AES PROGRAM :

import javax.crypto.Cipher;

import javax.crypto.spec.SecretKeySpec;

import java.util.Base64;

public class RijndaelExample {

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

String originalString = "Hello, World!";

String key = "ThisIsASecretKey";

byte[] encryptedBytes = encrypt(originalString, key);

System.out.println("Encrypted: " + Base64.getEncoder().encodeToString(encryptedBytes));

String decryptedString = decrypt(encryptedBytes, key);

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

}

public static byte[] encrypt(String plainText, String key) throws Exception {

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

SecretKeySpec secretKey = new SecretKeySpec(key.getBytes(), "AES");

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

return cipher.doFinal(plainText.getBytes());

}

public static String decrypt(byte[] cipherText, String key) throws Exception {

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

SecretKeySpec secretKey = new SecretKeySpec(key.getBytes(), "AES");

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

byte[] decryptedBytes = cipher.doFinal(cipherText);

return new String(decryptedBytes);

}

}

BLOWFISH PROGRAM :

import javax.crypto.Cipher;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.util.Base64;

public class BlowfishExample {

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

String originalString = "Hello, World!";

String keyString = "ThisIsASecretKey";

byte[] encryptedBytes = encrypt(originalString, keyString);

System.out.println("Encrypted: " + Base64.getEncoder().encodeToString(encryptedBytes));

String decryptedString = decrypt(encryptedBytes, keyString);

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

}

public static byte[] encrypt(String plainText, String key) throws Exception {

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

SecretKey secretKey = new SecretKeySpec(key.getBytes(), "Blowfish");

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

return cipher.doFinal(plainText.getBytes());

}

public static String decrypt(byte[] cipherText, String key) throws Exception {

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

SecretKey secretKey = new SecretKeySpec(key.getBytes(), "Blowfish");

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

byte[] decryptedBytes = cipher.doFinal(cipherText);

return new String(decryptedBytes);

}

}

RSA PROGRAM :

import java.security.\*;

import javax.crypto.Cipher;

import java.util.Base64;

public class RSAExample {

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

KeyPair keyPair = generateKeyPair();

PublicKey publicKey = keyPair.getPublic();

PrivateKey privateKey = keyPair.getPrivate();

// Original message

String originalMessage = "Hello, World!";

System.out.println("Original Message: " + originalMessage);

// Encrypt the message using public key

byte[] encryptedMessage = encrypt(originalMessage, publicKey);

// System.out.println("Encrypted Message: " + new String(encryptedMessage));

System.out.println("Encrypted: " + Base64.getEncoder().encodeToString(encryptedMessage));

String decryptedMessage = decrypt(encryptedMessage, privateKey);

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

}

public static KeyPair generateKeyPair() throws Exception {

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

keyPairGenerator.initialize(2048); // Key size 2048 bits

return keyPairGenerator.generateKeyPair();

}

public static byte[] encrypt(String plainText, PublicKey publicKey) throws Exception {

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

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

return cipher.doFinal(plainText.getBytes());

}

public static String decrypt(byte[] cipherText, PrivateKey privateKey) throws Exception {

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

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

byte[] decryptedBytes = cipher.doFinal(cipherText);

return new String(decryptedBytes);

}

}

DES PROGRAM :

import javax.crypto.Cipher;

import javax.crypto.spec.SecretKeySpec;

import java.util.Base64;

public class DESExample {

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

String originalString = "Hello, World!";

String key = "ThisIsASecretKey";

byte[] encryptedBytes = encrypt(originalString, key);

System.out.println("Encrypted: " + Base64.getEncoder().encodeToString(encryptedBytes));

String decryptedString = decrypt(encryptedBytes, key);

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

}

public static byte[] encrypt(String plainText, String key) throws Exception {

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

SecretKeySpec secretKey = new SecretKeySpec(key.getBytes(), "DES");

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

return cipher.doFinal(plainText.getBytes());

}

public static String decrypt(byte[] cipherText, String key) throws Exception {

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

SecretKeySpec secretKey = new SecretKeySpec(key.getBytes(), "DES");

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

byte[] decryptedBytes = cipher.doFinal(cipherText);

return new String(decryptedBytes);

}

}

DIFFIEHELLMAN KEY EXCHANGE PROGRAM :

import java.math.BigInteger;

import java.security.SecureRandom;

import java.util.Scanner;

public class DiffieHellman {

// Generate a random prime number of the specified bit length

private static BigInteger generatePrime(int bitLength) {

SecureRandom random = new SecureRandom();

return BigInteger.probablePrime(bitLength, random);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Generate a large prime number (p) and a primitive root modulo p (g)

int bitLength = 512; // or another suitable length like 2048

BigInteger p = generatePrime(bitLength);

BigInteger g = BigInteger.valueOf(2); // Choosing 2 as the primitive root (in practice, this should be verified)

// Display the chosen prime (p) and generator (g)

System.out.println("Prime (p): " + p);

System.out.println("Generator (g): " + g);

// User A's private key

System.out.print("Enter User A's private key (a): ");

BigInteger a = new BigInteger(scanner.next());

// User B's private key

System.out.print("Enter User B's private key (b): ");

BigInteger b = new BigInteger(scanner.next());

// Calculate public keys

BigInteger A = g.modPow(a, p);

BigInteger B = g.modPow(b, p);

// Display the public keys

System.out.println("User A's Public Key (A): " + A);

System.out.println("User B's Public Key (B): " + B);

// Calculate the shared secret keys

BigInteger sharedKeyA = B.modPow(a, p); // A calculates the shared key

BigInteger sharedKeyB = A.modPow(b, p); // B calculates the shared key

// Display the shared secret keys

System.out.println("User A's Shared Secret Key: " + sharedKeyA);

System.out.println("User B's Shared Secret Key: " + sharedKeyB);

scanner.close();

// Verify that both shared keys are the same

if (sharedKeyA.equals(sharedKeyB)) {

System.out.println("Key Exchange Successful. Shared secret is: " + sharedKeyA);

} else {

System.out.println("Key Exchange Failed.");

}

}

}

SH1 PROGRAM :

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.Scanner;

public class SHA1Digest {

// Method to convert byte array to hex string

private static String bytesToHex(byte[] bytes) {

StringBuilder hexString = new StringBuilder();

for (byte b : bytes) {

String hex = Integer.toHexString(0xff & b);

if (hex.length() == 1) hexString.append('0');

hexString.append(hex);

}

return hexString.toString();

}

// Method to compute SHA-1 hash

public static String computeSHA1(String input) {

try {

// Create a SHA-1 digest instance

MessageDigest sha1 = MessageDigest.getInstance("SHA-1");

// Update the digest with the input bytes

byte[] hashBytes = sha1.digest(input.getBytes());

// Convert the byte array to a hex string

return bytesToHex(hashBytes);

} catch (NoSuchAlgorithmException e) {

// Handle the exception if SHA-1 algorithm is not available

throw new RuntimeException(e);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read input text from the user

System.out.print("Enter text to hash: ");

String inputText = scanner.nextLine();

// Compute the SHA-1 hash of the input text

String sha1Hash = computeSHA1(inputText);

// Display the SHA-1 hash

System.out.println("SHA-1 Hash: " + sha1Hash);

scanner.close();

}

}