R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(ISO 9001:2008 Certified Institution)

RSM Nagar, Puduvoyal – 601206

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

IT8761 – SECURITY LAB MANUAL

IV Year / 7th Sem

ANNA UNIVERSITY REGULATIONS 2017

RMK College of Engineering and Technology
RSM Nagar, Puduvoyal-601206

INSTITUTION VISION

To be knowledge hub of providing quality technical education and promoting research for building up of our nation and its contribution for the betterment of humanity.

INSTITUTION MISSION

- To make the best use of state-of-the-art infrastructure to ensure quality technical education.
- To develop industrial collaborations to promote innovation and research capabilities.
- To inculcate values and ethics to serve humanity

RMK College of Engineering and Technology
RSM Nagar, Puduvoyal-601206

Department of Computer Science and Engineering

VISION

To be a source of knowledge in the field of Computer Science and Engineering to cater to the growing need of industry and society.

MISSION

- To avail state-of-the art infrastructure for adopting cutting edge technologies and encouraging research activities.
- To promote industrial collaborations for professional competency.
- To nurture social responsibility and ethics to become worthy citizens.

PROGRAM EDUCATIONAL OBJECTIVES

Graduates of Computer Science and Engineering Program will

PEO I: Become globally competent professionals in all spheres and pursue higher education world over.

PEO II: Successfully carry forward domain knowledge in computing and allied areas to solve complex real world engineering problems.

PEO III: Continuously upgrade their technical knowledge and expertise to keep pace with the technological revolution.

PEO IV: Serve the humanity with social responsibility combined with ethics.

PROGRAM SPECIFIC OUTCOMES

Graduates of Computer Science and Engineering Program will be able to:

- Analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.
- Apply software engineering principles and practices for developing quality software for scientific and business applications.
- To implement cost-effective solutions for the betterment of both industry and society with the technological skills acquired through the Centers of Excellence.

RMK College of Engineering and Technology
RSM Nagar, Puduvoyal-601206

Department of Computer Science and Engineering

PROGRAM OUTCOME

- **Engineering Knowledge:** Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
- **Problem Analysis:** Identify, formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
- **Design/ Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.
- **Conduct investigations of complex problems** using research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.
- **Modern Tool Usage:** Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **The Engineer and Society:** Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilitiesrelevant to professional engineering practice.
- **Environment and Sustainability:** Understand the impact of professional engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.
- **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.
- **Individual and Team Work:** Function effectively as an individual, and as a member or leader in diverse teams and in multi-disciplinary settings.
- **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
- **Project Management and Finance:** Demonstrate knowledge and understanding of engineering and management principles and apply these to one‟s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **Life-long Learning:** Recognize the need for and have the preparation and ability to Engage in independent and life- long learning in the broadest context of technological Change.

RMK College of Engineering and Technology

RSM Nagar, Puduvoyal-601206

Department of Computer Science and Engineering

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.

2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.

3. Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.

b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.

c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.

5. Execute your task in the laboratory, and record the results / output in the lab observation notebook, and get certified by the concerned faculty.

6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.

7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.

8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.

9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.

10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab aftercompleting the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department

RMK College of Engineering and Technology

RSM Nagar, Puduvoyal-601206

Department of Computer Science and Engineering

COURSE OUTCOMES AND KNOWLEDGE LEVEL

Upon Completion of the course, the students will be able to

Course Outcomes	Description	Level in Bloom's Taxonomy
C408.1	Develop code for classical Encryption Techniques to solve the problems.	K3
C408.2	Build cryptosystems by applying symmetric and public key encryption algorithms.	K3
C408.3	Construct code for authentication algorithms.	K3
C408.4	Develop a signature scheme using Digital signature standard.	K3
C408.5	Demonstrate the network security system using open source tools	K2
C408.6	Develop code for classical Encryption Techniques to solve the problems.	K3
C408.7	Exhibit ethical principles in engineering practices	A3
C408.8	Perform task as an individual and / or team member to manage the task in time	A3
C408.9	Express the Engineering activities with effective presentation and report.	A3
C408.10	Interpret the findings with appropriate technological / research citation.	A2

Correlation between Outcomes (COs) and Program Outcomes (POs)

Subject Code and Name	Course Outcoms	Program Outcomes												Program Specific Outcomes		
		K3	K4	K5	K5	K3,K5 ,K6	A3	A2	A3	A3	A3	A3	A2	PSO-1	PSO-2	PSO-3
		PO-1	PO-2	PO-3	PO-4	PO-5	PO-6	PO-7	PO-8	PO-9	PO-10	PO-11	PO-12			
IT8761 Security Laboratory	CO1	3	2	2	-	-	-	-	-	-	-	-	-	2	3	-
	CO2	3	2	2	-	-	-	-	-	-	-	-	-	2	3	-
	CO3	3	2	2	-	-	-	-	-	-	-	-	-	2	3	-
	CO4	3	2	2	-	-	-	-	-	-	-	-	-	2	3	-
	CO5	2	1	1	-	-	-	-	-	-	-	-	-	2	2	-
	CO6	3	2	2	-	-	-	-	-	-	-	-	-	2	3	-
	CO7	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-
	CO8	-	-	-	-	-	-	-		3		3	-	-	-	-
	CO9	-	-	-	-	-	-	-			3	-	-	-	-	-
	CO10	-	-	-	-	-	-	-	-	-	-	-	3	-	-	-
		3	2	2	-	-	-	-	-	-	-	-	-	2	3	-

SYLLABUS

IT8761	SECURITY LABORATORY	L T P C
		0 0 4 2

OBJECTIVES:

- To learn different cipher techniques
- To implement the algorithms DES, RSA, MD5, SHA-1
- To use network security tools and vulnerability assessment tools

LIST OF EXPERIMENTS

1. Perform encryption, decryption using the following substitution techniques

(i) Ceaser cipher, (ii) playfair cipher iii) Hill Cipher iv) Vigenere cipher

2. Perform encryption and decryption using following transposition techniques

i) Rail fence ii) row & Column Transformation

3. Apply DES algorithm for practical applications.

4. Apply AES algorithm for practical applications.

5. Implement RSA Algorithm using HTML and JavaScript

6. Implement the Diffie-Hellman Key Exchange algorithm for a given problem.

7. Calculate the message digest of a text using the SHA-1 algorithm.

8. Implement the SIGNATURE SCHEME - Digital Signature Standard.

9. Demonstrate intrusion detection system (ids) using any tool eg. Snort or any other s/w.

10. Automated Attack and Penetration Tools

Exploring N-Stalker, a Vulnerability Assessment Tool

11. Defeating Malware

i) Building Trojans ii) Rootkit Hunter

TOTAL: 60 PERIODS

OUTCOMES:

Upon Completion of the course, the students will be able to:

- Develop code for classical Encryption Techniques to solve the problems.
- Build cryptosystems by applying symmetric and public key encryption algorithms.
- Construct code for authentication algorithms.
- Develop a signature scheme using Digital signature standard.
- Demonstrate the network security system using open source tools

REFERENCES:

1. Build Your Own Security Lab, Michael Gregg, Wiley India

LIST OF EQUIPMENT FOR A BATCH OF 30 STUDENTS:

SOFTWARE: C / C++ / Java or equivalent compiler GnuPG, Snort, N-Stalker or Equivalent

HARDWARE: Standalone desktops - 30 Nos. (or) Server supporting 30 terminals or more.

LIST OF CONTENTS

Ex. No : 1 (a) **CAESAR CIPHER**

Date :

AIM

To perform encryption and decryption using Caesar Cipher.

ALGORITHM

1. Encryption:

   C = E(k, p) = (p + k) mod 26

   C - Cipher Text, p – Plain Text, k – key and E – Encryption Function

2. Decryption:

   p = D(k, C) = (C - k) mod 26

   D – Decryption Function

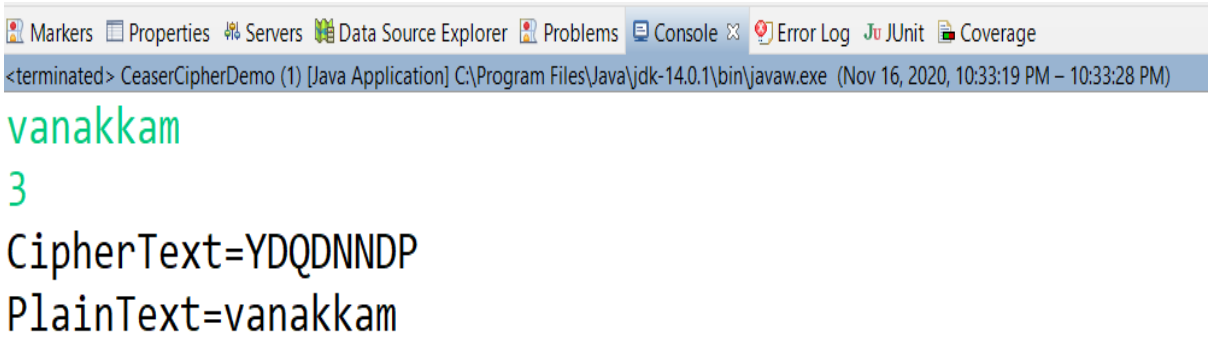
3. For ease of encryption and decryption:
   i) Input string is converted into uppercase and then to character array.
   ii) Each character is converted into its appropriate ASCII character. So A = 65, B = 66 and Z = 90.
   iii) Before Encryption/Decryption operation, input ASCII character is subtracted by 65 to make A = 0, B = 1 and Z = 25. (Since key space = {0,1,2,..,25})
   iv) After Encryption/Decryption operation, output ASCII character are added by 65 to compensate for earlier subtraction. (Since ASCII for Upper case alphabets are from 65-90)

/* Ex.No.1(a) CeaserCipher */

```
import java.util.Scanner;

class CeaserCipher {
	public String encryption(String plainText, int key) {
		String cipherText = "";
		char[] plainTextArray = plainText.toUpperCase().toCharArray();
		char[] cipherTextArray = new char[plainTextArray.length];
		for (int i = 0; i < plainTextArray.length; i++) {
			/*
			 * cipherTextArray[i] = (char) (((int) plainTextArray[i]
+ key - 65) % 26 + 65);
			 */
			cipherTextArray[i] = (char) (((int)
Math.floorMod(plainTextArray[i] + key - 65, 26)) + 65);
		}
		cipherText = String.valueOf(cipherTextArray);
		return cipherText;
	}

	public String decryption(String cipherText, int key) {
		String plainText = "";
		char[] cipherTextArray = cipherText.toCharArray();
		char[] plainTextArray = new char[cipherTextArray.length];
		for (int i = 0; i < cipherTextArray.length; i++) {
			/*
			 * plainTextArray[i] = (char) (((int) cipherTextArray[i]
- key - 65) % 26 + 65);
			 * Negative value produces wrong value
			 */
			plainTextArray[i] = (char) (((int)
Math.floorMod(cipherTextArray[i] - key - 65, 26)) + 65);
		}
		plainText = String.valueOf(plainTextArray).toLowerCase();
		return plainText;
	}
```

```
}

public class CeaserCipherDemo {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		CeaserCipher ceaserCipher = new CeaserCipher();
		Scanner scanner = new Scanner(System.in);
		String plainText = scanner.nextLine();
		int key = scanner.nextInt();
		String cipherText = ceaserCipher.encryption(plainText, key);
		System.out.println("CipherText=" + cipherText);
		String recoveredPlainText =
ceaserCipher.decryption(cipherText, key);
		System.out.println("PlainText=" + recoveredPlainText);
		scanner.close();
	}

}
```

OUTPUT



RESULT

The Java program to perform encryption and decryption using Caesar Cipher was successfully implemented.

Ex. No : 1 (b) **PLAYFAIR CIPHER**

Date :

AIM

To perform encryption using PlayFair Cipher.

ALGORITHM

1. Generate Digrams.
   a) Two characters in the digram should be unique.
      i) Repeating plaintext letters that are in the same pair are separated with a filler letter, such as x.
      Example: “success” should be constructed as “su cx ce sx sx”
   b) Character ‘i’ and ‘j’ are same.
2. Generate Key Matrix.
   a) Remove redundant characters in the key and should be filled first.
   b) Then fill the matrix with the remaining alphabets and it should not repeat any of the characters in the key.
3. Substitute. Once the plaintext digrams and key matrix are generated, perform substitution.
   a) Two plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row circularly following the last.
   b) Two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column circularly following the last.
   c) Otherwise, each plaintext letter in a pair is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter.

/*Ex.No.1(b) PlayFair Cipher*/

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Scanner;
import java.util.Set;

class PlayFairCipher {
	public String encrypt(String plainText, String key) {
		List<Character> digrams = generateDigrams(plainText);
		StringBuilder cipherText = new StringBuilder();
		Character[][] keyMatrix = generateKeyMatrix(key);
		// System.out.println(Arrays.deepToString(keyMatrix));
		List<Character> pair = new ArrayList<>();
		char ch1 = '\0';
		char ch2 = '\0';
		for (int i = 0; i < digrams.size(); i = i + 2) {
			ch1 = digrams.get(i);
			ch2 = digrams.get(i + 1);
			pair = substitute(ch1, ch2, digrams, keyMatrix);
			cipherText.append(pair.get(0));
			cipherText.append(pair.get(1));
		}
		return (cipherText.toString().toUpperCase());
	}

	public static List<Character> generateDigrams(String msg) {
		List<Character> msgDigram = new ArrayList<>();
		for (char ch : msg.toCharArray()) {
			msgDigram.add(ch);
		}
		int i = 0;
		System.out.println("Plaintext Digrams:");
		while (i < msgDigram.size()) {
```

```
                if (i == msgDigram.size() - 1) {
                      msgDigram.add('x');
                }
                // If 'j' present in plaintext, replace it with 'i'
                if (msgDigram.get(i) == 'j') {
                      msgDigram.remove(i);
                      msgDigram.add(i, 'i');
                }
                if (msgDigram.get(i + 1) == 'j') {
                      msgDigram.remove(i + 1);
                      msgDigram.add(i + 1, 'i');
                }
                // If both the characters are same
                if (msgDigram.get(i) == msgDigram.get(i + 1)) {
                      msgDigram.add(i + 1, 'x');
                }
                System.out.print(msgDigram.get(i) + "" + msgDigram.get(i
+ 1) + " ");
                i = i + 2;
            }
            return msgDigram;
      }

      // Construct a 5 x 5 Matrix
      private Character[][] generateKeyMatrix(String key) {
            Character[] uniqueKey = findUniqueKey(key);
            Character[] keyVector = generateKeyVector(uniqueKey);
            Character[][] matrix = new Character[5][5];
            System.out.println("\nKEY MATRIX");
            for (int i = 0; i < matrix.length; i++) {
                  for (int j = 0; j < matrix[0].length; j++) {
                        matrix[i][j] = keyVector[(i * matrix[0].length) +
j];

      System.out.print(Character.toUpperCase(matrix[i][j]) + "\t");
                  }
```

```
                System.out.println("\n");
            }
            return matrix;
        }

        public List<Character> substitute(char ch1, char ch2,
List<Character> digrams, Character[][] keyMatrix) {
            List<Character> pair = new ArrayList<>();
            int row1 = 0, col1 = 0, row2 = 0, col2 = 0;
            for (int i = 0; i < keyMatrix.length; i++) {
                for (int j = 0; j < keyMatrix[0].length; j++) {
                    if (ch1 == keyMatrix[i][j]) {
                        row1 = i;
                        col1 = j;
                    }
                    if (ch2 == keyMatrix[i][j]) {
                        row2 = i;
                        col2 = j;
                    }
                }
            }
            if (row1 == row2) {
                pair.add(keyMatrix[row1][(col1 + 1) % 5]);
                pair.add(keyMatrix[row2][(col2 + 1) % 5]);
            } else if (col1 == col2) {
                pair.add(keyMatrix[(row1 + 1) % 5][col1]);
                pair.add(keyMatrix[(row2 + 1) % 5][col2]);
            } else {
                pair.add(keyMatrix[row1][col2]);
                pair.add(keyMatrix[row2][col1]);
            }
            return pair;
        }

        // Remove Redundancy from Key
        public Character[] findUniqueKey(String key) {
```

```
            Set<Character> uniqueKey = new LinkedHashSet<>();
            for (Character c : key.toCharArray()) {
                  if (c == 'j') {
                        uniqueKey.add('i');
                  } else {
                        uniqueKey.add(c);
                  }
            }
            // System.out.println(uniqueKey);
            return uniqueKey.toArray(new Character[uniqueKey.size()]);
      }

      public Character[] generateKeyVector(Character[] uniqueKey) {
            Character[] alphabets = new Character[26];
            Character[] tempKey = new Character[25];
            for (int i = 0; i < 26; i++) {
                  char c = (char) (i + 97);
                  alphabets[i] = c;
            }
            tempKey = difference(uniqueKey, alphabets);
            Character[] matrixKey = appendKey(uniqueKey, tempKey);
            return matrixKey;
      }

      // Get characters for remaining Key Matrix
      private Character[] difference(Character[] uniqueKey, Character[]
alphabets) {
            Set<Character> set1 = new
LinkedHashSet<>(Arrays.asList(uniqueKey));
            Set<Character> set2 = new
LinkedHashSet<>(Arrays.asList(alphabets));
            set2.removeAll(set1);
            if (set1.contains('j')) {
                  set2.remove('i');
            } else {
                  set2.remove('j');
```

```
            }
            return set2.toArray(new Character[set2.size()]);
      }

      // Append Remaining Characters to the Key Matrix
      private Character[] appendKey(Character[] uniqueKey, Character[]
tempKey) {
            Character[] finalKey = new Character[25];
            for (int i = 0; i < uniqueKey.length; i++) {
                  finalKey[i] = uniqueKey[i];
            }
            for (int i = uniqueKey.length, j = 0; i < 25; i++, j++) {
                  finalKey[i] = tempKey[j];
            }
            return finalKey;
      }
}

public class PlayFairCipherDemo {

      public static void main(String[] args) {
            PlayFairCipher playFairCipher = new PlayFairCipher();
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the plaintext:");
            String plainText = sc.nextLine().toLowerCase();
            System.out.println("Enter the key:");
            String key = sc.nextLine().toLowerCase();
            String cipherText = playFairCipher.encrypt(plainText, key);
            System.out.println("CipherText=" + cipherText);
            sc.close();
      }
}
```

OUTPUT

```
<terminated> PlayFairCipherDemo [Java Application] C:\Pro
Enter the plaintext:
success
Enter the key:
dravid
Plaintext Digrams:
su cx ce sx sx
KEY MATRIX
D       R       A       V       I

B       C       E       F       G

H       K       L       M       N

O       P       Q       S       T

U       W       X       Y       Z

CipherText=OYEWEFQYQY
```

RESULT

The Java program to perform encryption using PlayFair Cipher was successfully implemented.

Ex. No : 1 (c) **HILL CIPHER**

Date :

AIM

To perform encryption using Hill Cipher.

ALGORITHM

4. Encryption:

$c1 = (k11p1 + k21p2 + k31p3) \bmod 26$

$c2 = (k12p1 + k22p2 + k32p3) \bmod 26$

$c3 = (k13p1 + k23p2 + k33p3) \bmod 26$

c1, c2 and c3 are cipher text matrix elements

k11, k21, k31, k12, k22, k32, k13, k23 and k33 are key matrix elements

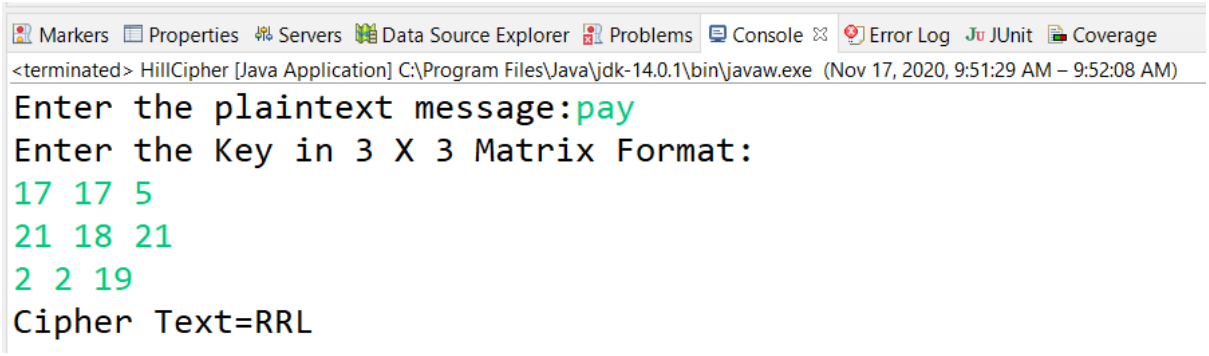
p1, p2, p3 are plain text matrix elements

5. For ease of encryption and decryption:
   i) Input string is converted into uppercase and then to character array.
   ii) Each character is converted into its appropriate ASCII character. So A = 65, B = 66 and Z = 90.
   iii) Before Encryption/Decryption operation, input ASCII character is subtracted by 65 to make A = 0, B = 1 and Z = 25. (Since key space = {0,1,2,..,25})
   iv) After Encryption/Decryption operation, output ASCII character are added by 65 to compensate for earlier subtraction. (Since ASCII for Upper case alphabets are from 65-90)

/*Ex.No.1(c) Hill Cipher*/

```
import java.util.Scanner;

public class HillCipher {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter the plaintext message:");
		String plainText = sc.nextLine();
		int[][] key = new int[3][3];
		System.out.println("Enter the Key in 3 X 3 Matrix Format:");
		for (int i = 0; i < 3; i++) {
			for (int j = 0; j < 3; j++) {
				key[i][j] = sc.nextInt();
			}
		}
		String cipherText = encrypt(plainText.toUpperCase(), key);
		System.out.println("Cipher Text=" + cipherText);
	}

	public static String encrypt(String plainText, int key[][]) {
		char[] text = plainText.toCharArray();
		int c1, c2, c3, p1, p2, p3;
		p1 = (int) text[0] - 65;
		p2 = (int) text[1] - 65;
		p3 = (int) text[2] - 65;
		c1 = (key[0][0] * p1 + key[1][0] * p2 + key[2][0] * p3) % 26;
		c2 = (key[0][1] * p1 + key[1][1] * p2 + key[2][1] * p3) % 26;
		c3 = (key[0][2] * p1 + key[1][2] * p2 + key[2][2] * p3) % 26;
		char[] cipherText = new char[3];
		cipherText[0] = (char) (c1 + 65);
		cipherText[1] = (char) (c2 + 65);
		cipherText[2] = (char) (c3 + 65);
		return String.valueOf(cipherText);
	}
}
```

OUTPUT



RESULT

The Hill Cipher encryption was successfully implemented.

Ex. No : 1 (d) **VIGENERE CIPHER**

Date :

AIM

To perform encryption and decryption using Vigenere Cipher.

ALGORITHM

1. Encryption

   $C_i = (p_i + k_{i \bmod m}) \bmod 26$

2. Decryption

   $p_i = (C_i - k_{i \bmod m}) \bmod 26$

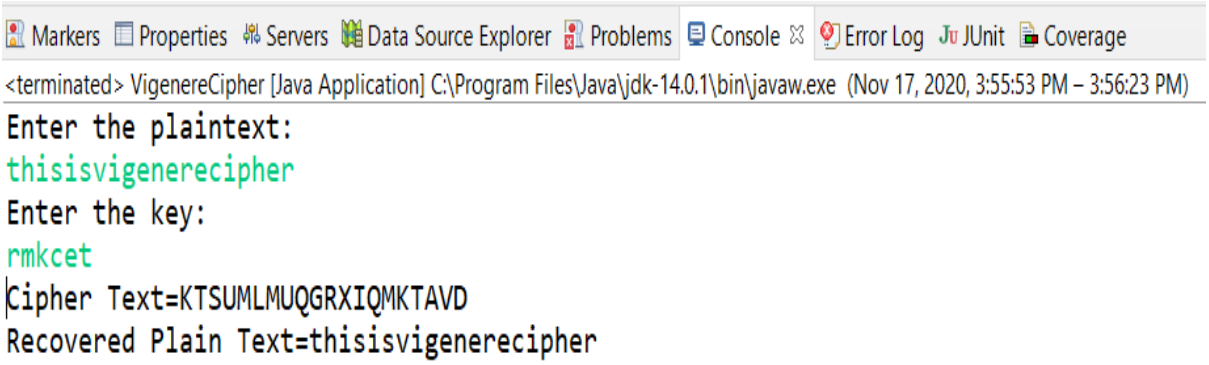
3. For ease of encryption and decryption:
   i) Input string is converted into uppercase and then to character array.
   ii) Each character is converted into its appropriate ASCII character. So A = 65, B = 66 and Z = 90.
   iii) Before Encryption/Decryption operation, input ASCII character is subtracted by 65 to make A = 0, B = 1 and Z = 25. (Since key space = {0,1,2,..,25})
   iv) After Encryption/Decryption operation, output ASCII character are added by 65 to compensate for earlier subtraction. (Since ASCII for Upper case alphabets are from 65-90)

/*Ex.No.1(d) Vigenere Cipher*/

```
package com.securitylab.classical;

import java.util.Arrays;
import java.util.Scanner;

public class VigenereCipher {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the plaintext:");
		String plainText = sc.nextLine();
		System.out.println("Enter the key:");
		String key = sc.nextLine();
		String cipherText = encrypt(plainText, key);
		System.out.println("Cipher Text=" + cipherText);
		System.out.println("Recovered Plain Text=" +
decrypt(cipherText, key));
		sc.close();
	}

	public static String encrypt(String plainText, String key) {
		char[] plainTextChar = plainText.toUpperCase().toCharArray();
		char[] keyChar = padKey(key, plainTextChar.length);
		char[] cipherTextChar = new char[keyChar.length];
		for (int i = 0; i < cipherTextChar.length; i++) {
			cipherTextChar[i] = (char) (int) ((plainTextChar[i] +
keyChar[i] - 130) % 26 + 65);
		}
		return (String.valueOf(cipherTextChar));
	}

	public static String decrypt(String cipherText, String key) {
		char[] recoveredPlainTextChar = cipherText.toCharArray();
		char[] keyChar = padKey(key, recoveredPlainTextChar.length);
		for (int i = 0; i < recoveredPlainTextChar.length; i++) {
```

```
                  recoveredPlainTextChar[i] = (char) (int)
((cipherText.charAt(i) - keyChar[i] + 26) % 26 + 65);
            }
            return String.valueOf(recoveredPlainTextChar).toLowerCase();
      }

      // Making length of Key same as length of Plain Text message
      public static char[] padKey(String key, int length) {
            char[] keyChar =
Arrays.copyOf(key.toUpperCase().toCharArray(), length);
            int i = 0;
            for (int j = key.toCharArray().length; j < keyChar.length;
j++) {
                  keyChar[j] = keyChar[i];
                  i++;
            }
            return keyChar;
      }
}
```

OUTPUT



RESULT

The Vigenere Cipher has been successfully implemented.

Ex. No : 2(a) **RAIL FENCE CIPHER**

Date :

AIM

To perform encryption and decryption using Rail Fence technique.

ALGORITHM

1. Encryption
    - The plaintext is written down as a sequence of diagonals and then read off as a sequence of rows.
    - For example, to encipher the message “meet me after the toga party” with a rail fence of depth 2, we write the following:

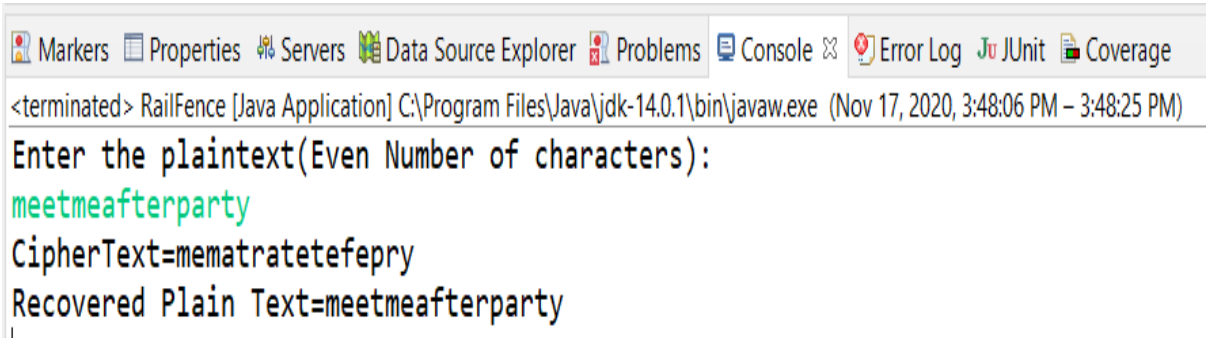
      m e m a t r h t g p r y

      e t e f e t e o a a t
    - The encrypted message is MEMATRHTGPRYETEFETEOAAT.

/*Ex.No.2(a) Rail Fence Cipher*/

```
import java.util.Scanner;

public class RailFence {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the plaintext(Even Number of characters):");
		String plainText = sc.nextLine();
		char[] plainTextChar = plainText.toCharArray();
		char[] cipherTextChar = new char[plainTextChar.length];
		int i, j = 0;
		for (i = 0; i < (cipherTextChar.length / 2); i++) {
			cipherTextChar[i] = plainTextChar[j];
			cipherTextChar[i + plainTextChar.length / 2] = plainTextChar[j + 1];
			j = j + 2;
		}

		System.out.println("CipherText=" + String.valueOf(cipherTextChar));
		char[] recoveredPlainTextChar = new char[cipherTextChar.length];
		j = 0;
		for (i = 0; i < recoveredPlainTextChar.length; i = i + 2) {
			recoveredPlainTextChar[i] = cipherTextChar[j];
			recoveredPlainTextChar[i + 1] = cipherTextChar[j +
(plainTextChar.length / 2)];
			j++;
		}
		System.out.println("Recovered Plain Text=" +
String.valueOf(recoveredPlainTextChar));
		sc.close();
	}
}
```

OUTPUT



RESULT

The Rail Fence Cipher has been successfully implemented using Java.

Ex. No : 2(b) **COLUMN TRANSPOSITION CIPHER**

Date :

AIM

To perform encryption using column transposition technique.

ALGORITHM

2. Encryption
    - The plaintext is written down as a sequence of diagonals and then read off as a sequence of rows.
    - For example, to encipher the message “meet me after the toga party” with a rail fence of depth 2, we write the following:

      m e m a t r h t g p r y

      e t e f e t e o a a t
    - The encrypted message is MEMATRHTGPRYETEFETEOAAT.

```
                /* Ex. No. 2(b) Column Transformation*/
import java.util.Map;
import java.util.Scanner;
import java.util.TreeMap;

public class ColumnTransposition {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the Plaintext:");
		String plainText = sc.nextLine();
		System.out.println("Enter the Key:");
		int key = sc.nextInt();
		encrypt(plainText, key);
		sc.close();
	}

	public static void encrypt(String plainText, int key) {
		StringBuilder sb = new StringBuilder(plainText);
		// Padding with filler characters
		if (plainText.length() % 5 != 0) {
			for (int i = plainText.length() % 5; i < 5; i++) {
				sb.append("x");
			}
		}
		// keyArr is the key vector: 43215
		char[] keyArr = String.valueOf(key).toCharArray();
		Map<Integer, Integer> keyInfo = new TreeMap<>();
		int index = 0;
		System.out.println("\nPlaintext Matrix\n");
		for (int j : keyArr) {
			j = Character.getNumericValue(keyArr[index]);
			// Printing key vector
			System.out.print(j + "\t");
			// keyInfo Map: Col1-4, Col2-3, Col3-2, Col4-1, Col5-5
			// index: Column number; j:key value
```

```
                keyInfo.put(j, index);
                index++;
            }
            System.out.println("\n");
            // Plaintext Matrix
            char[][] matrix = getMatrix(sb, key);
            String cipherText = permuteMatrix(matrix, keyInfo);
            System.out.println("CipherText=" + cipherText.toUpperCase());
      }

      // Transposed String
      public static String permuteMatrix(char[][] matrix, Map<Integer,
Integer> keyInfo) {
            StringBuilder sb = new StringBuilder();
            int colIndex = 0;
            for (int k = 1; k <= keyInfo.size(); k++) {
                colIndex = keyInfo.get(k);
                for (int row = 0; row < matrix.length; row++) {
                    sb.append(matrix[row][colIndex]);
                }
            }
            return sb.toString();
      }

      public static char[][] getMatrix(StringBuilder sb, int key) {
            int strLen = sb.length();
            int keyLen = (int) Math.log10(key) + 1;
            char[][] matrix = new char[strLen / keyLen][keyLen];
            for (int i = 0; i < matrix.length; i++) {
                for (int j = 0; j < matrix[0].length; j++) {
                    matrix[i][j] = sb.charAt((i * matrix[0].length) +
j);

      System.out.print(Character.toUpperCase(matrix[i][j]) + "\t");
                }
                System.out.println("\n");
```

```
            }
            return matrix;
        }

}
```

OUTPUT

```
<terminated> ColumnTransposition [Java Application] C:\Program
Enter the Plaintext:
wearethebestintheworld
Enter the Key:
34215

Plaintext Matrix

3       4       2       1       5

W       E       A       R       E

T       H       E       B       E

S       T       I       N       T

H       E       W       O       R

L       D       X       X       X

CipherText=RBNOXAEIWXWTSHLEHTEDEETRX
```

RESULT

The Column Transposition Cipher has been successfully implemented using Java.

Ex. No : 3 **DATA ENCRYPTION STANDARD**

Date :

AIM

To perform encryption and decryption using DES.

ALGORITHM

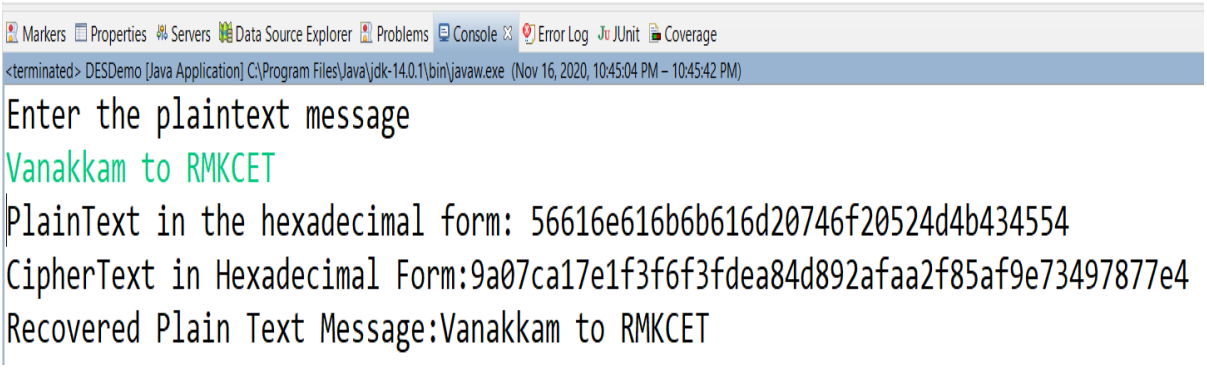
1. Using Java's in-built packages, DES algorithm is implemented.
2. Import necessary packages.
3. Add security provider.
4. Convert the input message to byte format for easy manipulation.
5. Get key and other cryptographic details using appropriate methods.
6. Perform encryption using appropriate methods.
7. To recover the plain text message, perform decryption using appropriate method.

/*Ex.No.3 DES*/

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.Key;
import java.security.NoSuchAlgorithmException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;

public class DESDemo {
	public static void main(String[] args) throws IOException,
NoSuchAlgorithmException {
		Key key = getKey();
		BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
		System.out.println("Enter the plaintext message");
		String plainTextMessage = br.readLine();
		byte[] cipherText = encrypt(plainTextMessage, key);
		BigInteger b = new BigInteger(1, cipherText);
		System.out.println("CipherText in Hexadecimal Form:" +
b.toString(16));
		System.out.println("Recovered Plain Text Message:" +
decrypt(cipherText, key));
	}

	public static Key getKey() throws NoSuchAlgorithmException {
		KeyGenerator kg = KeyGenerator.getInstance("DES");
		kg.init(56);
		Key k = kg.generateKey();
```

```
            return k;
      }

      public static byte[] encrypt(String plainTextMessage, Key key) {
            byte[] cipherText = null;
            try {
                  Cipher cipher = Cipher.getInstance("DES");
                  byte[] input = plainTextMessage.getBytes();
                  BigInteger b = new BigInteger(1, input);
                  System.out.println("PlainText in the hexadecimal form: "
+ b.toString(16));
                  cipher.init(Cipher.ENCRYPT_MODE, key);
                  cipherText = cipher.doFinal(input);
            } catch (InvalidKeyException | IllegalBlockSizeException |
BadPaddingException | NoSuchAlgorithmException
                        | NoSuchPaddingException e) {
                  e.printStackTrace();
            }
            return cipherText;
      }

      public static String decrypt(byte[] cipherTextMessage, Key key) {
            byte[] plainText = null;
            try {
                  Cipher cipher = Cipher.getInstance("DES");
                  cipher.init(Cipher.DECRYPT_MODE, key);
                  plainText = cipher.doFinal(cipherTextMessage);
            } catch (InvalidKeyException | IllegalBlockSizeException |
BadPaddingException | NoSuchAlgorithmException
                        | NoSuchPaddingException e) {
                  e.printStackTrace();
            }
            return new String(plainText);
      }

}
```

OUTPUT



RESULT

The Java program to perform encryption and decryption using DES was successfully implemented.

Ex. No : 4 **ADVANCED ENCRYPTION STANDARD**

Date :

AIM

To perform encryption and decryption using AES.

ALGORITHM

1. Using Java's in-built packages, AES algorithm is implemented.
2. Import necessary packages.
3. Add security provider.
4. Convert the input message to byte format for easy manipulation.
5. Get key and other cryptographic details using appropriate methods.
6. Perform encryption using appropriate methods.
7. To recover the plain text message, perform decryption using appropriate method.

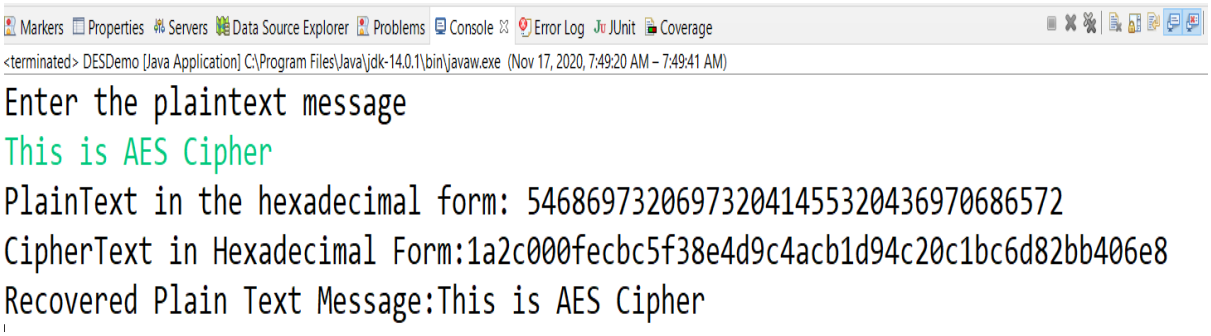
/*Ex.No.4 AES*/

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;

public class AESDemo {

	public static void main(String[] args) throws IOException,
NoSuchAlgorithmException {
		SecretKey key = getKey();
		BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
		System.out.println("Enter the plaintext message");
		String plainTextMessage = br.readLine();
		byte[] cipherText = encrypt(plainTextMessage, key);
		BigInteger b = new BigInteger(1, cipherText);
		System.out.println("CipherText in Hexadecimal Form:" +
b.toString(16));
		System.out.println("Recovered Plain Text Message:" +
decrypt(cipherText, key));
	}

	public static SecretKey getKey() throws NoSuchAlgorithmException {
		KeyGenerator kg = KeyGenerator.getInstance("AES");
		kg.init(128);
```

```
            SecretKey k = kg.generateKey();
            return k;
      }

      public static byte[] encrypt(String plainTextMessage, SecretKey key)
{
            byte[] cipherText = null;
            try {
                  Cipher cipher =
Cipher.getInstance("AES/ECB/PKCS5Padding");
                  byte[] input = plainTextMessage.getBytes();
                  BigInteger b = new BigInteger(1, input);
                  System.out.println("PlainText in the hexadecimal form: "
+ b.toString(16));
                  cipher.init(Cipher.ENCRYPT_MODE, key);
                  cipherText = cipher.doFinal(input);
            } catch (InvalidKeyException | IllegalBlockSizeException |
BadPaddingException | NoSuchAlgorithmException
                        | NoSuchPaddingException e) {
                  e.printStackTrace();
            }
            return cipherText;
      }

      public static String decrypt(byte[] cipherTextMessage, SecretKey
key) {
            byte[] plainText = null;
            try {
                  Cipher cipher =
Cipher.getInstance("AES/ECB/PKCS5Padding");
                  cipher.init(Cipher.DECRYPT_MODE, key);
                  plainText = cipher.doFinal(cipherTextMessage);
            } catch (InvalidKeyException | IllegalBlockSizeException |
BadPaddingException | NoSuchAlgorithmException
                        | NoSuchPaddingException e) {
                  e.printStackTrace();
```

```
            }

            return new String(plainText);
        }

}
```

OUTPUT



RESULT

The Java program to perform encryption and decryption using AES was successfully implemented.

Ex. No : 5 **RSA**

Date :

AIM

To perform encryption and decryption using RSA.

ALGORITHM

Version 1:

1. Using Java's in-built packages, RSA algorithm is implemented.
2. Import necessary packages.
3. Add security provider.
4. Convert the input message to byte format for easy manipulation.
5. Get key and other cryptographic details using appropriate methods.
6. Perform encryption using appropriate methods.
7. To recover the plain text message, perform decryption using appropriate method.

Version 2:

1. Use BigInteger class to perform RSA encryption and decryption.
2. Encryption

   $C = M^e \bmod n$
3. Decryption

   $M = C^d \bmod n$
4. Get encryption constant, prime numbers p and q from the user.
5. Calculate n and phi(n).
6. Calculate decryption constant.
7. Use various methods of BigInteger to perform modular arithmetic.

Version 3:

1. Use HTML for creating user interaction form and JavaScript for computing the formulae.
2. Encryption: $C = M^e \bmod n$
3. Decryption: $M = C^d \bmod n$
4. Get encryption constant, prime numbers p and q from the user.
5. Calculate n, phi(n), decryption constant.
6. Compute encryption and decryption using the above formulae.

/*Ex.No.5(a) RSA Using Java Crypto & Security Packages*/

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.Key;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;

public class RSADemo {

      public static void main(String[] args) throws IOException,
NoSuchAlgorithmException, NoSuchPaddingException,
                  InvalidKeyException, IllegalBlockSizeException,
BadPaddingException, NoSuchProviderException {
            KeyPair kp = getKeys();
            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
            System.out.println("Enter the plaintext message");
            String plainTextMessage = br.readLine();
            byte[] cipherText = encrypt(plainTextMessage, kp.getPublic());
            System.out.println("CipherText in the hexadecimal form: " +
new BigInteger(1, cipherText).toString(16));
            byte[] recoveredPlainText = decrypt(cipherText,
kp.getPrivate());
            System.out.println("Recovered PlainText : " + new
String(recoveredPlainText));
```

```
	}

	public static KeyPair getKeys() throws NoSuchAlgorithmException {
		KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
		kpg.initialize(2048);
		KeyPair kp = kpg.genKeyPair();
		return kp;
	}

	public static byte[] encrypt(String plainTextMessage, Key publicKey)
{
		byte[] cipherText = null;
		try {
			Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-
256AndMGF1Padding");
			byte[] input = plainTextMessage.getBytes();
			BigInteger b = new BigInteger(1, input);
			System.out.println("PlainText in the hexadecimal form: "
+ b.toString(16));
			cipher.init(Cipher.ENCRYPT_MODE, publicKey);
			cipherText = cipher.doFinal(input);
			// System.out.println(new String(cipherText));
		} catch (InvalidKeyException | IllegalBlockSizeException |
BadPaddingException | NoSuchAlgorithmException
				| NoSuchPaddingException e) {
			e.printStackTrace();
		}
		return cipherText;
	}

	public static byte[] decrypt(byte[] cipherText, Key privateKey) {
		byte[] plainText = null;
		try {
			Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-
256AndMGF1Padding");
			cipher.init(Cipher.DECRYPT_MODE, privateKey);
```

```
                plainText = cipher.doFinal(cipherText);
            } catch (NullPointerException | InvalidKeyException |
IllegalBlockSizeException | BadPaddingException
                        | NoSuchAlgorithmException |
NoSuchPaddingException e) {
                e.printStackTrace();
            }
            return plainText;
      }

}
```

/*Ex.No.5(b): RSA Using Java BigInteger*/

```
import java.io.BufferedReader;
import java.io.IOException;
import java.math.BigInteger;

public class RSABigIntegerDemo {

	public static void main(String[] args) throws IOException {
		// TODO Auto-generated method stub
		BufferedReader br = new BufferedReader(new
java.io.InputStreamReader(System.in));
		System.out.println("Enter Prime p:");
		BigInteger p = new BigInteger(br.readLine());
		System.out.println("Enter Prime q:");
		BigInteger q = new BigInteger(br.readLine());
		BigInteger phi =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
		BigInteger n = p.multiply(q);
		System.out.println("Enter Encryption Constant e:");
		BigInteger e = new BigInteger(br.readLine());
		BigInteger d = e.modInverse(phi);
		System.out.println("Enter Plaintext Message M:");
		BigInteger M = new BigInteger(br.readLine());
		BigInteger C = encrypt(M, e, n);
		System.out.println("Encrypted Message C=" + C);
		BigInteger M2 = decrypt(C, d, n);
		System.out.println("Decrypted Message M=" + M2);
	}

	public static BigInteger encrypt(BigInteger M, BigInteger e,
BigInteger n) {
		BigInteger C = M.modPow(e, n);
		return C;
	}
```

```
	public static BigInteger decrypt(BigInteger C, BigInteger d,
BigInteger n) {
		BigInteger M = BigInteger.ONE;
		M = C.modPow(d, n);
		return M;
	}

}
```

/*Ex.No.5(c) RSA Using HTML and JavaScript*/

```
<!-- index.html -->
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>RSA</title>
<link href="style.css" rel="stylesheet">
<script type="text/javascript" src="rsaencrypt.js"></script>
</head>
<body>
<form name="rsaform">
<fieldset>
<legend>Selecting Primes and Key Generation by Receiver:</legend>
<label for="p">Enter Prime p:</label>
<input type="text" id="p" name="p"><br>
<label for="q">Enter Prime q:</label>
<input type="text" id="q" name="q"><br>
<label for="e">Enter Encryption Constant e:</label>
<input type="text" id="e" name="e"><br>
</fieldset> <br>
<fieldset>
<legend>Encryption by Sender:</legend>
<label for="M">Enter Message:</label>
<input type="text" id="M" name="M"><br>
<input type="button" value="Encrypt" onClick="encrypt()">
<input type="reset" value="Clear"><br>
<label for="M">Encrypted Message:</label>
<input type="text" id="C" name="C"><br>
</fieldset><br>

<fieldset>
<legend>Decryption by Sender:</legend>
<label for="C2">Cipher Text Message Received:</label>
<input type="text" id="C2" name="C2"><br>
```

```
<input type="button" value="Calculate Decryption Constant d"
onClick="getDecryptionConstant()"><br>
<label for="d">Decryption Constant:</label>
<input type="text" id="d" name="d"><br>
<input type="button" value="Decrypt" onClick="decrypt()">
<input type="reset" value="Clear"><br>
<label for="M2">Decrypted Message:</label>
<input type="text" id="M2" name="M2">
</fieldset>

</form>
</body>
</html>
```

/*rsaencrypt.js*/

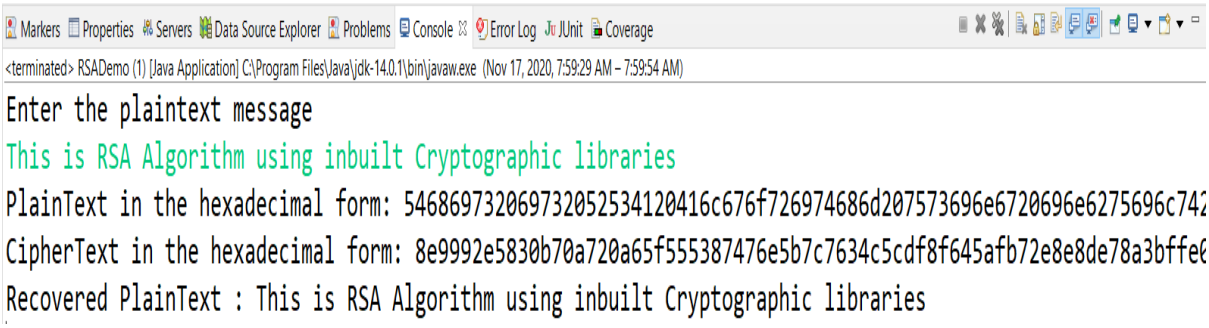
```
function getPrimes() {
      var p = parseInt(document.rsaform.p.value);
      var q = parseInt(document.rsaform.q.value);
      return [p,q];
}

function calculateN(){
      let p,q;
      [p,q] = getPrimes();
      var n = p * q;
      return n;
}

function calculatePhi(){
      let p,q;
      [p,q] = getPrimes();
      var phi = (p-1) * (q-1);
      return phi;
}

function getEncryptionConstant(){
```

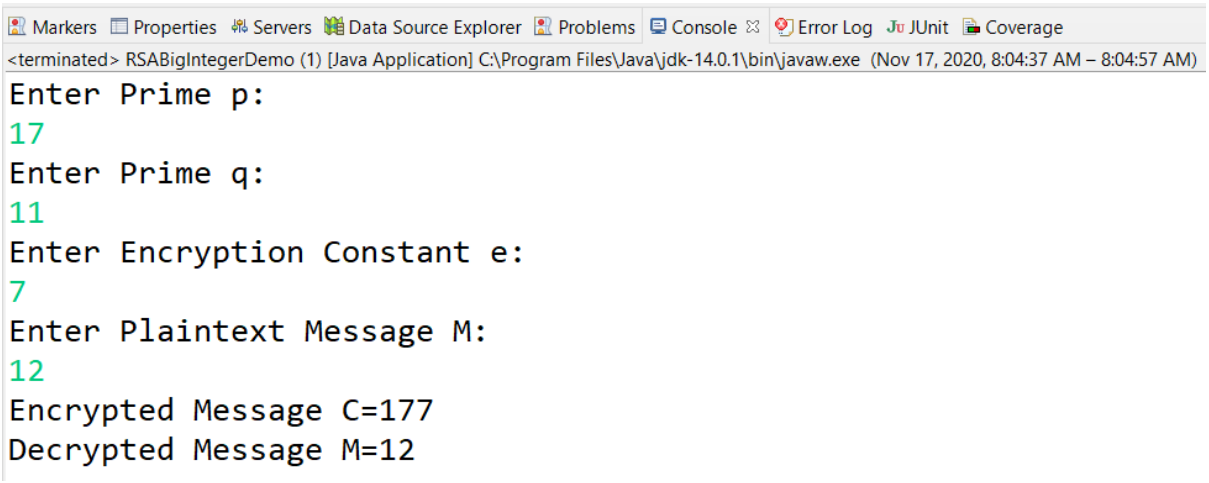
```
	var e = parseInt(document.rsaform.e.value);
	return e;
}

function getDecryptionConstant() {
	var phi = calculatePhi();
	var e = getEncryptionConstant();
	var d = (modInverse(e, phi) + phi) % phi;
	document.getElementById('d').value = d;
	return d;
}

function getPublicKey() {
	var e = getEncryptionConstant();
	var n = calculateN();
	return [e, n];
}

function getPrivateKey() {
	var d = getDecryptionConstant();
	var n = calculateN();
	return [d, n];
}

function encrypt() {
	let e, n;
	[e,n] = getPublicKey();
	var M = parseInt(document.rsaform.M.value);
	const C = powerMod(M, e, n);
	//alert("Encrypted Cipher Text="+ C);
	document.getElementById('C').value = C;
	document.getElementById('C2').value = C;
	return C;
}
```

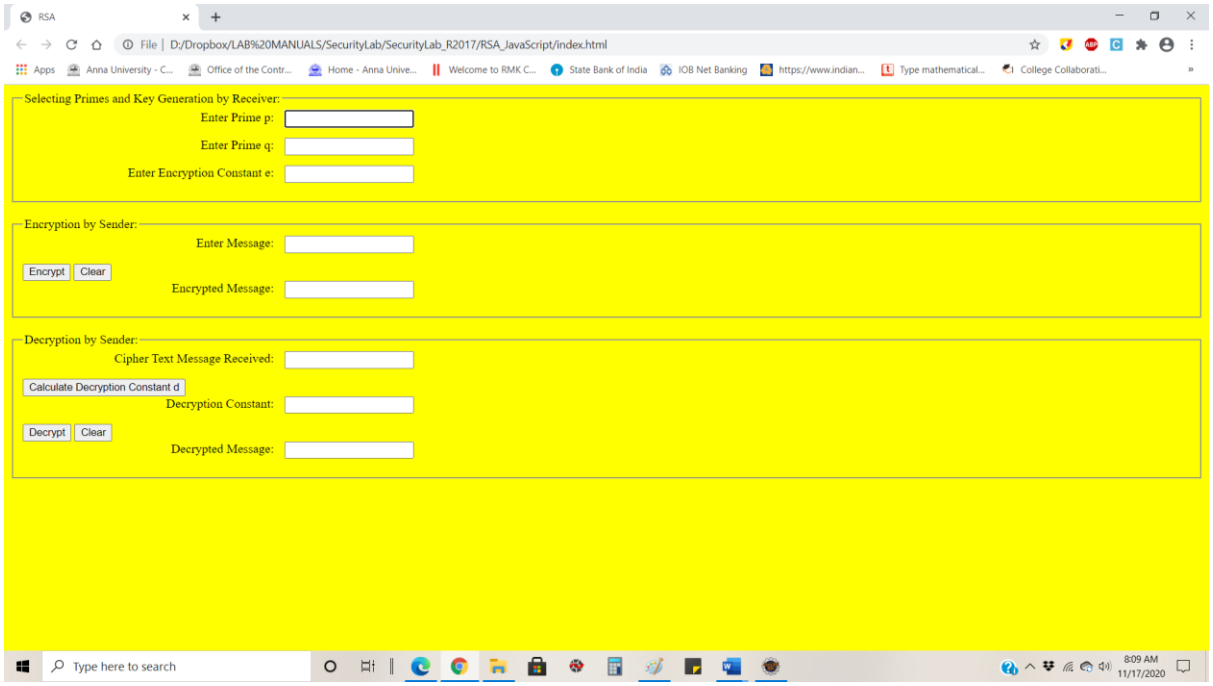
```
function decrypt(){
      let d, n;
      [d, n] = getPrivateKey();
      var C = document.getElementById('C2').value;
      const  M = powerMod(C, d, n);
      document.getElementById('M2').value = M;
}

/*Source: http://umaranis.com/2018/07/12/calculate-modular-exponentiation-
powermod-in-javascript-ap-n/*/
function powerMod(base, exponent, modulus) {
    if (modulus === 1) return 0;
    var result = 1;
    base = base % modulus;
    while (exponent > 0) {
        if (exponent % 2 === 1)  //odd number
            result = (result * base) % modulus;
        exponent = exponent >> 1; //divide by 2
        base = (base * base) % modulus;
    }
    return result;
}

function modInverse(b, m) {
      let A1, A2, A3;
      [A1, A2, A3] = [1, 0, m];
      let B1, B2, B3;
      [B1, B2, B3] = [0, 1, b];
      let T1, T2, T3;
      var Q;
      while (B3 != 0 || B3 != 1) {
            if (B3 == 0)
                  return 0;
            if (B3 == 1)
                  return B2;
            Q = Math.floor(A3 / B3);
```

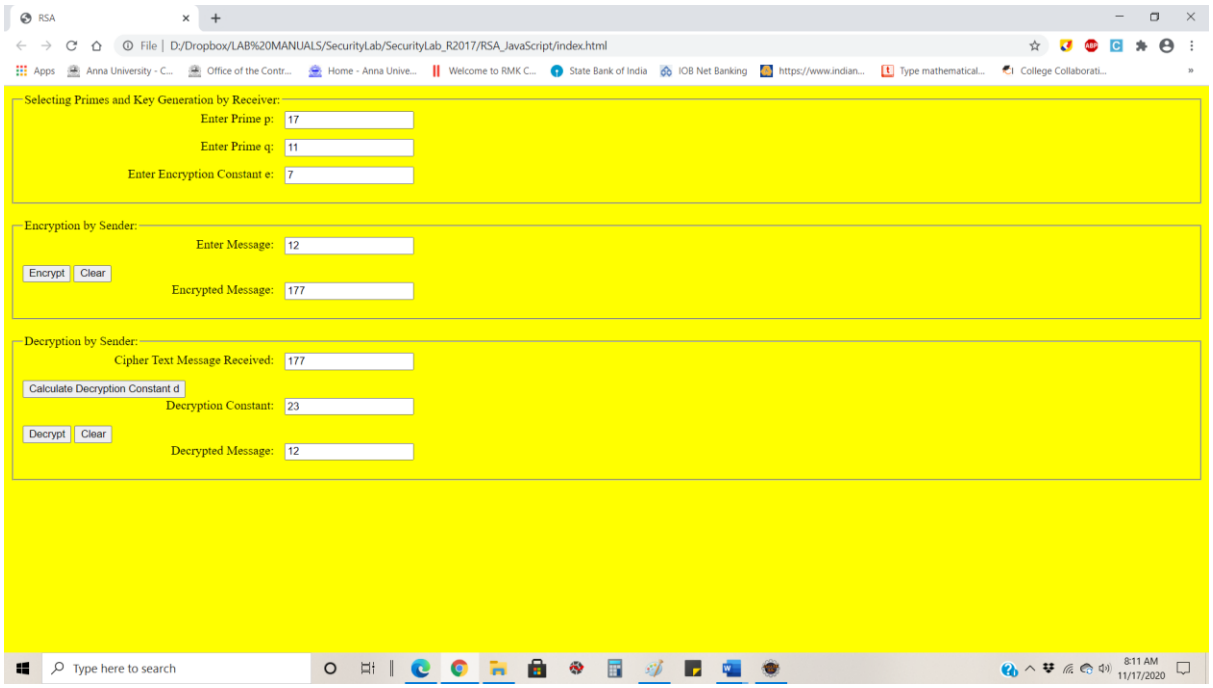
```
            [T1, T2, T3] = [A1 - Q * B1, A2 - Q * B2, A3 - Q * B3];
            [A1, A2, A3] = [B1, B2, B3];
            [B1, B2, B3] = [T1, T2, T3];
            }
            return 0;
}

/*style.css*/
body {
      background-color: yellow;
}

.clearfix::after {
  clear: both;
}

label {
      float:left;
      width:20em;
      text-align:right;
}

input[type=text] {
      margin-left: 1em;
      margin-bottom: 1em;
}
```

OUTPUT


Markers Properties Servers Data Source Explorer Problems Console Error Log JUnit Coverage
<terminated> RSADemo (1) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (Nov 17, 2020, 7:59:29 AM – 7:59:54 AM)
Enter the plaintext message
This is RSA Algorithm using inbuilt Cryptographic libraries
PlainText in the hexadecimal form: 546869732069732052534120416c676f726974686d207573696e6720696e6275696c742
CipherText in the hexadecimal form: 8e9992e5830b70a720a65f555387476e5b7c7634c5cdf8f645afb72e8e8de78a3bffe
Recovered PlainText : This is RSA Algorithm using inbuilt Cryptographic libraries


Markers Properties Servers Data Source Explorer Problems Console Error Log JUnit Coverage
<terminated> RSABigIntegerDemo (1) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (Nov 17, 2020, 8:04:37 AM – 8:04:57 AM)
Enter Prime p:
17
Enter Prime q:
11
Enter Encryption Constant e:
7
Enter Plaintext Message M:
12
Encrypted Message C=177
Decrypted Message M=12





RESULT

The RSA algorithm was successfully implemented by (i) Java crypto and security packages, (ii) Java BigInteger methods, and (iii) HTML and JavaScript successfully.

Ex. No : 6 **DIFFIE – HELLMAN KEY EXCHANGE**

Date :

AIM

To perform Diffie-Hellman Key Exchange.

ALGORITHM

1. Use various methods of BigInteger to perform modular arithmetic.
2. Get the prime number and one of its primitive root from the user.
3. Get the private keys of User A and User B.
4. Calculate public keys of User A and User B.
5. Calculate shared secret key.
6. Algorithm Formulae

   X_A - User A's Private Key

   X_B - User B's Private Key

   User A's Public Key: $Y_A = \alpha^{X_A} \bmod q$

   User B's Public Key: $Y_B = \alpha^{X_B} \bmod q$

   Shared Secret Key calculated by User A: $K = (Y_B)^{X_A} \bmod q$

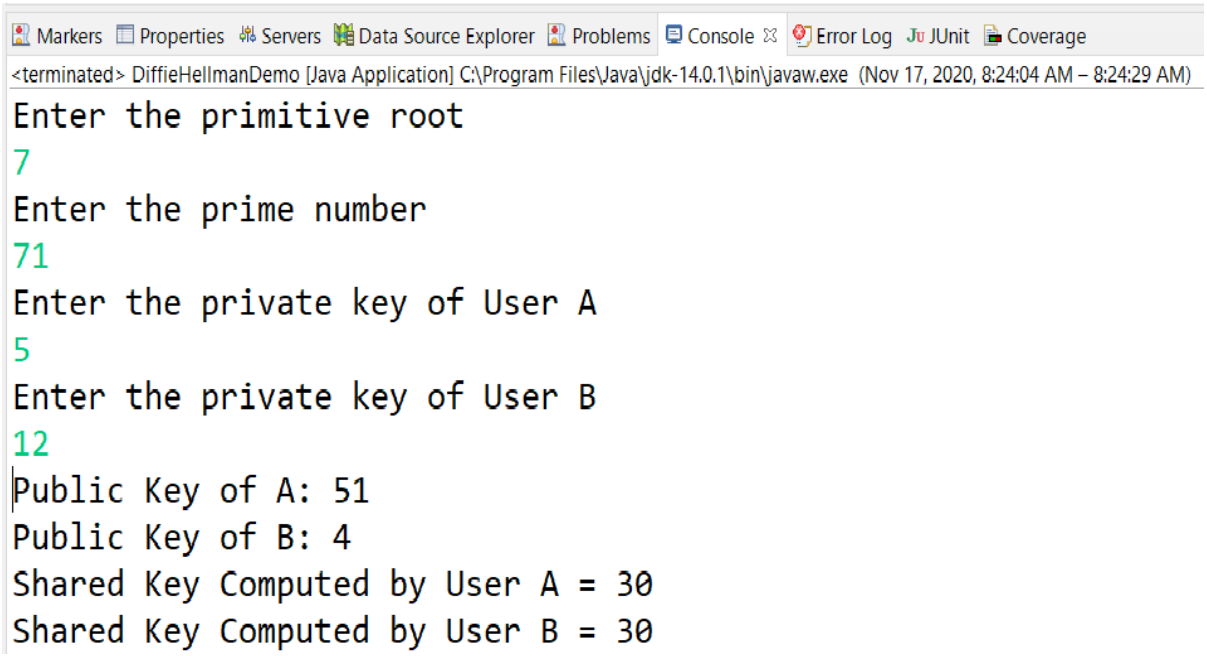
   Shared Secret Key calculated by User B: $K = (Y_A)^{X_B} \bmod q$

/*Ex.No.6 Diffie-Hellman*/

```
import java.math.BigInteger;
import java.util.Scanner;

public class DiffieHellmanDemo {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the primitive root");
		BigInteger g = sc.nextBigInteger();
		System.out.println("Enter the prime number");
		BigInteger q = sc.nextBigInteger();
		System.out.println("Enter the private key of User A");
		BigInteger xA = sc.nextBigInteger();
		System.out.println("Enter the private key of User B");
		BigInteger xB = sc.nextBigInteger();
		BigInteger yA = computePublicKey(g, xA, q);
		BigInteger yB = computePublicKey(g, xB, q);
		System.out.println("Public Key of A: " + yA);
		System.out.println("Public Key of B: " + yB);
		BigInteger K1 = computeSharedKey(xA, yB, q);
		BigInteger K2 = computeSharedKey(xB, yA, q);
		System.out.println("Shared Key Computed by User A = " + K1);
		System.out.println("Shared Key Computed by User B = " + K2);
		sc.close();
	}

	public static BigInteger computePublicKey(BigInteger g, BigInteger
xA, BigInteger q) {
		BigInteger Y = g.modPow(xA, q);
		return Y;
	}
```

```
	public static BigInteger computeSharedKey(BigInteger X, BigInteger
Y, BigInteger q) {
		BigInteger K = Y.modPow(X, q);
		return K;
	}

}
```

OUTPUT



RESULT

The Diffie-Hellman algorithm was successfully implemented using Java.

Ex. No : 7 **SHA-1**

Date :

AIM

To generate message digest value using SHA-1.

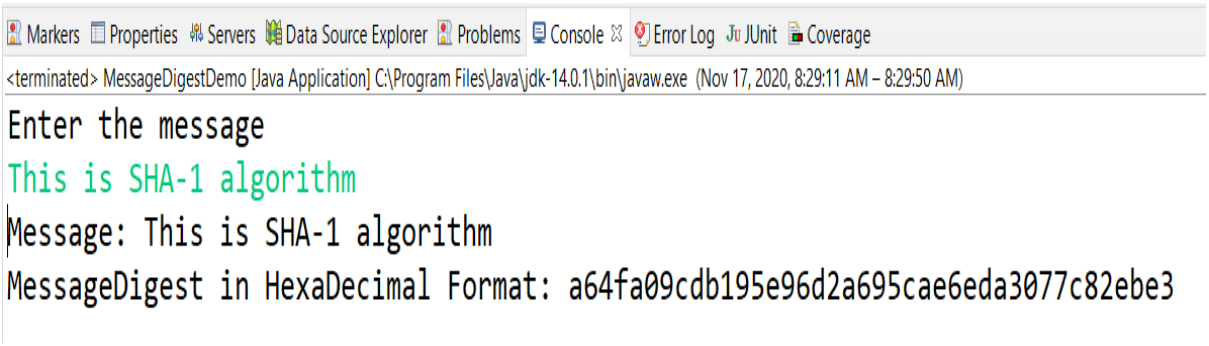
ALGORITHM

1. Using Java's in-built packages, SHA-1 algorithm is implemented.
2. Import necessary packages.
3. Convert the input message to byte format for easy manipulation.
4. Using appropriate methods, message digest is generated.
5. Message digest is displayed in hexadecimal format.

/*Ex.No.7 SHA-1*/

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class MessageDigestDemo {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the message");
		String message = sc.nextLine();
		byte[] hash = generateMessagedigest(message);
		BigInteger b = new BigInteger(1, hash);
		String hashValue = b.toString(16);
		System.out.println("MessageDigest in HexaDecimal Format: " +
hashValue);
	}

	public static byte[] generateMessagedigest(String message) {
		byte[] hash = null;
		byte[] input = message.getBytes();
		System.out.println("Message: " + new String(input));
		MessageDigest md;
		try {
			md = MessageDigest.getInstance("SHA-1");
			md.update(input);
			hash = md.digest();
		} catch (NoSuchAlgorithmException e) {
			e.printStackTrace();
		}
		return hash;
	}

}
```

OUTPUT



RESULT

The SHA-1 algorithm was successfully implemented using Java.

Ex. No : 8 **DIGITAL SIGNATURE STANDARD**

Date :

AIM

To generate and verify digital signature using DSS.

ALGORITHM

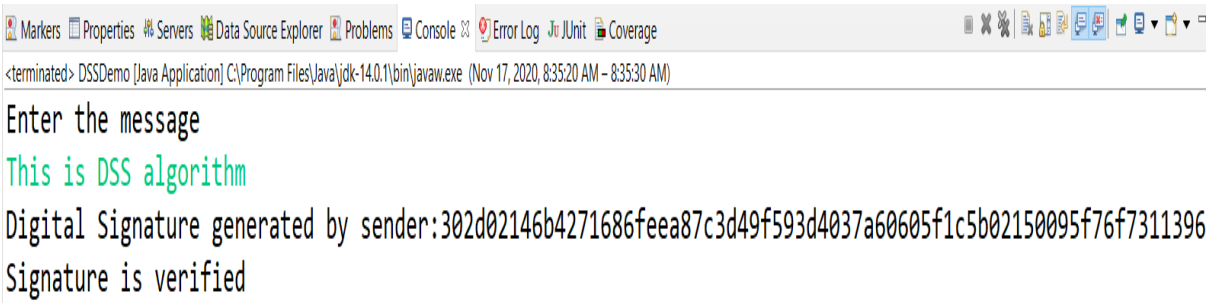
1. Using Java's in-built packages, DSS algorithm is implemented.
2. Import necessary packages.
3. Add security provider.
4. Convert the input message to byte format for easy manipulation.
5. Get cryptographic details using appropriate methods.
6. Sign and verify the message using appropriate methods.

/*Ex.No.8 Digital Signature Standard*/

```
import java.math.BigInteger;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.util.Scanner;

public class DSSDemo {

	public static void main(String[] args) throws Exception {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the message");
		String inputMessage = sc.nextLine();
		Signature signAlgorithm =
Signature.getInstance("SHA256withDSA");
		KeyPair kp = getKeys();
		byte[] signBytes = generateSignature(inputMessage,
kp.getPrivate(), signAlgorithm);
		BigInteger signedMessage = new BigInteger(1, signBytes);
		System.out.println("Digital Signature generated by sender:" +
signedMessage.toString(16));

		if (verifySignature(inputMessage, signBytes, kp.getPublic(),
signAlgorithm))
			System.out.println("Signature is verified");
		else
			System.out.println("Signature is not matching");
		sc.close();
	}

	public static byte[] generateSignature(String inputMessage,
PrivateKey privateKey, Signature signAlgorithm) {
```

```
            byte[] sigBytes = null;
            try {
                  signAlgorithm.initSign(privateKey);
                  byte[] message = inputMessage.getBytes();
                  signAlgorithm.update(message);
                  sigBytes = signAlgorithm.sign();
            } catch (Exception e) {
                  e.printStackTrace();
            }
            return sigBytes;
      }

      public static boolean verifySignature(String inputMessage, byte[]
sigBytes, PublicKey publicKey,
                  Signature signAlgorithm) {
            boolean result = true;
            try {
                  signAlgorithm.initVerify(publicKey);
                  signAlgorithm.update(inputMessage.getBytes());
                  result = signAlgorithm.verify(sigBytes);
            } catch (Exception e) {
                  e.printStackTrace();
            }
            return result;
      }

      public static KeyPair getKeys() throws NoSuchAlgorithmException {
            KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
            kpg.initialize(512);
            KeyPair kp = kpg.genKeyPair();
            return kp;
      }

}
```

OUTPUT



RESULT

The Digital Signature Standard algorithm was successfully implemented using Java.

Ex. No : 9 **SNORT**

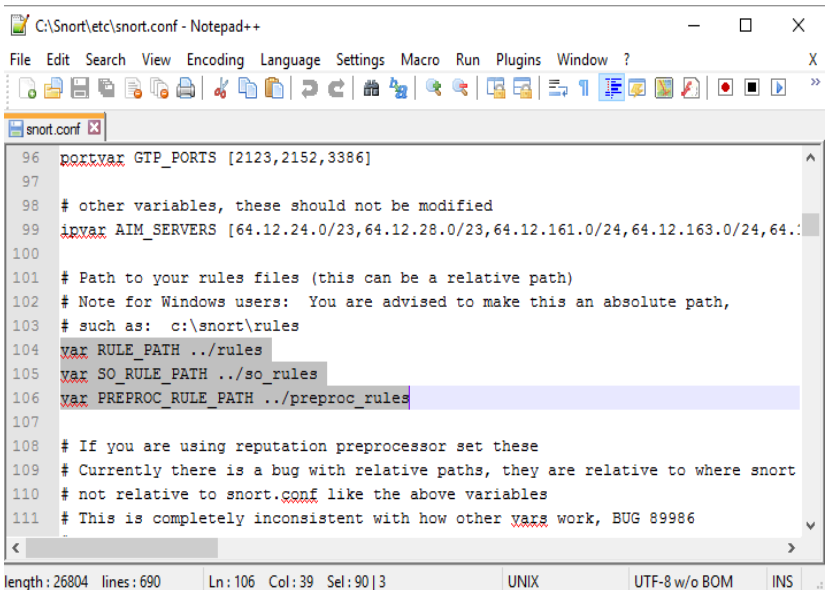
Date :

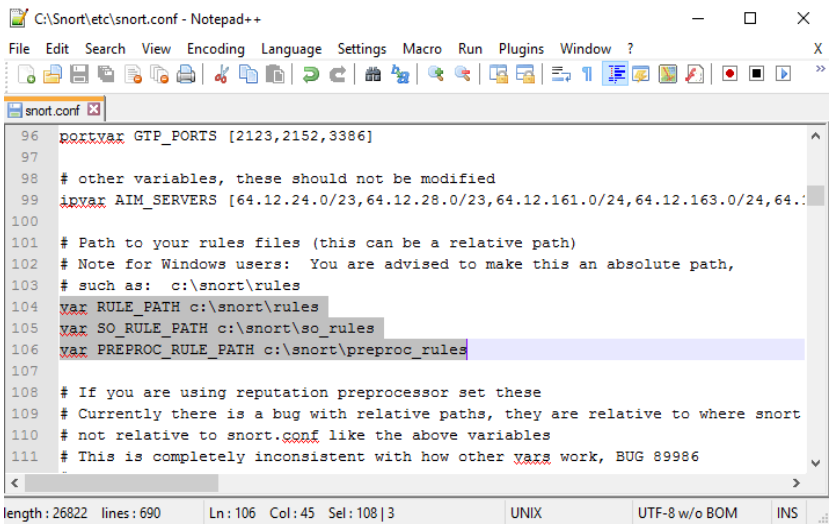
AIM

To demonstrate intrusion detection using Snort.

PROCEDURE

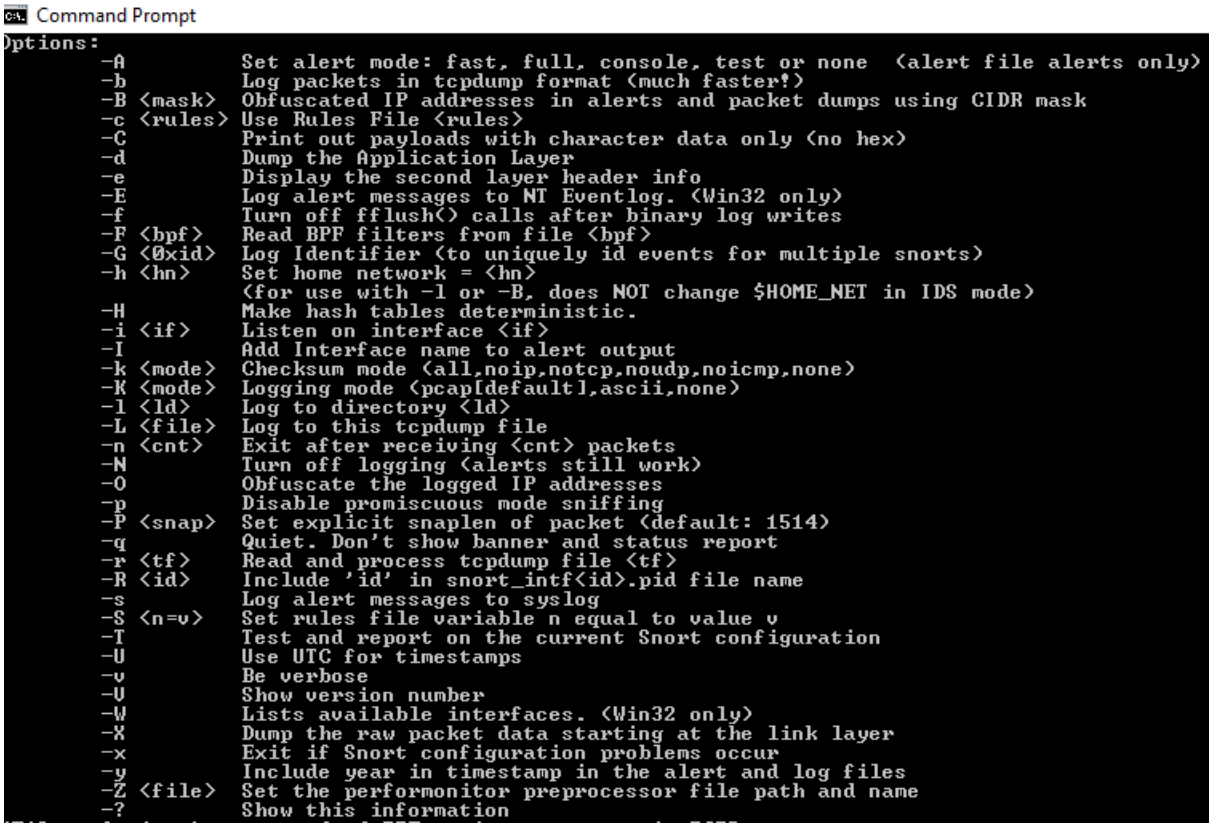
1. Install WinPcap libraries.
2. Install Snort
3. After installation, open ***snort.conf*** using any text editor. (It is present in the location C:\Snort\etc)
4. Edit the contents specifying rules path.





5. To use snort from any window path, set **path** variable as **C:\Snort\bin** under **Environment Variables**.
6. Run ***snort -W*** to see a list of interfaces available to Snort

7. Snort Options



8. Sniffer Mode

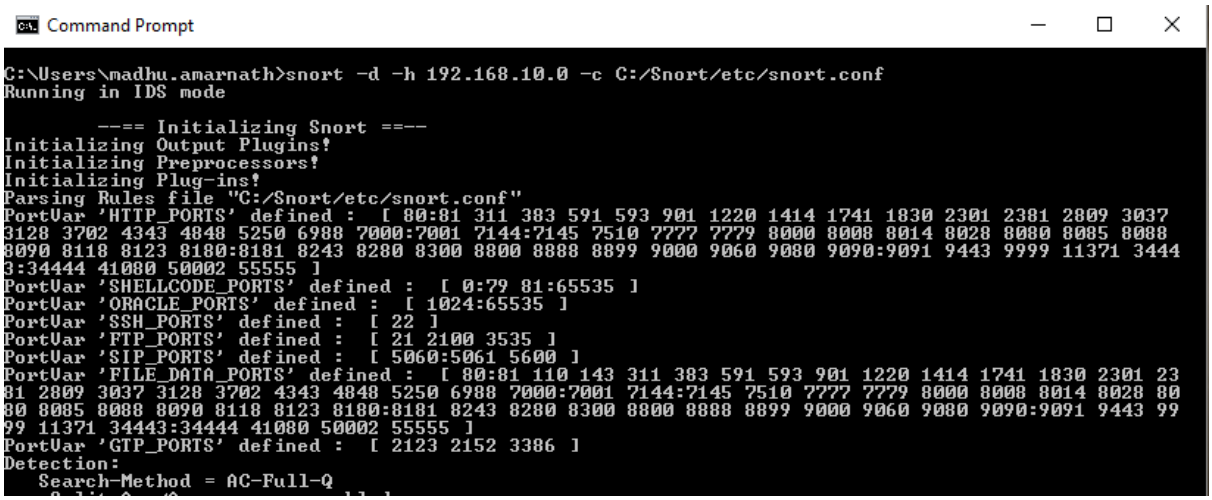
   snort -i 1 –v

9. Packet Logger Mode

   snort -l c:\snort\log -i1

10. Network IDS Mode

snort -d -h 192.168.10.0 -c C:/Snort/etc/snort.conf



RESULT

The Snort tool was used to demonstrate Intrusion Detection System.

Ex. No : 10 **Automated Attack and Penetration Tools: ZAP**

Date :

AIM

To perform penetration testing on a web application using ZAP.

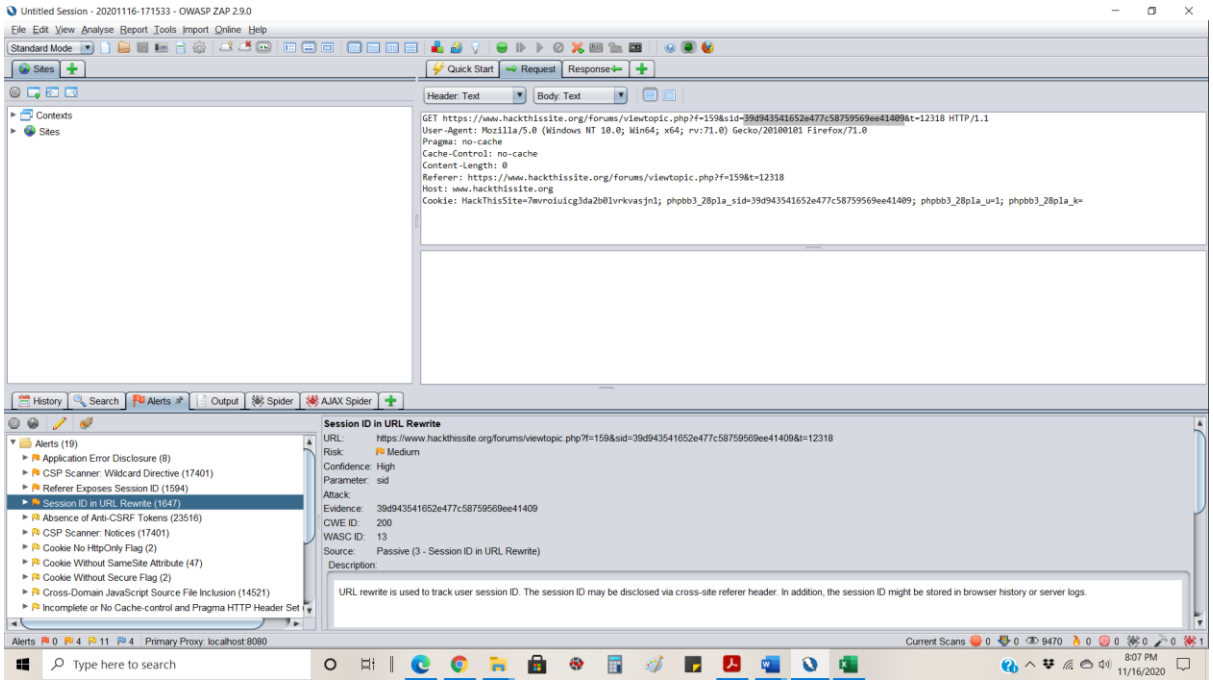
DESCRIPTION

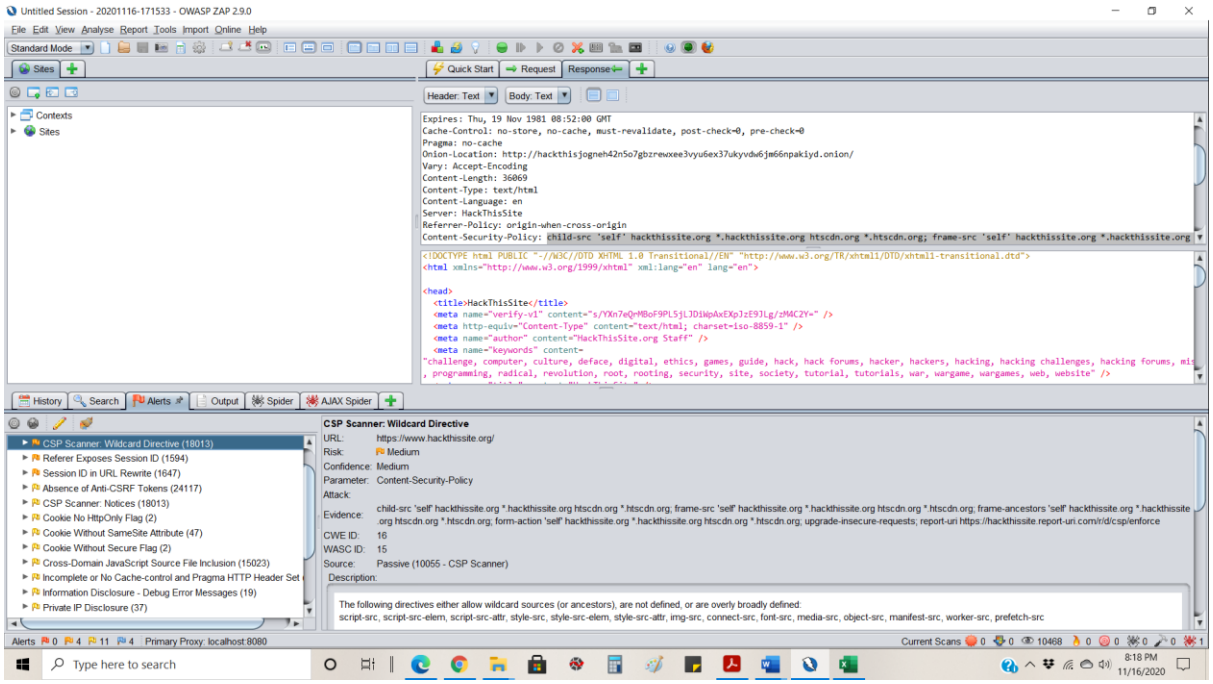
- Zed Attack Proxy (ZAP) is a free, open-source penetration testing tool being maintained under the umbrella of the Open Web Application Security Project (OWASP).
- ZAP is designed specifically for testing web applications and is both flexible and extensible.
- A penetration test, also known as a pen test or ethical hacking, is a simulated cyber- attack against your computer system to check for exploitable vulnerabilities.
- The Open Web Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software. (Website: https://owasp.org/)
- Requirements: The Windows and Linux versions require Java 8 or higher to run.
- Download Link: https://www.zaproxy.org/download/
- Getting Started: https://www.zaproxy.org/getting-started/

PROCEDURE

1. Start ZAP and click the Quick Start tab of the Workspace Window.
2. Click the large Automated Scan button.
3. In the URL to attack text box, enter the full URL of the web application you want to attack.
    a. Choose a website that you have permission to perform penetration testing.
    b. https://securitytrails.com/blog/vulnerable-websites-for-penetration-testing
    c. https://www.hackthissite.org/
4. Click the Attack.
5. Once the scanning is complete, click the “Alerts” tab to view the vulnerabilities details.

OUTPUT





RESULT

Thus, the OWASP ZAP tool was used to perform pen testing on a website and discover vulnerabilities successfully.

Ex. No : 11 **GMER – Rootkit Detection and Removal**

Date :

AIM

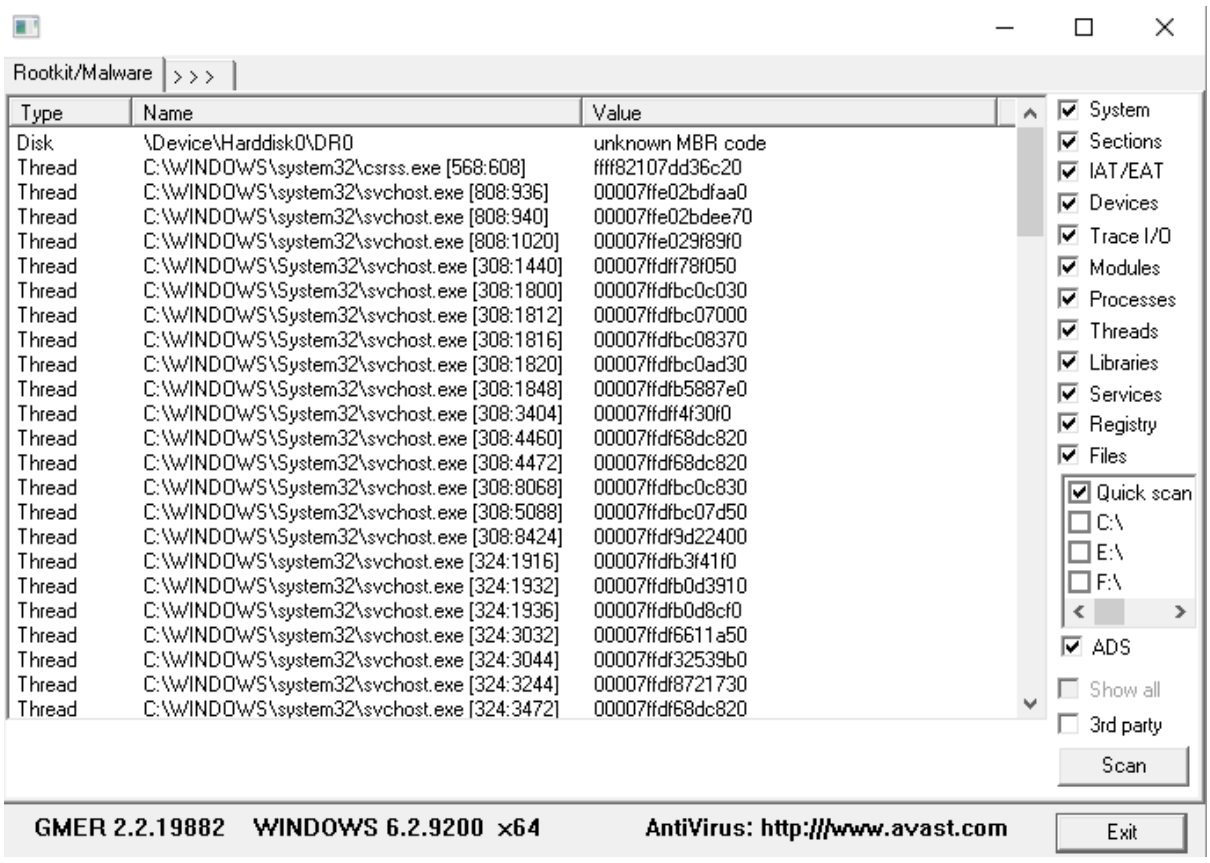
To use GMER application to detect and remove rootkits.

DESCRIPTION

- GMER is an application that detects and removes rootkits.
- A rootkit is a malicious software that allows an unauthorized user to have privileged access to a computer and to restricted areas of its software.
- A rootkit allows someone to maintain command and control over a computer without the computer user/owner knowing about it.

PROCEDURE

1. Click the **Scan** button.



RESULT

The GMER application was installed and run for detection of rootkits. No rootkits were found.

APPENDIX 1 – Important Links

1. Java JDK
   https://www.oracle.com/in/java/technologies/javase/javase-jdk8-downloads.html

2. Eclipse IDE
   https://www.eclipse.org/downloads/packages/release/2020-09/r/eclipse-ide-enterprise-java-developers

3. ZAP – Penetration Testing Tool
   https://www.zaproxy.org/download/

4. Snort - Network Intrusion Detection & Prevention System
   https://www.snort.org/downloads/snort/Snort_2_9_8_3_Installer.exe

5. GMER – Application to detect and remove rootkits
   http://www.gmer.net/

6. WinPcap – Windows Packet Capture Library
   http://www.winpcap.org/install/default.htm