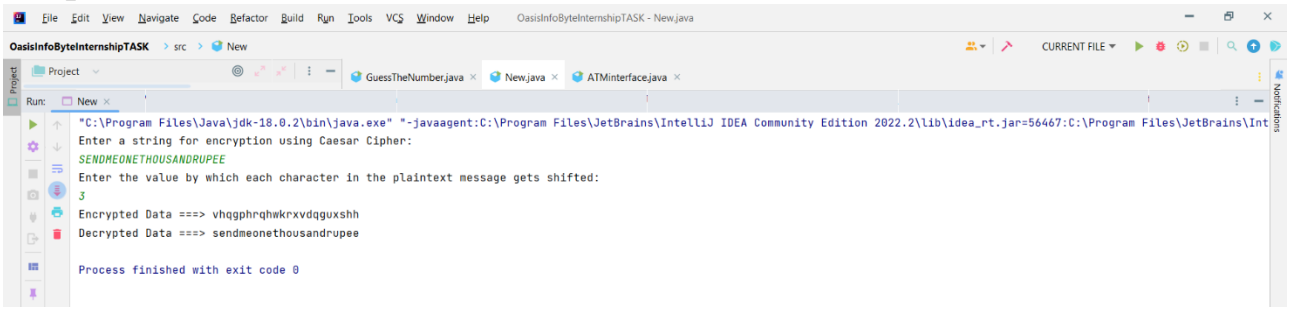
```
        String inputStr = sc.nextLine();

        System.out.println("Enter the value by which each character in the plaintext message gets shifted: ");
        int shiftKey = Integer.valueOf(sc.nextLine());

        System.out.println("Encrypted Data ===> "+encryptData(inputStr, shiftKey));
        System.out.println("Decrypted Data ===> "+decryptData(encryptData(inputStr, shiftKey), shiftKey));

        // close Scanner class object
        sc.close();
    }
}
```

Output:



Conclusion:

Submission Date :
Name of the student:
Roll No:

Experiment No. 04

Implementation of Symmetric & Asymmetric Key cryptography

Aim : Implementation of Symmetric & Asymmetric Key cryptography using Substitution Cipher & RSA Algorithm

Software Requirement:

C / C++ / Java/Python or equivalent compiler

Theory:

No one can deny the importance of security in data communications and networking. Security in networking is based on cryptography, the science and art of transforming messages to make them secure and immune to attack. Cryptography can provide several aspects of security related to the interchange of messages through networks. These aspects are confidentiality, integrity, authentication, and nonrepudiation.

Cryptography can provide confidentiality, integrity, authentication, and nonrepudiation of messages. Cryptography can also provide entity authentication.
Cryptography, a word with Greek origins, means "secret writing." However, we use the term to refer to the science and art of transforming messages to make them secure and immune to attacks.

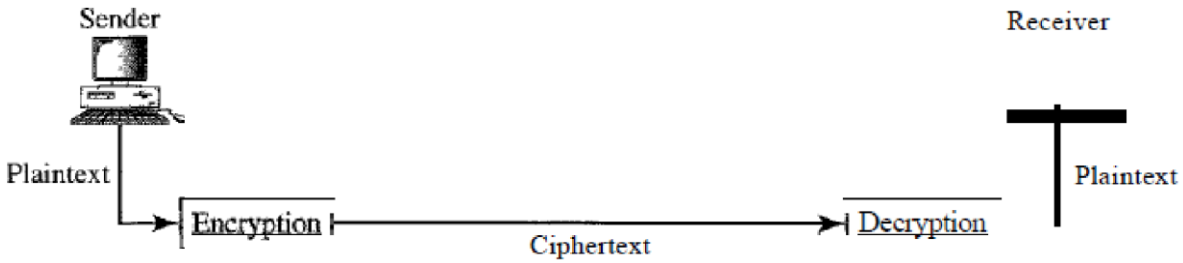


Fig: Cryptography component

Plaintext and Cipher text

The original message, before being transformed, is called plaintext. After the message is transformed, it is called ciphertext. An encryption algorithm transforms the plaintext into cipher text; a decryption algorithm transforms the cipher text back into plaintext. The sender uses an encryption algorithm, and the receiver uses a decryption algorithm.

Cipher

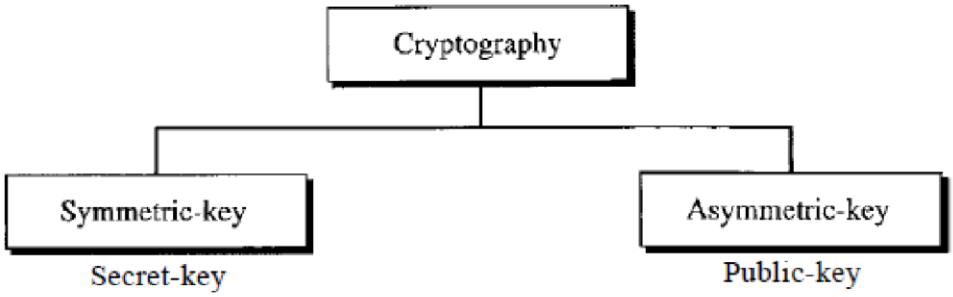
We refer to encryption and decryption algorithms as ciphers. The term cipher is also used to refer to different categories of algorithms in cryptography. This is not to say that every sender-receiver pair needs their very own unique cipher for a secure communication.

On the contrary, one cipher can serve millions of communicating pairs.

Key

A key is a number (or a set of numbers) that the cipher, as an algorithm, operates on. To encrypt a message, we need an encryption algorithm, an encryption key, and the plaintext. These create the cipher text. To decrypt a message, we need a decryption algorithm, a decryption key, and the cipher text. These reveal the original plaintext.

Categories of cryptography



Symmetric· Key Cryptography

In symmetric-key cryptography, the same key is used by both parties. The sender uses this key and an encryption algorithm to encrypt data; the receiver uses the same key and the corresponding decryption algorithm to decrypt the data.

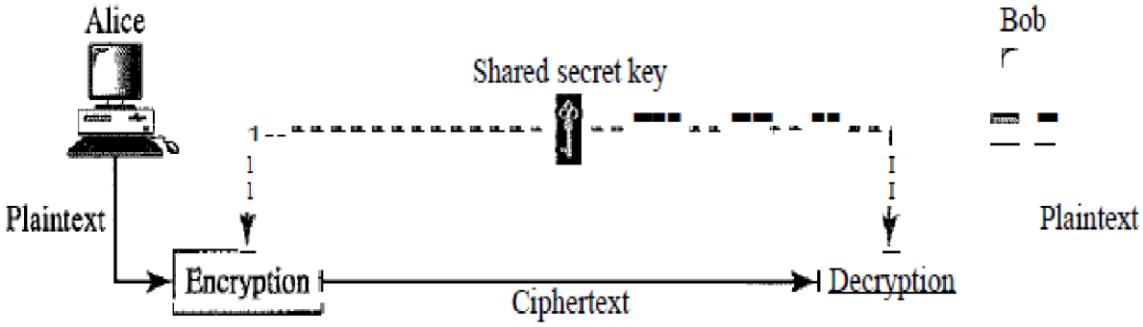


Fig: Symmetric Key Cryptography

1) **Monoalphabetic Cipher** :

A monoalphabetic cipher is any cipher in which the letters of the plain text are mapped to

cipher text letters based on a single alphabetic key. Examples of monoalphabetic ciphers would include the Caesar-shift cipher, where each letter is shifted based on a numeric key, and the atbash cipher, where each letter is mapped to the letter symmetric to it about the center of the alphabet.

2) **Polyalphabetic Cipher :**

A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The Vigenère cipher is probably the best-known example of a polyalphabetic cipher, though it is a simplified special case.

Asymmetric-Key Cryptography

In asymmetric or public-key cryptography, there are two keys: a private key and a public key. The private key is kept by the receiver. The public key is announced to the public. In Figure imagine Alice wants to send a message to Bob. Alice uses the public key to encrypt the message.
When the message is received by Bob, the private key is used to decrypt the message.

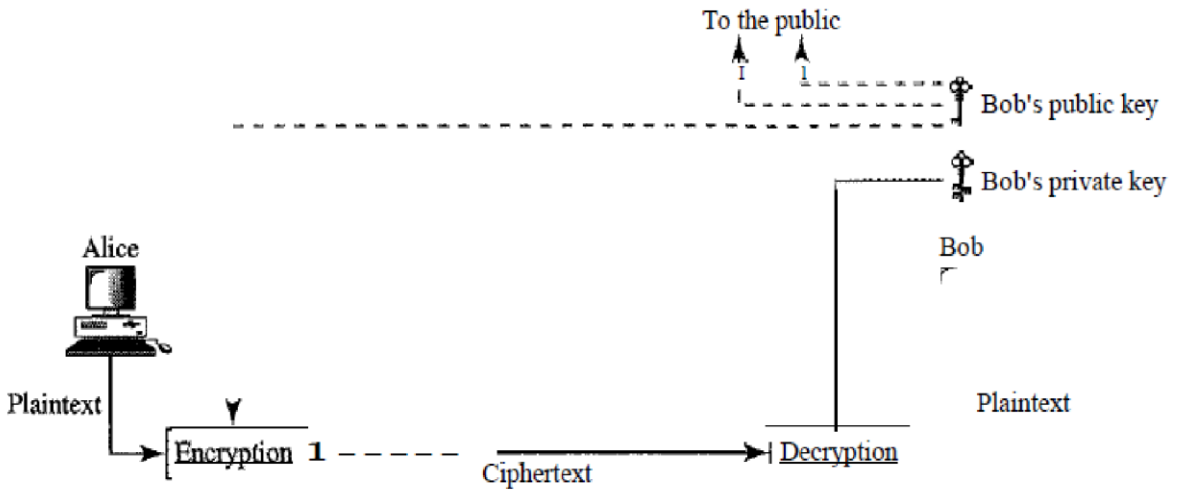


Fig: Asymmetric-Key Cryptography

RSA Algorithm :

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d and n such that with modular exponentiation for all integer m:

$$(m^e)^d = m \pmod n$$

The public key is represented by the integers n and e; and, the private key, by the integer

d. m represents the message. RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key.

ALGORITHM:

STEP-1: Select two co-prime numbers as p and q.

STEP-2: Compute n as the product of p and q.

STEP-3: Compute (p-1)*(q-1) and store it in z.

STEP-4: Select a random prime number e that is less than that of z.

STEP-5: Compute the private key, d as d*e mod(z)=1 **STEP-6:** The cipher text is computed as $(message)^e$ * mod n.

STEP-7: Decryption is done as $(cipher)^d$mod n.

Key Generation	
Select p, q	p, q both prime, p≠q
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1) \times (q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$
Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod n$
Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod n$

Problem Statement: Write Programs to implement

a) Encrypt plaintext: "SEND ME ONE THOUSAND RUPEE" using Symmetric key encryption techniques: Mono-alphabetic & Poly-alphabetic substitution Cipher

b) Encrypt plaintext “02” using the RSA public-key encryption algorithm

Program:

A) Symmetric key agorithms

// Java Program to Implement the Monoalphabetic Cypher

```
import java.io.*;
class GFG {
   public static char normalChar[]= { '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' };
   public static char codedChar[]= { 'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O','P', 'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K','L', 'Z', 'X', 'C',
'V', 'B', 'N', 'M' };
   public static String stringEncryption(String s)
   {
      String encryptedString = "";
      for (int i = 0; i < s.length(); i++) {    // comparing each character of the string and
         for (int j = 0; j < 26; j++) {// encoding each character using the indices
            if (s.charAt(i) == normalChar[j])
            {
               encryptedString += codedChar[j];
               break;
            }
            if (s.charAt(i) < 'a' || s.charAt(i) > 'z')
            {
               encryptedString += s.charAt(i);
               break;
            }
         }
      }

      // return encryptedString
      return encryptedString;
   }
   public static String stringDecryption(String s)
   {
      String decryptedString = "";
      for (int i = 0; i < s.length(); i++)
      {
         for (int j = 0; j < 26; j++) {
            if (s.charAt(i) == codedChar[j])
            {
               decryptedString += normalChar[j];
               break;
            }
            if (s.charAt(i) < 'A' || s.charAt(i) > 'Z')
            {
               decryptedString += s.charAt(i);
               break;
            }
         }
```

```
    }
    return decryptedString;
  }
  public static void main(String args[])
  {
    String str = "SEND ME ONE THOUSAND RUPEE";
    System.out.println("Plain text: " + str);
    String encryptedString = stringEncryption(str.toLowerCase());
    System.out.println("Encrypted message: "+encryptedString);
    System.out.println("Decrypted message: "+stringDecryption(encryptedString));

  }
}
```

// Java code to implement Polyalphabetic Cipher

```
class GFG
{
// This function generates the key in
// a cyclic manner until it's length isi'nt
// equal to the length of original text
static String generateKey(String str, String key)
{
        int x = str.length();
  for (int i = 0; ; i++)
        {
                if (x == i)
                        i = 0;
                if (key.length() == str.length())
                        break;
                key+=(key.charAt(i));
        }
        return key;
}
// This function returns the encrypted text
// generated with the help of the key
static String cipherText(String str, String key)
{
        String cipher_text="";
        for (int i = 0; i < str.length(); i++)
        {
                // converting in range 0-25
                int x = (str.charAt(i) + key.charAt(i)) %26;
                // convert into alphabets(ASCII)
                x += 'A';
                cipher_text+=(char)(x);
        }
        return cipher_text;
}
// This function decrypts the encrypted text
// and returns the original text
static String originalText(String cipher_text, String key)
```

```
{
        String orig_text="";
        for (int i = 0 ; i < cipher_text.length() && i < key.length(); i++)
        {
                // converting in range 0-25
                int x = (cipher_text.charAt(i) -
                                        key.charAt(i) + 26) %26;
                // convert into alphabets(ASCII)
                x += 'A';
                orig_text+=(char)(x);
        }
        return orig_text;
}
// This function will convert the lower case character to Upper case
static String LowerToUpper(String s)
{
        StringBuffer str =new StringBuffer(s);
        for(int i = 0; i < s.length(); i++)
        {
                if(Character.isLowerCase(s.charAt(i)))
                {
                        str.setCharAt(i, Character.toUpperCase(s.charAt(i)));
                }
        }
        s = str.toString();
        return s;
}
// Driver code
public static void main(String[] args)
{
        String Str = "SEND ME ONE THOUSAND RUPEE";
        String Keyword = "Aditya";

        String str = LowerToUpper(Str);
        String keyword = LowerToUpper(Keyword);

        String key = generateKey(str, keyword);
        String cipher_text = cipherText(str, key);

        System.out.println("Ciphertext : "+ cipher_text + "\n");
        System.out.println("Original/Decrypted Text : "+ originalText(cipher_text, key));
        }
}
```

B) ASymmetric key agorithms(RSA Algorithm)

// Java Program to Implement the RSA Algorithm

```
import java.math.*;
import java.util.*;
class RSA {
	public static void main(String args[])
	{
		int p, q, n, z, d = 0, e, i;
		// The number to be encrypted and decrypted
		int msg = 12;
		double c;
		BigInteger msgback;
		// 1st prime number p
		p = 3;
		// 2nd prime number q
		q = 11;
		n = p * q;
		z = (p - 1) * (q - 1);
		System.out.println("the value of z = " + z);
		for (e = 2; e < z; e++) {
			// e is for public key exponent
			if (gcd(e, z) == 1) {
				break;
			}
		}
		System.out.println("the value of e = " + e);
		for (i = 0; i <= 9; i++) {
			int x = 1 + (i * z);
			// d is for private key exponent
			if (x % e == 0) {
				d = x / e;
				break;
			}
		}
		System.out.println("the value of d = " + d);
		c = (Math.pow(msg, e)) % n;
		System.out.println("Encrypted message is : " + c);
		// converting int value of n to BigInteger
		BigInteger N = BigInteger.valueOf(n);
		// converting float value of c to BigInteger
		BigInteger C = BigDecimal.valueOf(c).toBigInteger();
		msgback = (C.pow(d)).mod(N);
		System.out.println("Decrypted message is : "+ msgback);
	}
	static int gcd(int e, int z)
	{
		if (e == 0)
			return z;
		else
			return gcd(z % e, e);
	}
}
```

Output:

Monoalphabetic Cypher

```
Output                                                    Clear
java -cp /tmp/TEOgbMQJ9p GFG
Plain text: SEND ME ONE THOUSAND RUPEE
Encrypted message: LTFR DT GFT ZIGXLQFR KXHTT
Decrypted message: send me one thousand rupee
```

Polyalphabetic Cipher

```
Output                                                    Clear
java -cp /tmp/TEOgbMQJ9p GFG

Ciphertext : SHVWRMEWWGCTTKWNQANGBKSPEH
Original/Decrypted Text : SENDTMETONETTHOUSANDTRUPEE
```

RSA Algorithm

```
Output                                                    Clear
java -cp /tmp/ugxaISIJJ6 RSA
the value of z = 20
the value of e = 3
the value of d = 7
Encrypted message is : 12.0
Decrypted message is : 12
```

Conclusion:

Submission Date :

Name of the student:

Roll No:

Experiment No. 05

Implementation of Steganography

Aim : Implementation of Steganography

Software Requirement:

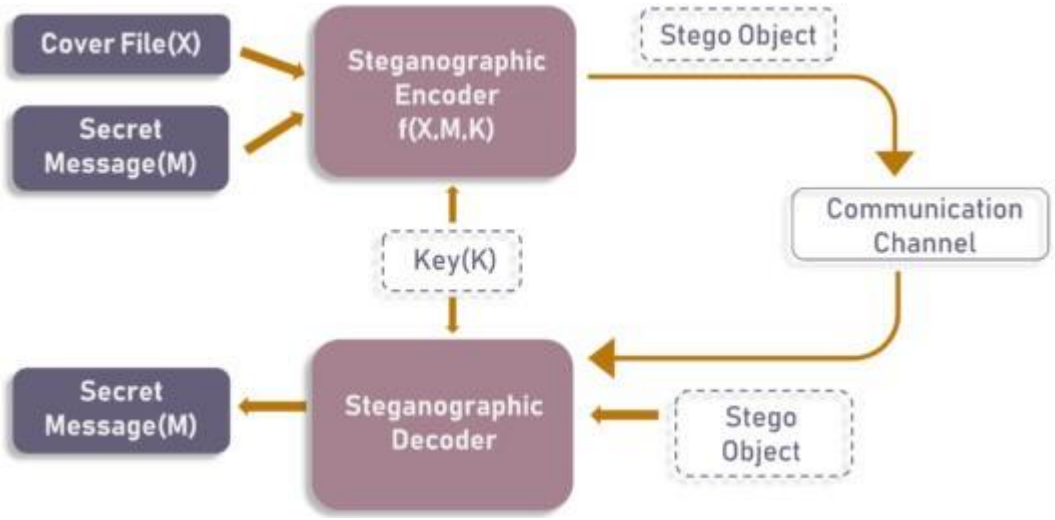
'Openstego tool' software, Xiao software & weblink

Theory:

What is Steganography?

Steganography is the art and science of embedding secret messages in a cover message in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message

The diagram below depicts a basic steganographic model.



As the image depicts, both cover file(X) and secret message(M) are fed into steganographic encoder as input. Steganographic Encoder function, f(X,M,K) embeds the secret message into a cover file. Resulting Stego Object looks very similar to your cover file, with no visible changes. This completes encoding. To retrieve the secret message, Stego Object is fed into Steganographic Decoder.

Steganography is the practice of concealing a secret message behind a normal message. It stems from two Greek words, which are *steganos,* means covered and *graphia,* means writing. Steganography is an ancient practice, being practiced in various forms for thousands of years to keep communications private. Now, we have a lot of modern steganographic techniques and tools to make sure that knows our data remains secret. Now you might be wondering if steganography is same as cryptography. No, they are two different concepts and this steganography tutorial presents you the main differences between them.

How is Steganography different from Cryptography?

At their core, both of them have almost the same goal, which is protecting a message or information from the third parties. However, they use a totally different mechanism to protect the information. Cryptography changes the information to ciphertext which cannot be understood without a decryption key. So, if someone were to intercept this encrypted message, they could easily see that some form of encryption had been applied. On the other hand, steganography does not change the format of the information but it conceals the existence of the message.

	STEGANOGRAPHY	CRYPTOGRAPHY
Definition	It is a technique to hide the existence of communication	It's a technique to convert data into an incomprehensible form
Purpose	Keep communication secure	Provide data protection
Data Visibility	Never	Always
Data Structure	Doesn't alter the overall structure of data	Alters the overall structure of data
Key	Optional, but offers more security if used	Necessary requirement
Failure	Once the presence of a secret message is discovered, anyone can use the secret data	If you possess the decryption key, then you can figure out original message from the ciphertext

So, in other words, steganography is more discreet than cryptography when we want to send confidential information. The downside being, the hidden message is easier to extract if the presence of secret is discovered. For the remainder of this steganography tutorial, we will learn about different steganography techniques and tools.

Steganography Techniques

Depending on the nature of the cover object(actual object in which secret data is embedded), steganography can be divided into five types:

1) Text Steganography
2) Image Steganography
3) Video Steganography
4) Audio Steganography
5) Network Steganography

Text Steganography

Text Steganography is hiding information inside the text files. It involves things like changing the format of existing text, changing words within a text, generating random character sequences or using context-free grammars to generate readable texts. Various techniques used to hide the data in the text are:

- Format Based Method
- Random and Statistical Generation
- Linguistic Method

Image Steganography

Hiding the data by taking the cover object as the image is known as image steganography. In digital steganography, images are widely used cover source because there are a huge number of bits present in the digital representation of an image. There are a lot of ways to hide information inside an image.

Common approaches include:

- Least Significant Bit Insertion
- Masking and Filtering
- Redundant Pattern Encoding
- Encrypt and Scatter
- Coding and Cosine Transformation

Best Tools to Perform Steganography

There are many software available that offer steganography. Some offer normal steganography, but a few offer encryption before hiding the data. These are the steganography tools which are available for free:

Openstego tool: is a free steganography tool using which you can hide confidential information in image files.

Stegosuite is a free steganography tool which is written in Java. With Stegosuite you can easily hide confidential information in image files.

Steghide is an open source Steganography software that lets you hide a secret file in image or audio file.

Xiao Steganography is a free software that can be used to hide data in BMP images or in WAV files.

SSuite Picsel is another free portable application to hide text inside an image file but it takes a different approach when compared to other tools.

OpenPuff is a professional steganographic tool where you can store files in image, audio, video or flash files

These are few tools to perform steganography. There are many other different tools with different capabilities.

Problem Statement: Download steganography software/use weblinks to perform Image steganography.

Make table with following columns:

Sr.no.|| Tools/software/website link|| input image file size|| output/coverted image file size|| features (of used tool/software/website)||remark(comparative remark)

Attach print of input image file & print of properties of output file generated by all tools

Output:

Original Photo:-

OpenStego:-

Steganography

Steganography Online:-

Conclusion:

SOFTWARE	Input image file size	Output image file size	Features	Remark
OPENSTEGO	46 kb	56.1 kb	Password Protection and Encryption is available. It supports Various Image formats(JPGE, PNG, BMP, etc).	**OPENSTEGO** Tool is better than **STEGANOGRAPHY online** tool.
STEGANOGRAPHY	8880086 bytes	8880087 bytes	Password Protection and Encryption is Depends on implementation. It supports Various Media formats(images, audio, video).	**STEGANOGRAPHY** is better among of these tools
STEGANOGRAPHY Online	200 kb	1.91 mb	Password Protection and Encryption is Depends on implementation. It supports Various Media formats(images, audio, video).	**STEGANOGRAPHY Online** is least preferred tool among all these tools

D.Y. Patil College of Engineering, Akurdi, Pune – 44

Department of Electronics And Telecommunication

Subject: Network Security(PR)

Submission Date :_____

Name of the student:_____

Roll No:_____

Experiment No. 06

Study of different wireless network components and features of the Mobile Security App

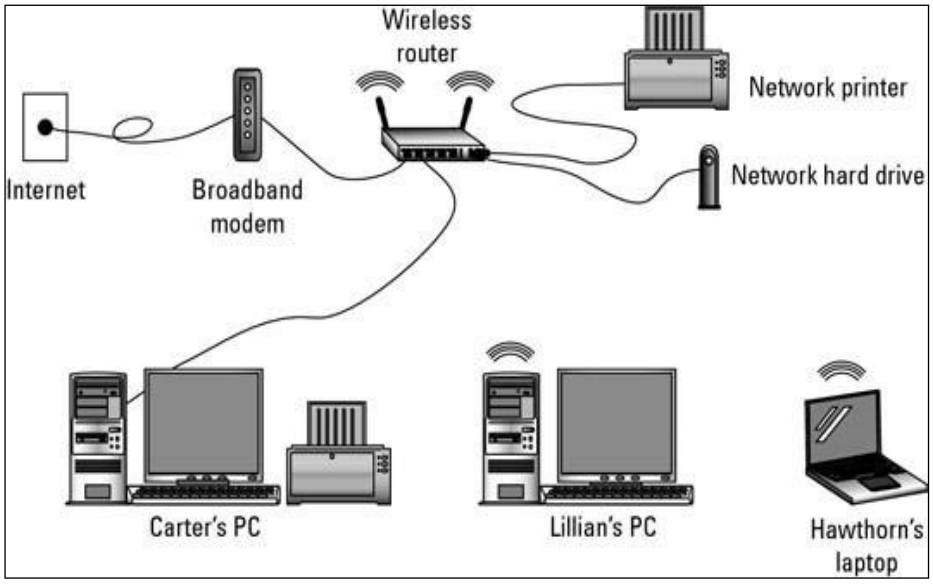
Aim: Study of different wireless network components and features of any one of the Mobile Security Apps.

Objectives

- Know about the devices and components in a wireless network.
- Know about the network security issues in different types of networkdevices.
- Identify a mobile security app and how it works for mobile security?

Introduction

As long as you have all the hardware, you can quickly set up any wirelessnetwork. Here is everything you need to know about the hardware you need to have in place before you use Windows to configure the wireless network. There are two types of wireless networks: infrastructure and ad hoc.The *infrastructure* network is most likely the type of wireless setup youhave in your home or office. It's laid out similarly to a wired network, butwithout wires.



The basic wireless, peer-to-peer network consists of these components:

Wireless Network Adapters

Wireless network adapters (also known as *wireless NICs* or *wirelessnetwork cards*) are required for each device on a wireless network. All newer laptop computers incorporate wireless

adapters as a built-in feature of the system.No wireless hardware other than adapters is required to build a small local network. However, to increase the performance of network connections, accommodate more computers, and increase the network's range, additional types of hardware can be deployed.

Wireless Routers

Wireless routers function comparably to traditional routers for wired Ethernet networks. One generally deploys wireless routers when building an all-wireless network from the ground up.Similar to routers, access points allow wireless networks to join an existing wired network. One typically deploys access points when growing a network that already has routers installed. In home networking, a single access point (or router) possesses sufficient range to span most residential buildings. Businesses in office buildings often must deploy multiple access points and/or routers.

Wireless Antennas

Access points and routers often utilize a Wi-Fi wireless antenna thatsignificantly increase the communication range of the wireless radio signal. These antennas are optional and removable on most equipment. It's alsopossible to mount aftermarket add-on antennas on wireless clients toincrease the range of wireless adapters.

Wireless Repeaters

A wireless repeater connects to a router or access point. Often called signal boosters or *range expanders*, repeaters serve as a two-way relay station for wireless radio signals, helping clients otherwise unable to receive anetwork's wireless signal to join.

Wire-based connections:

Almost every wireless router has one or more standard, wire-based Ethernet port. One port is used to connect the router toa broadband modem. Other Ethernet ports might be also available, allowing you to connect standard wire-based networking to the wireless hub.

Wireless NIC:

Your computer needs a wireless Network Interface Card, or NIC, to talk with the wireless router. A laptop comes standard with a wireless NIC, but for a desktop PC you have to get a wireless NIC as an option. It’s installed internally as an expansion card, or you can use one of the various plug-in USB wireless NICs.These are the components for infrastructure type of wireless

network. The other type of network called the *ad hoc* type of wireless network is basically a group of wireless computers connected with each other. An ad-hoc network has no central hub or router. Instead, all its computers can directly access the other computers' files and shared resources. They may or may not have Internet access, but that's not the point of the ad hoc network.

- One of the beauties of a wireless network is that you can mix in wired components as needed. If you need more Ethernet ports, for example, simply add a switch to the wireless router.
- Despite the wireless nature of wireless networking, you still need an Ethernet cable (a wire) to connect a wireless router to a broadband modem.
- Another advantage of a wireless network is that it's portable. It's far easier to pull up stakes with a wireless network than to pack up all the bits and pieces of a wired network. If you live in an apartment, or just move around a lot, wirelesses setup a good option.
- The term *access point* is often abbreviated AP. Don't be puzzled when you see the words *wireless AP* — it simply refers to the access point, not to the Associated Press.
- A wireless network is often called a *WLAN*, for wireless local-area network.
- A wireless network is also referred to by the term *Wi-Fi*. It standsfor *wi*reless *fi*delity.
- Adhoc networks are often used by computer gamers to gather in a single location to play games with each other.

Mobile App Security

Mobile app security is the extent of protection that mobile device applications (apps) have from malware and the activities of crackers and other criminals. The term can also refer to various technologies and production practices that minimize the risk of exploits to mobile devices through their apps.A mobile device has numerous components, all of them vulnerable to security weaknesses. The parts are made, distributed, and used by multiple players, each of whom plays a crucial role the security of a device. Each player should incorporate security measures into mobile devices as they are designed and built and into mobile apps as they are conceived and written, but these tasks are not always adequately carried out.

Common vulnerabilities for mobile devices include architectural flaws, device loss or theft, platform weakness, isolation and permission problems and application weaknesses.

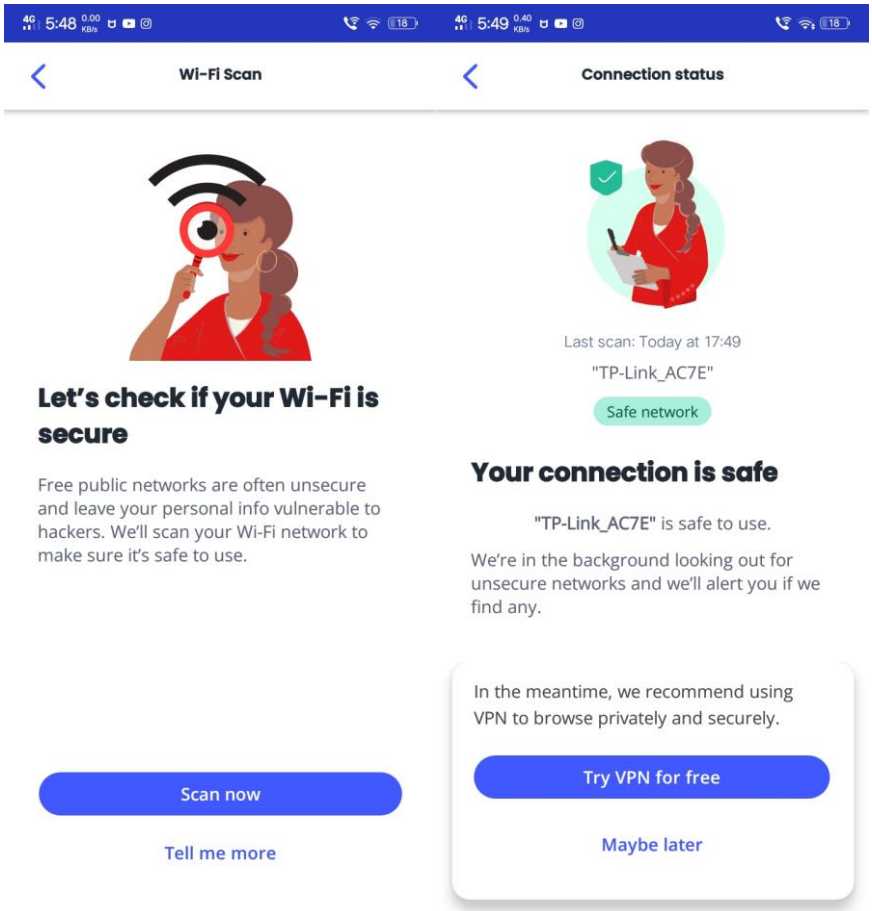
When evaluating mobile devices and apps for security, developers shouldask themselves the following questions.

- How do users obtain a particular app?
- Should a firm create its own app store?
- How is an app vetted before it is offered for sale?
- How is an app protected against malware?
- How can users tell the difference between a legitimate app and a fake?
- How easily can automatic update features get hijacked?
- What measures exist to control the risk of device jail breaking?
- What kind of permissions should a particular app ask for?
- Can any other apps keep track of when, where, and how a certain app is used?

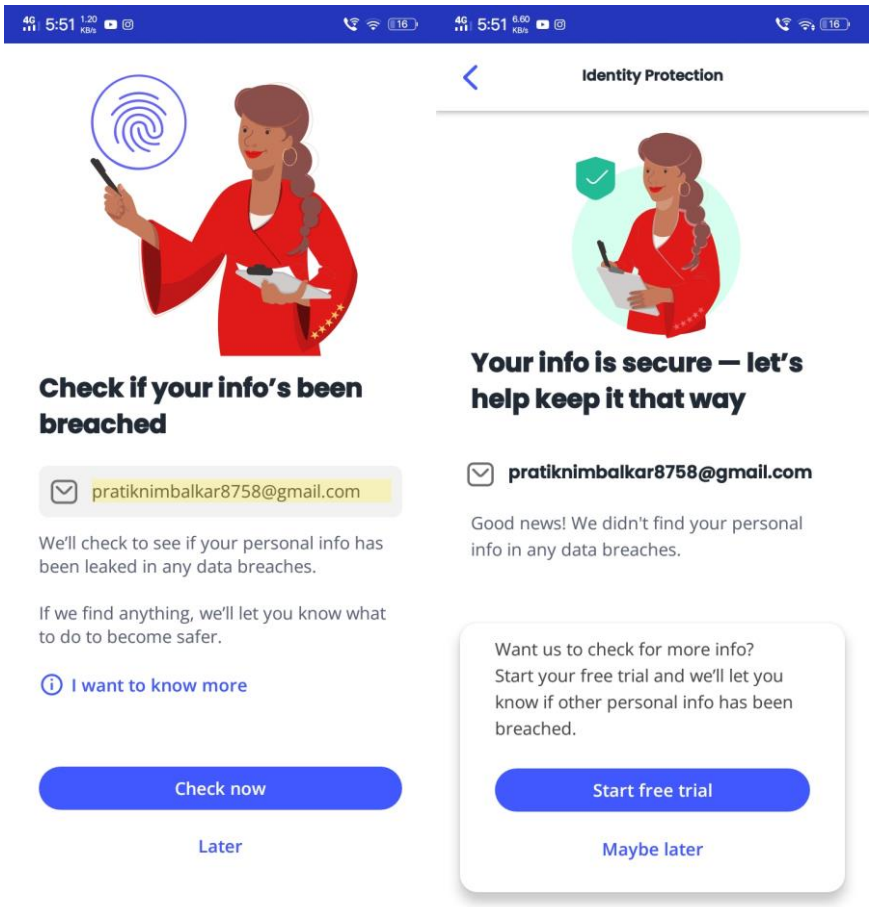
Problem Statement:

Note the features of any one mobile security app. Attach screenshots of features of that app

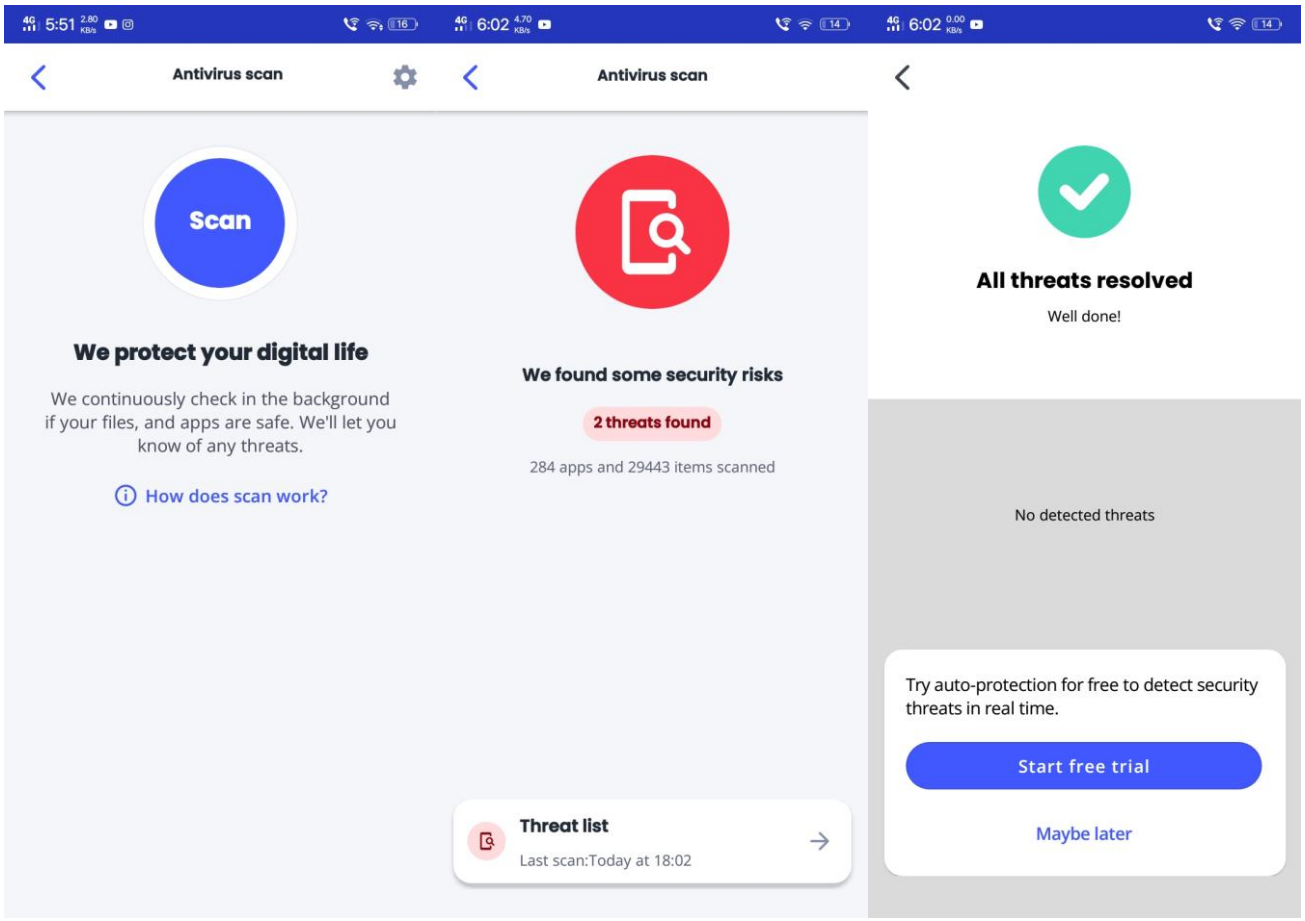
1) Wifi Scan



2) Identity Protection

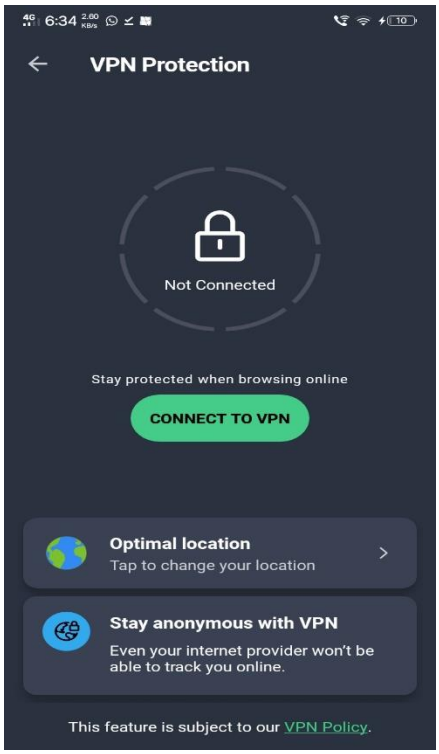


3) Antivirus Scan

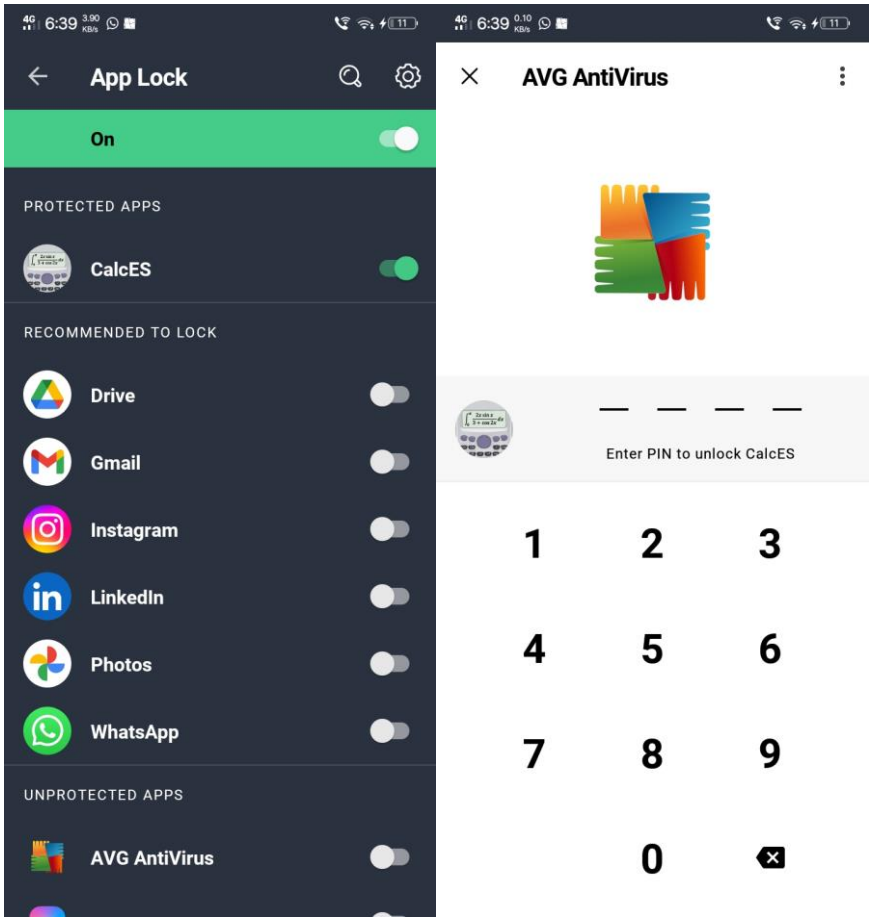


4) Protection Score

5) VPN Protection

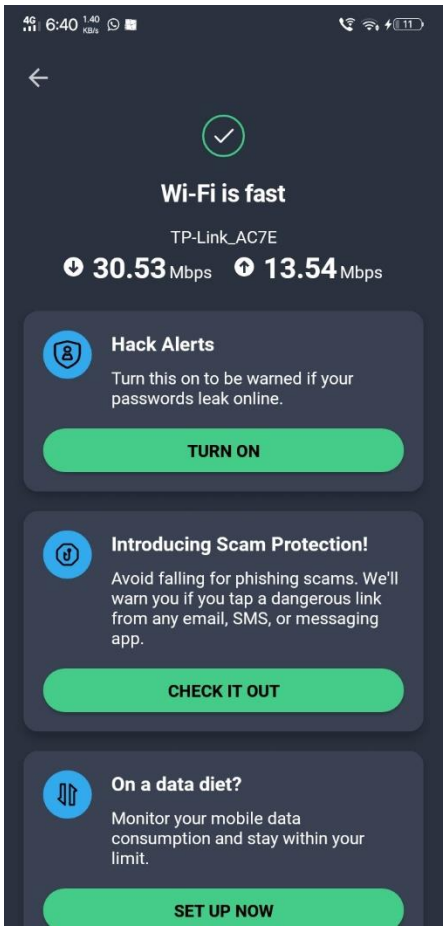


6) App Lock

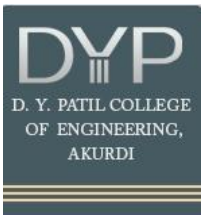

App Lock
On
PROTECTED APPS
CalcES
RECOMMENDED TO LOCK
Drive
Gmail
Instagram
LinkedIn
Photos
WhatsApp
UNPROTECTED APPS
AVG AntiVirus
AVG AntiVirus
Enter PIN to unlock CalcES
1 2 3
4 5 6
7 8 9
0

7) App Insights

8) Wifi Speed



Conclusion:



D.Y. Patil College of Engineering, Akurdi, Pune – 44

Department Of Electronics And Telecommunication

Submission Date :
Name of the student:
Roll No:

Experiment No. 09

Implementation of DES: Vlab

Aim : In this experiment, you are asked to design the triple DES cryptosystem provided that you are given an implementation of DES.

Software Requirement:

C / C++ / Java/Python or equivalent compiler

Theory:

Data Encryption Standard (DES)

DES stands for Data Encryption Standard. There are certain machines that can be used to crack the DES algorithm. The DES algorithm uses a key of 56-bit size. Using this key, the DES takes a block of 64-bit plain text as input and generates a block of 64-bit cipher text.

The DES process has several steps involved in it, where each step is called a round. Depending upon the size of the key being used, the number of rounds varies. For example, a 128-bit key requires 10 rounds, a 192-bit key requires 12 rounds, and so on.

Initial Permutation (IP)

The plain text is divided into smaller chunks of 64-bit size. The IP is performed before the first round. This phase describes the implementation of the transposition process. For example, the 58th bit replaces the first bit, the 50th bit replaces the second bit, and so on. The resultant 64-bit text is split into two equal halves of 32-bit each called Left Plain Text (LPT) and Right Plain Text (RPT).

Step 1: Key Transformation

We already know that the DES process uses a 56-bit key, which is obtained by eliminating all the bits present in every 8th position in a 64-bit key. In this step, a 48-bit key is generated. The 56-bit key is split into two equal halves and depending upon the number of rounds the bits are shifted to the left in a circular fashion.

Due to this, all the bits in the key are rearranged again. We can observe that some of the bits get eliminated during the shifting process, producing a 48-bit key. This process is known as compression permutation.

Step 2: Expansion Permutation

Let's consider an RPT of the 32-bit size that is created in the IP stage. In this step, it is expanded from 32-bit to 48-bit. The RPT of 32-bit size is broken down into 8 chunks of 4 bits each and extra two bits are added to every chunk, later on, the bits are permutated among themselves leading to 48-bit data. An XOR function is applied in between the 48-bit key obtained from step 1 and the 48-bit expanded RPT.

ALGORITHM:

The process begins with the 64-bit plain text block getting handed over to an initial permutation (IP) function.

The initial permutation (IP) is then performed on the plain text.

Next, the initial permutation (IP) creates two halves of the permuted block, referred to as Left Plain Text (LPT) and Right Plain Text (RPT).

Each LPT and RPT goes through 16 rounds of the encryption process.

Finally, the LPT and RPT are rejoined, and a Final Permutation (FP) is performed on the newly combined block.

The result of this process produces the desired 64-bit ciphertext.

The encryption process step (step 4, above) is further broken down into five stages:

Key transformation

Expansion permutation

S-Box permutation

P-Box permutation

XOR and swap

For decryption, we use the same algorithm, and we reverse the order of the 16 round keys.

PROCEDURE:

Step 1 : Generate Plaintext **m, keyA** and **keyB** by clicking on rexpective buttons **PART I** of the simulation page.

Step 2 : Enter generated Plaintext **m** from **PART I** to **PART II** in "Your text to be encrypted/decrypted:" block.

Step 3 : Enter generated **keyA** from **PART I** to **PART II** "Key to be used:" block and click on DES encrpt button to output ciphertext **c1**.This is First Encryption.

Step 4 : Enter generated ciphertext **c1** from **PART II** "Output:" Block to **PART II** in "Your text to be encrypted/decrypted:" block.

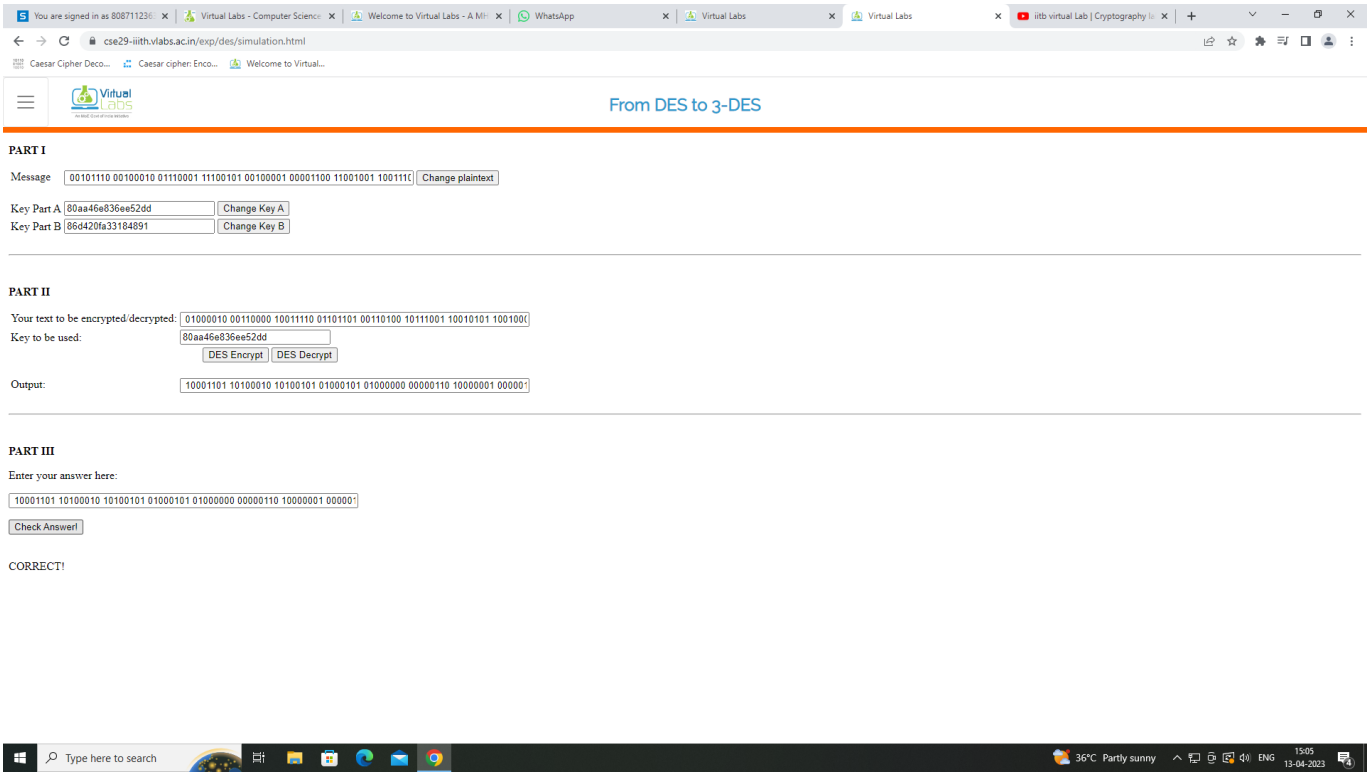
Step 5 : Enter generated **keyB** from **PART I** to **PART II** in "Key to be used:" block and click on DES decrypt button to output ciphertect **c2**.This is Second Encryption.

Step 6 : Enter generated ciphertext c2** from **PART II** "Output:" block to **PART II** in "Your text to be encrypted/decrypted:" block.

Step 7 : Enter generated **keyA** from **PART I** to **PART II** "Key to be used:" block and click on DES encrpt button to output ciphertext **c3**.This is Third Encryption. As Encryption is done thrice.This Scheme is called triple DES.

Step 7 : Enter generated ciphertext **c3** from **PART II** "Output:" Block to PART **III** "Enter your answer here:" block inorder to verify your Triple DES.

Output:



Conclusion:

D.Y. Patil College of Engineering, Akurdi, Pune – 44

Department Of Electronics And Telecommunication

Submission Date :
Name of the student:
Roll No:

Experiment No. 10

Implementation of AES: Vlab

Aim : In this experiment, you are asked to encrypt long messages using various modes of operation wherein a block cipher (in this case, AES) is provided to you .

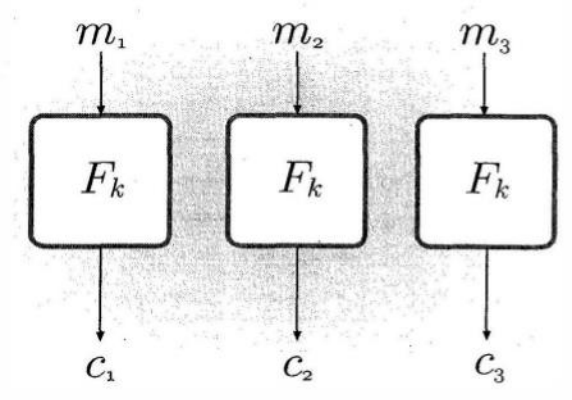
Software Requirement:

C / C++ / Java/Python or equivalent compiler

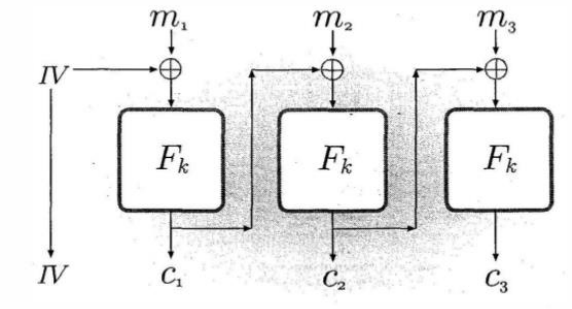
Theory:

The Advanced Encryption Standard (AES) is a symmetric block cipher chosen by the U.S. government to protect classified information.

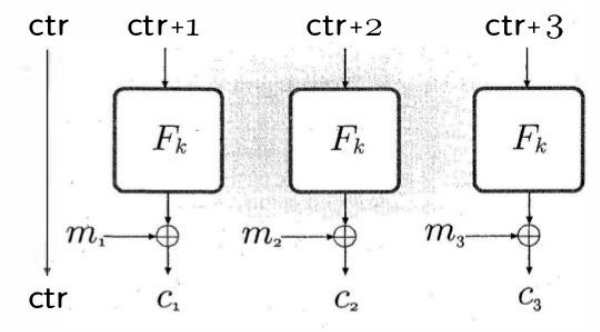
AES is implemented in software and hardware throughout the world to encrypt sensitive data. It is essential for government computer security, cybersecurity and electronic data protection.



Electronic Code Book(ECB) mode



Cipher Block Chaining(CBC) mode



Counter mode

PROCEDURE:

Step I : Choose a mode of operation from **PART I**

Step II : Select KeySize, Plaintext, KeyText, Intialization vector(IV)(for ECB and OFB modes only) and CTR(forctr mode only) in **PART II**

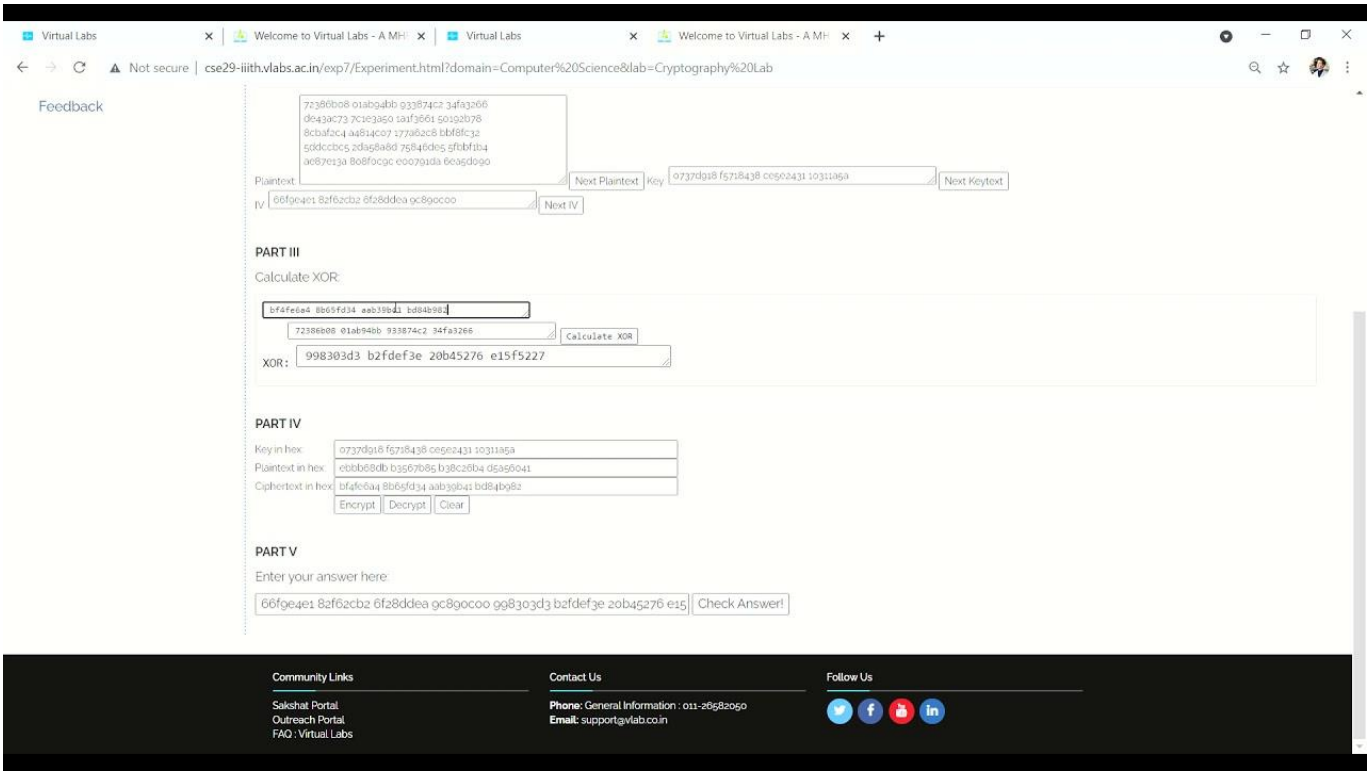
Step III : Whenever necessay use XOR opeartion in **PART III** in accordance with choosen mode of operation

Step IV : Use fuction **FK** and "Key in hex:" field in **PART IV** should be filled keytext generated in **Step2**

Step V : Fill "Plaintext in hex:" field with approriate value in accordance with choosen mode of operation and click on encrypt button

Step VI : Enter your answer in **PART V** to check your ciphertext

Output:



Conclusion: