OTP

public class otp {

    public static void main(String[] args) {

        String otp="1804";

        for (int i = 1; i < 10000; i++) {

            // Format the number to have four digits

            String formattedNumber = String.format("%04d", i);

            if(formattedNumber.equals(otp)){

                System.out.println("Otp is cracked...The otp is "+formattedNumber);

                break;

            }

            System.out.println(formattedNumber);

        }

    }

}

Q2.2 power 20

public class KeyTime {

    public static void main(String[] args) {

        long startTime = System.nanoTime(); // Record start time

        for (long i = 0; i <= (1L << 20); i++) {

            // Convert the number to its 64-bit binary representation

            String binaryString = Long.toBinaryString(i);

            // Pad with leading zeros to ensure 64 bits

            String paddedBinary = String.format("%64s", binaryString).replace(' ', '0');

            System.out.println(paddedBinary);

        }

        long endTime = System.nanoTime(); // Record end time

        double elapsedTimeInSeconds = (endTime - startTime) / 1e9; // Convert nanoseconds to seconds

        System.out.println("Time taken: " + elapsedTimeInSeconds + " seconds");

    }

}

Q3.

DES

import java.util.Scanner;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import java.nio.charset.StandardCharsets;

import java.util.Base64;

public class DES {

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

        Scanner sc=new Scanner(System.in);

        System.out.print("Enter the aadharNumber: ");

        String aadharNumber = sc.nextLine();

        System.out.println();

        KeyGenerator keyGenerator = KeyGenerator.getInstance("DES");

        SecretKey secretKey = keyGenerator.generateKey();

        System.out.println("The key is "+secretKey);

        Cipher cipher = Cipher.getInstance("DES");

        cipher.init(Cipher.ENCRYPT\_MODE, secretKey);

        byte[] encryptedBytes = cipher.doFinal(aadharNumber.getBytes(StandardCharsets.UTF\_8));

        String encryptedAadhar = Base64.getEncoder().encodeToString(encryptedBytes);

        System.out.println("Encrypted Aadhar: " + encryptedAadhar);

        cipher.init(Cipher.DECRYPT\_MODE, secretKey);

        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedAadhar));

        String decryptedAadhar = new String(decryptedBytes, StandardCharsets.UTF\_8);

        System.out.println("Decrypted Aadhar: " + decryptedAadhar);

        sc.close();

    }

}

Triple\_DES

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.nio.charset.StandardCharsets;

import java.util.Arrays;

import java.util.Base64;

public class Triple\_DES {

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

        String aadharNumber = "391300351234";

        // Generate three unique DES keys

        SecretKey key1 = generateDESkey();

        SecretKey key2 = generateDESkey();

        SecretKey key3 = generateDESkey();

        // Encrypt using Triple DES

        String encryptedAadhar = tripleDESEncrypt(aadharNumber, key1, key2, key3);

        System.out.println("Encrypted Aadhar: " + encryptedAadhar);

        // Decrypt using the same keys

        String decryptedAadhar = tripleDESDecrypt(encryptedAadhar, key1, key2, key3);

        System.out.println("Decrypted Aadhar: " + decryptedAadhar);

    }

    private static SecretKey generateDESkey() throws Exception {

        // Generate a 168-bit DES key (equivalent to 3 \* 56 bits)

        KeyGenerator keyGenerator = KeyGenerator.getInstance("DESede");

        keyGenerator.init(168); // Initialize with a key size of 168 bits

        return keyGenerator.generateKey();

    }

    private static String tripleDESEncrypt(String plaintext, SecretKey key1, SecretKey key2, SecretKey key3) throws Exception {

        Cipher cipher = Cipher.getInstance("DESede/ECB/PKCS5Padding");

        // Concatenate all three keys

        byte[] concatenatedKeyBytes = new byte[24];

        System.arraycopy(key1.getEncoded(), 0, concatenatedKeyBytes, 0, 8);

        System.arraycopy(key2.getEncoded(), 0, concatenatedKeyBytes, 8, 8);

        System.arraycopy(key3.getEncoded(), 0, concatenatedKeyBytes, 16, 8);

        SecretKey fullKey = new SecretKeySpec(concatenatedKeyBytes, "DESede");

        cipher.init(Cipher.ENCRYPT\_MODE, fullKey);

        byte[] encryptedBytes = cipher.doFinal(plaintext.getBytes(StandardCharsets.UTF\_8));

        return Base64.getEncoder().encodeToString(encryptedBytes);

    }

    private static String tripleDESDecrypt(String ciphertext, SecretKey key1, SecretKey key2, SecretKey key3) throws Exception {

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

    Cipher cipher = Cipher.getInstance("DESede/ECB/PKCS5Padding");

    // Concatenate all three keys

    byte[] concatenatedKeyBytes = new byte[24];

    System.arraycopy(key1.getEncoded(), 0, concatenatedKeyBytes, 0, 8);

    System.arraycopy(key2.getEncoded(), 0, concatenatedKeyBytes, 8, 8);

    System.arraycopy(key3.getEncoded(), 0, concatenatedKeyBytes, 16, 8);

    SecretKey fullKey = new SecretKeySpec(concatenatedKeyBytes, "DESede");

    cipher.init(Cipher.DECRYPT\_MODE, fullKey);

    byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(ciphertext));

    System.out.println("Decrypted bytes: " + Arrays.toString(decryptedBytes));

    String decryptedString = new String(decryptedBytes, StandardCharsets.UTF\_8);

    System.out.println("Decrypted Aadhar: " + decryptedString);

    return decryptedString;

}

}

AES

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import java.nio.charset.StandardCharsets;

import java.util.Base64;

public class AES {

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

        String message = "391300351234";

        // Generate AES key

        SecretKey key = generateAESKey();

        // Encrypt using AES

        String encryptedMessage = aesEncrypt(message, key);

        System.out.println("Encrypted message: " + encryptedMessage);

        // Decrypt using the same key

        String decryptedMessage = aesDecrypt(encryptedMessage, key);

        System.out.println("Decrypted message: " + decryptedMessage);

    }

    private static SecretKey generateAESKey() throws Exception {

        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");

        keyGenerator.init(256); // Initialize with a key size of 256 bits

        return keyGenerator.generateKey();

    }

    private static String aesEncrypt(String plaintext, SecretKey key) throws Exception {

        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

        cipher.init(Cipher.ENCRYPT\_MODE, key);

        byte[] encryptedBytes = cipher.doFinal(plaintext.getBytes(StandardCharsets.UTF\_8));

        return Base64.getEncoder().encodeToString(encryptedBytes);

    }

    private static String aesDecrypt(String ciphertext, SecretKey key) throws Exception {

        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

        cipher.init(Cipher.DECRYPT\_MODE, key);

        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(ciphertext));

        return new String(decryptedBytes, StandardCharsets.UTF\_8);

    }

}

Q4.

import java.util.Scanner;

public class CaesarCipher {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Input word to encrypt

        System.out.print("Enter a word to encrypt: ");

        String word = scanner.nextLine().toUpperCase();

        // Encrypt the word using a Caesar cipher with a random key (for demonstration)

        int key = 5; // Example key (should be randomly generated in real scenarios)

        String encryptedWord = encrypt(word, key);

        System.out.println("Encrypted word: " + encryptedWord);

        // Attempt to crack the key using brute force

        System.out.println("Attempting to crack the key using brute force:");

        for (int i = 0; i < 26; i++) { // Brute force all possible keys

            String decryptedWord = decrypt(encryptedWord, i);

            System.out.println("Key " + i + ": " + decryptedWord);

        }

    }

    private static String encrypt(String word, int key) {

        StringBuilder encrypted = new StringBuilder();

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

            if (Character.isLetter(ch)) {

                char shifted = (char) ('A' + (ch - 'A' + key) % 26);

                encrypted.append(shifted);

            } else {

                encrypted.append(ch);

            }

        }

        return encrypted.toString();

    }

    private static String decrypt(String encryptedWord, int key) {

        return encrypt(encryptedWord, 26 - key); // Decrypt by shifting back by the key

    }

}