EX.NO: ROLL.NO: 210701278

DATE: Digital Signature Algorithm

AIM:

Demonstrating digital signature generation and verification using RSA and SHA-256.

ALGORITHM:

1. Generate RSA key pair with a 2048-bit key size.
2. Create digital signature by hashing input with SHA-256 and encrypting with private key.
3. Verify signature by decrypting with public key and comparing hash with input.
4. Output signature in hexadecimal format.
5. Output verification result as boolean.

PROGRAM:

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.Signature;
import java.util.Scanner;

import javax.xml.bind.DatatypeConverter;

public class Dsa {

private static final String
SIGNING_ALGORITHM
= "SHA256withRSA";
private static final String RSA = "RSA";
private static Scanner sc;

public static byte[] Create_Digital_Signature(
byte[] input,
PrivateKey Key)
throws Exception
{
Signature signature
= Signature.getInstance(
SIGNING_ALGORITHM);
```

EX.NO: ROLL.NO: 210701278

DATE:

Implement the substitution technique Caesar Cipher

AIM:

To encrypt and decrypt a user-provided message using the Caesar Cipher technique with a specified shift value, ensuring confidentiality of communication.

ALGORITHM:

1. Start with the main function which prompts the user to enter the message and the shift value.
2. Read the message and shift value entered by the user.
3. Call the Caesar Cipher function passing the message and the shift value.
4. In the Caesar Cipher function:
    - Iterate through each character of the message.
    - Check if the character is an alphabet letter.
    - If it is, determine if it is uppercase or lowercase.
    - Apply the Caesar Cipher encryption algorithm by shifting the letter by the specified amount.
5. Print the encrypted message.

PROGRAM:

```
#include <stdio.h>
#include<ctype.h>
void caesarCipher(char message[], int shift);

int main() {
   char message[100];
   int shift;
   printf("Enter the message to encrypt: ");
   scanf("%s", message);
   printf("Enter the shift value: ");
   scanf("%d", &shift);
   caesarCipher(message, shift);
   printf("Encrypted message: %s\n", message);
   return 0;
}

void caesarCipher(char message[], int shift) {
   int i;
   for (i = 0; message[i] != '\0'; ++i) {
      char ch = message[i];
      if (isalpha(ch)) {
         if (isupper(ch)) {
```

EX.NO: ROLL.NO: 210701278

DATE:

Implement the Playfair Cipher technique

AIM:

To implement playfair cipher technique on the user input message.

ALGORITHM:

1. Initialize the Playfair key matrix based on the provided key, handling duplicates and 'J' substitution.
2. Preprocess the plaintext, removing non-alphabetic characters, converting to uppercase, and adding 'X' between consecutive identical characters.
3. Implement a method to retrieve the row and column positions of characters within the key matrix.
4. Encrypt the plaintext by iterating through character pairs, applying Playfair Cipher rules based on character positions, and constructing the ciphertext.
5. Accept user input for the key and plaintext, instantiate the Playfair Cipher, encrypt the plaintext, and output the ciphertext.

PROGRAM:

```
import java.util.*;

class PlayfairCipher {
   private char[][] keyMatrix;

   public PlayfairCipher(String key) {
      key = key.replaceAll("[Jj]", "I").toUpperCase();
      Set<Character> uniqueChars = new LinkedHashSet<>();
      for (char c : key.toCharArray()) {
         if (!Character.isLetter(c)) continue;
         uniqueChars.add(c);
      }
      StringBuilder keyBuilder = new StringBuilder();
      for (char c : uniqueChars) {
         keyBuilder.append(c);
      }
      String cleanKey = keyBuilder.toString();
      String alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ";
      for (char c : cleanKey.toCharArray()) {
         alphabet = alphabet.replace(Character.toString(c), "");
      }
      cleanKey += alphabet;
      keyMatrix = new char[5][5];
      int row = 0, col = 0;
```

```
KeyPair keyPair
= Generate_RSA_KeyPair();

// Function Call
byte[] signature
= Create_Digital_Signature(
input.getBytes(),
keyPair.getPrivate());

System.out.println(
"Signature Value:\n "
+ DatatypeConverter
.printHexBinary(signature));

System.out.println(
"Verification: "
+ Verify_Digital_Signature(
input.getBytes(),
signature, keyPair.getPublic()));
}
}
```

OUTPUT:

RESULT: