Ex-1

```
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad, unpad
# Key must be 8 bytes long for DES
key = b'8bytekey'
# Create a DES cipher object in ECB mode
cipher = DES.new(key, DES.MODE_ECB)
# Data to encrypt
data = b'HELLOWORLD DES'
# Pad data to make it multiple of DES block size (8 bytes)
padded_data = pad(data, DES.block_size)
# Encrypt the padded data
encrypted_data = cipher.encrypt(padded_data)
print("Encrypted:", encrypted_data.hex())
# Decrypt the encrypted data
decrypted_padded = cipher.decrypt(encrypted_data)
# Remove padding
decrypted_data = unpad(decrypted_padded, DES.block_size)
# Display decrypted text
print("Decrypted:", decrypted_data.decode())
```

Ex-2

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
# Generate random AES key (16 bytes = 128 bits)
key = get_random_bytes(16)
```

```
# Generate random IV (Initialization Vector)
iv = get_random_bytes(16)
# Create AES cipher object in CBC mode
cipher = AES.new(key, AES.MODE_CBC, iv)
# Data to encrypt
data = b"This is AES encryption"
# Pad data to match AES block size (16 bytes)
padded_data = pad(data, AES.block_size)
# Encrypt data
ciphertext = cipher.encrypt(padded_data)
print("Encrypted:", ciphertext.hex())
# --- Decryption process ---
cipher_dec = AES.new(key, AES.MODE_CBC, iv)
decrypted_padded = cipher_dec.decrypt(ciphertext)
decrypted_data = unpad(decrypted_padded, AES.block_size)
print("Decrypted:", decrypted_data.decode())
```

Ex-3

```
from math import gcd
def RSA(p: int, q: int, message: int):
    # Step 1: Compute n and φ(n)
    n = p * q
    t = (p - 1) * (q - 1)
    # Step 2: Find e (public key) such that 1 < e < t and gcd(e, t) = 1
    for i in range(2, t):
        if gcd(i, t) == 1:
            e = i
```

```
            break
    # Step 3: Find d (private key) such that (d * e) % t = 1
    j = 1
    while True:
        if (j * e) % t == 1:
            d = j
            break
        j += 1
    # Step 4: Encryption (C = message^e mod n)
    C = pow(message, e, n)
    print(f"Encrypted data (ciphertext) is: {C}")
    # Step 5: Decryption (M = C^d mod n)
    M = pow(C, d, n)
    print(f"Decrypted data (original message) is: {M}")
print("Example 1:")
RSA(p=53, q=59, message=89)
print("\nExample 2:")
RSA(p=3, q=7, message=12)
```

Ex-4

```
from Crypto.Cipher import Blowfish
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
# Step 1: Define secret key (must be 4–56 bytes)
key = b"My Secret Key"
# Step 2: Define block size and IV
bs = Blowfish.block_size
```

```
iv = get_random_bytes(bs)
# Step 3: Create cipher object for encryption
cipher_encrypt = Blowfish.new(key, Blowfish.MODE_CBC, iv)
# Step 4: Plain text data
data = b"HELLO BLOWFISH CBC MODE"
# Step 5: Pad data to block size
padded_data = pad(data, bs)
# Step 6: Encrypt data
encrypted = cipher_encrypt.encrypt(padded_data)
print("Encrypted:", encrypted.hex())
# Step 7: Create cipher object for decryption (same key & IV)
cipher_decrypt = Blowfish.new(key, Blowfish.MODE_CBC, iv)
# Step 8: Decrypt and unpad
decrypted_padded = cipher_decrypt.decrypt(encrypted)
decrypted = unpad(decrypted_padded, bs)
print("Decrypted:", decrypted.decode('utf-8'))
```

Ex-5

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.GCMParameterSpec;
import java.util.Base64;
public class AESGCMExample {
    // AES-GCM constants
    private static final int AES_KEY_SIZE = 128; // bits
    private static final int GCM_NONCE_LENGTH = 12; // bytes
```

```
    private static final int GCM_TAG_LENGTH = 128; // bits
    public static void main(String[] args) throws Exception {
        // Original message
        String plainText = "Rijndael Algorithm with GCM Mode in Java";
        System.out.println("Original Text : " + plainText);
        // 1. Generate AES key
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(AES_KEY_SIZE);
        SecretKey secretKey = keyGen.generateKey();
        // 2. Generate random nonce (IV)
        byte[] nonce = new byte[GCM_NONCE_LENGTH];
        java.security.SecureRandom random = new java.security.SecureRandom();
        random.nextBytes(nonce);
        // 3. Encrypt the message
        byte[] cipherText = encrypt(plainText, secretKey, nonce);
        // 4. Decrypt the message
        String decryptedText = decrypt(cipherText, secretKey, nonce);
        // Output results
        System.out.println("Encrypted (Base64): " +
Base64.getEncoder().encodeToString(cipherText));
        System.out.println("Decrypted Text     : " + decryptedText);
    }
    // Encryption method using AES-GCM
    public static byte[] encrypt(String plainText, SecretKey key, byte[] nonce)
throws Exception {
        Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
        GCMParameterSpec gcmSpec = new
GCMParameterSpec(GCM_TAG_LENGTH, nonce);
```

```
        cipher.init(Cipher.ENCRYPT_MODE, key, gcmSpec);
        return cipher.doFinal(plainText.getBytes());
    }
    // Decryption method using AES-GCM
    public static String decrypt(byte[] cipherText, SecretKey key, byte[] nonce)
throws Exception {
        Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
        GCMParameterSpec gcmSpec = new
GCMParameterSpec(GCM_TAG_LENGTH, nonce);
        cipher.init(Cipher.DECRYPT_MODE, key, gcmSpec);
        byte[] decryptedText = cipher.doFinal(cipherText);
        return new String(decryptedText);
    }
}
```

Ex-6

```
import java.util.*;
public class RC4 {
    private int[] S = new int[256];
    private int x, y;
    // Constructor: Key Scheduling Algorithm (KSA)
    public RC4(byte[] key) {
        int keyLength = key.length;
        for (int i = 0; i < 256; i++) {
            S[i] = i;
        }
        int j = 0;
        for (int i = 0; i < 256; i++) {
```

```
        j = (j + S[i] + key[i % keyLength]) & 0xFF;
        swap(S, i, j);
    }
    x = 0;
    y = 0;
}
// Swap function
private void swap(int[] S, int i, int j) {
    int temp = S[i];
    S[i] = S[j];
    S[j] = temp;
}
// RC4 Encryption/Decryption (same process)
public byte[] encrypt(byte[] plaintext) {
    byte[] ciphertext = new byte[plaintext.length];
    for (int i = 0; i < plaintext.length; i++) {
        x = (x + 1) & 0xFF;
        y = (y + S[x]) & 0xFF;
        swap(S, x, y);
        int keyStream = S[(S[x] + S[y]) & 0xFF];
        ciphertext[i] = (byte) (plaintext[i] ^ keyStream);
    }
    return ciphertext;
}
// Decryption is same as encryption
public byte[] decrypt(byte[] ciphertext) {
    return encrypt(ciphertext);
```

```
    }
    // Convert bytes to hexadecimal string
    public static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            String hex = Integer.toHexString(b & 0xFF);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
    // Main function
    public static void main(String[] args) {
        String key = "secretkey";
        String plaintext = "hello world";
        RC4 rc4 = new RC4(key.getBytes());
        // Encrypt
        byte[] encrypted = rc4.encrypt(plaintext.getBytes());
        System.out.println("Plaintext: " + plaintext);
        System.out.println("Ciphertext (hex): " + bytesToHex(encrypted));
        // Decrypt
        byte[] decrypted = rc4.decrypt(encrypted);
        System.out.println("Decrypted: " + new String(decrypted));
    }
}
```

Ex -7

```
<!DOCTYPE html>
<html>
<head>
   <title>Diffie–Hellman Key Exchange</title>
</head>
<body>
   <script>
      // Function to compute (a^b) mod p
      function power(a, b, p) {
         if (b == 1)
            return a % p;
         else
            return (Math.pow(a, b) % p);
      }
      // Driver code
      var p, g, a, b, A, B, keyA, keyB;
      // Input prime number p and primitive root g
      p = parseInt(prompt("Enter a prime number p:"));
      document.write("The value of p: " + p + "<br>");
      g = parseInt(prompt("Enter a primitive root of p (g):"));
      document.write("The value of g: " + g + "<br>");
      // Private key for Alice
      a = parseInt(prompt("Enter the private key for Alice (a):"));
      document.write("The private key (a) for Alice: " + a + "<br>");
      // Public key for Alice A = g^a mod p
```

```
        A = power(g, a, p);
        document.write("The public key (A) for Alice: " + A + "<br>");
        // Private key for Bob
        b = parseInt(prompt("Enter the private key for Bob (b):"));
        document.write("The private key (b) for Bob: " + b + "<br>");
        // Public key for Bob B = g^b mod p
        B = power(g, b, p);
        document.write("The public key (B) for Bob: " + B + "<br>");
        // Shared secret keys
        keyA = power(B, a, p); // Alice computes
        keyB = power(A, b, p); // Bob computes
        document.write("<br>Shared secret key for Alice: " + keyA + "<br>");
        document.write("Shared secret key for Bob: " + keyB + "<br>");
    </script>
</body>
</html>
```

Ex-9

```
import java.security.MessageDigest;
public class SHA1Example {
    public static void main(String[] args) {
        try {
            String message = "Hello, SHA-1";
            // Create a MessageDigest instance for SHA-1
            MessageDigest md = MessageDigest.getInstance("SHA-1");
            // Compute the hash (digest)
            byte[] digest = md.digest(message.getBytes("UTF-8"));
```

```
            // Convert the byte array to a hexadecimal string
            StringBuilder hexString = new StringBuilder();
            for (byte b : digest) {
                hexString.append(String.format("%02x", b));
            }
            System.out.println("Message: " + message);
            System.out.println("SHA-1 Digest: " + hexString.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Ex-10

```
import java.security.MessageDigest;
public class MD5Example {
    public static void main(String[] args) {
        try {
            String message = "Hello, MD5";
            // Create a MessageDigest instance for MD5
            MessageDigest md = MessageDigest.getInstance("MD5");
            // Compute the hash (digest)
            byte[] digest = md.digest(message.getBytes("UTF-8"));
            // Convert byte array to hexadecimal string
            StringBuilder hexString = new StringBuilder();
            for (byte b : digest) {
```

```
            hexString.append(String.format("%02x", b));
        }
        System.out.println("Message: " + message);
        System.out.println("MD5 Digest: " + hexString.toString());
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```