Cryptography and Network Security Lab
III -II (IT)

1.XOR a string with a Zero

AIM: Write a C program that contains a string (char pointer) with a value \Hello World'. The program should XOR each character in this string with 0 and displays the result.

DESCRIPTION:

An XOR gate is a digital logic gate with two or more inputs and one output that performs exclusive disjunction.

ALGORITHM:

1. start
2. take input as "Hello World" which is assigned to variable "str"
3. initialise the variable 'len' for calculating the length of the string
4. print the length of the word followed by the input

PROGRAM:

```
#include<stdlib.h>
main()
{
char str[]="Hello World";
char str1[11];
int i,len;
len=strlen(str);
for(i=0;i<len;i++)
{
str1[i]=str[i]^0;
printf("%c",str1[i]);
}

printf("\n");
}
```

Output:
Hello World
Hello World

2.XOR a String With 127

AIM: Write a C program that contains a string (char pointer) with a value\HelloWorld'.
The program should AND and XOR each character in this string with127 and display the result.

DESCRIPTION:

The AND gate is an electronic circuit that gives a high output(1) only if all the inputs are high.A dot(.) is used to show the AND operation i.e. A.B this dot is sometimes omitted.

ALGORITHM:

1. start
2. take the input 'hello world' which is assigned to variable 'str'
3. perform AND operation between the string and 127.
4. then print the result
5. stop.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
char str[]="hello World";
char str1[11];
char str2[11];
int i,len;
len = strlen(str);
for(i=0;i<len;i++)
{
```

```
str1[i] = str[i]&127;
printf("%c",str1[i]);
}
printf("\n");
for(i=0;i<len;i++)
{
str2[i] = str[i]^127;
printf("%c",str2[i]);
}
printf("\n");
}
```

Output:

```
Hello World
Hello World
Hello World
```

3.Encryption & Decryption using Cryptographic Algorithms

AIM: Write a Java program to perform encryption and decryption using the following algorithms:

a) Ceaser Cipher
b) Substitution Cipher
c) Hill Cipher

a)Ceaser Cipher

DESCRIPTION:

The Caesar cipher technique is one of the earliest and simplest method of encryption technique. It's simple type of substitution cipher i.e. each letter of given text is replaced by a letter, some fixed number of positions down the alphabet. Thus, to cipher a given text, we need an integer value, known as shift which indicates the number of positions each letter of the text has been moved down. The encryption can be replaced by using modular arithmetic by first transforming the letter into numbers according to the scheme A=0,B=1,….
Y=24, Z=25.

e.g. ABCD; shift=4
Cipher: EFGH

ALGORITHM:

1. start
2. read the string
3. traverse the given text one character at a time
4. for each character transforms the given character as per the key
5. it prints the encryption and decryption of the given string
6. stop

PROGRAM:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
class CeaserCipher {
static Scanner sc=new Scanner(System.in);
static BufferedReader br = new   BufferedReader(new   InputStreamReader(System.in));
public static void main(String[] args) throws IOException {

System.out.print("Enter any String: ");
String str = br.readLine();
System.out.print("\nEnter the Key: ");
int key = sc.nextInt();
String encrypted = encrypt(str, key);
System.out.println("\nEncrypted String is: " +encrypted);
String decrypted = decrypt(encrypted, key);
System.out.println("\nDecrypted String is: " +decrypted);
System.out.println("\n");
}
public static String encrypt(String str, int key) {
String encrypted = "";
for(int i = 0; i < str.length(); i++) {
int c = str.charAt(i);
if (Character.isUpperCase(c)) {
c = c + (key % 26);
if (c > 'Z')
c = c - 26;
}
else if (Character.isLowerCase(c)) {
```

```
c = c + (key % 26);
if (c > 'z')
c = c - 26;
}
encrypted += (char) c;
}
return encrypted;
}
public static String decrypt(String str, int key) {
String decrypted = "";
for(int i = 0; i < str.length(); i++) {
int c = str.charAt(i);
if (Character.isUpperCase(c)) {
c = c - (key % 26);
if (c < 'A')
c = c + 26;
}
else if (Character.isLowerCase(c)) {
c = c - (key % 26);
if (c < 'a')
c = c + 26;
}
decrypted += (char) c;
}
return decrypted;
}
}
```

Output:

```
Enter any String: Hello World
Enter the Key: 5
Encrypted String is: MjqqtBtwqi
Decrypted String is: Hello World
```

b)Substitution Cipher

DESCRIPTION:

Substitution Cipher is a method of encrypting by which units of plain text are replaced with cipher text, the 'units' may be single letter or mixture of the letters. In a substitution cipher, the units of the plain text are retained in the same sequence in the cipher text, but the units themselves are altered.

ALGORITHM:

1. start
2. assign a= 'abcdefghijklmnopqrstuvwxyz' b=

   'zyxwvutsrqponmkjihgfedcba'
3. read input string which need to be encrypted
4. compare variables 'a' and 'b' values and substitute the input string position with variable 'b' strings till end of input string
5. print cipher text which is produced from step-4 6. stop

PROGRAM:

```
import java.io.*;
import java.util.*;
public class SubstitutionCipher {
    static Scanner sc = new Scanner(System.in);
static BufferedReader br = new   BufferedReader(new   InputStreamReader(System.in));
public static void main(String[] args) throws IOException {

String a = "abcdefghijklmnopqrstuvwxyz ";
String b = "zyxwvutsrqponmlkjihgfedcba ";
System.out.print("Enter any string: ");
```

```
String str = br.readLine();
String encrypt = "";
String decrypt = "";
char c;
for(int i=0;i<str.length();i++)
{
c = str.charAt(i);
int j = a.indexOf(c);
encrypt = encrypt+b.charAt(j);
}
System.out.println("The encrypted data is: " +encrypt);
for(int i=0;i<str.length();i++)
{
c = encrypt.charAt(i);
int j = a.indexOf(c);
decrypt = decrypt+b.charAt(j);
}
System.out.println("The decrypted data is: " +decrypt);
}
}
```

Output:

```
Enter any string: aceho
The encrypted data is: zxvsl
```

C) Hill Cipher

DESCRIPTION:

Hill cipher is a polygraph substitution cipher based on linear algebra.Each letter is represented by a number modulo 26.To encrypt a message,each block of 'n'letters is multiplied by an inversible n*n matrix against modulus 26.To decrypt the message,each block is multiplied by the inverse of the matrix used for encryption.The matrix used for encryption is the cipher key and it should be chosen randomly from the set of inversible n*n.

ALGORITHM:

a. start

b. to encrypt a string each block of '3'letters are multiplied by inversible 3*3matrix against %26

c. to decrypt a string each block is multiplied by inverse of the matrix used for encryption

d. read a three-letter string

e. It pints the inverse matrix,encryption and decryption of the given three letter string

f. stoP

PROGRAM:

```
import java.io.*;
import java.util.*;
import java.io.*;
public class HillCipher {
static int[][] decrypt = new int[3][1];
static int[][] b = new int[3][3];
static int[][] mes = new int[3][1];
static int[][] res = new int[3][1];
static int a[][] = {{1,2,3},{0,1,4},{5,6,0}};
static BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); static
Scanner sc = new Scanner(System.in);
public static void main(String[] args) throws IOException {
getkeymes();
for(int i=0;i<3;i++)
for(int j=0;j<1;j++)
for(int k=0;k<3;k++) {
res[i][j]=res[i][j]+a[i][k]*mes[k][j]; }
System.out.print("\nEncrypted string is : ");
for(int i=0;i<3;i++) {
System.out.print((char)(res[i][0]%26+97));
res[i][0]=res[i][0];
}
inverse();
for(int i=0;i<3;i++)
for(int j=0;j<1;j++)
for(int k=0;k<3;k++) {
decrypt[i][j] = decrypt[i][j]+b[i][k]*res[k][j]; }
System.out.print("\nDecrypted string is : ");
```

```
for(int i=0;i<3;i++){
System.out.print((char)(decrypt[i][0]%26+97));
}
System.out.print("\n");
}
public static void getkeymes() throws IOException {

System.out.print("\nEnter a 3 letter string: "); String msg = br.readLine();
for(int i=0;i<3;i++)
mes[i][0] = msg.charAt(i)-97;
}
public static void inverse() {
int p, q;
int[][] c = a;
for(int i=0;i<3;i++)
for(int j=0;j<3;j++) {
//a[i][j]=sc.nextint();
if(i==j)
b[i][j]=1;
else b[i][j]=0;
}
for(int k=0;k<3;k++) {
for(int i=0;i<3;i++) {
p = c[i][k];
q = c[k][k];
for(int j=0;j<3;j++) {
if(i!=k) {
c[i][j] = c[i][j]*q-p*c[k][j];
b[i][j] = b[i][j]*q-p*b[k][j];
} } } }
for(int i=0;i<3;i++)
```

```
for(int j=0;j<3;j++) {
b[i][j] = b[i][j]/c[i][i]; }
System.out.println("");
System.out.println("\nInverse Matrix is : ");
for(int i=0;i<3;i++) {
for(int j=0;j<3;j++)
System.out.print(b[i][j] + " ");
System.out.print("\n"); }
}
}
```

Output:

```
Enter a 3 letter string: hai
Encrypted string is :fdx
Inverse Matrix is :
0.083333336 0.41666666 -0.33333334
-0.41666666 -0.083333336 0.6666667
0.5833333 -0.083333336 -0.33333334
Decrypted string is :hai
```

4. JAVA program for DES algorithm logic

AIM: Write a Java program to implement the DES algorithm logic.

DESCRIPTION:

DES is an implementation of Feistel Cipher. It uses16 round Feistel structure. The block size is 64 bits. Though, key length is 64-bits, DES has an effective key length and 56-bits. Since 8 of the 64bits of the key one not used by the encryption algorithm.

ALGORITHM:

1.First we need to get the key generator instance using DES algorithm.

2.Generate secure key that will be used for encryption and decryption

3. Get Cipher instance using DES algorithm. One for encrypt mode and another for decrypt mode. Initialize the cipher object using key and ParameterSpec object.

4. For encryption, create object of CipherOutputStream using encrypt cipher. For decryption, create object of CipherInputStream using decrypt cipher.

5. Read the input stream and write the output stream.

PROGRAM:

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.util.Base64;

class DESExample {

   Cipher ecipher;
   Cipher dcipher;

   DESExample(SecretKey key) throws Exception {
      ecipher = Cipher.getInstance("DES");
      dcipher = Cipher.getInstance("DES");
      ecipher.init(Cipher.ENCRYPT_MODE, key);
      dcipher.init(Cipher.DECRYPT_MODE, key);
   }

   public String encrypt(String str) throws Exception {
      // Encode the string into bytes using utf-8
      byte[] utf8 = str.getBytes("UTF8");

      // Encrypt
      byte[] enc = ecipher.doFinal(utf8);

      // Encode bytes to base64 to get a string

      return Base64.getEncoder().encodeToString(enc);
```

```
    }

    public String decrypt(String str) throws Exception {
        // Decode base64 to get bytes
        byte[] dec = Base64.getDecoder().decode(str);

        byte[] utf8 = dcipher.doFinal(dec);

        // Decode using utf-8
        return new String(utf8, "UTF8");
    }
    public static void main(String[] argv) throws Exception {
        final String secretText = "www.knowledgefactory.net";
        System.out.println("SecretText: " + secretText);
        SecretKey key = KeyGenerator.getInstance("DES").generateKey();
        DESExample encrypter = new DESExample(key);
        String encrypted = encrypter.encrypt(secretText);
        System.out.println("Encrypted Value: " + encrypted);
        String decrypted = encrypter.decrypt(encrypted);
        System.out.println("Decrypted: " + decrypted);
    }
}
```

OUTPUT:

Secret Text: www.knowledgefactory.net

Encrypted Value: 5ayDddgkaJDiVSWqkG0IvZNwKVz8IeQWF1+ulLDb5rI=

Decrypted: www.knowledgefactory.net

5.Program to implement BlowFish algorithm logic

AIM: Write a C/JAVA program to implement the BlowFish algorithm logic.

DESCRIPTION:

Blowfish is a symmetric key block cipher. Blowfish provides a good encryption rate in software and no effective cryptanalysis of it has been found to date.
Blowfish has a 64bit block size and a variable key length from 32bits up to 448bits. It is a 16-round Feistel cipher and uses large key-dependent s-boxes.

ALGORITHM:

1. Blowfish has a 64-bit block size and a variable key length from 30bits up to 48bits.

2. It is a 16-round Feistel cipher and uses large key dependent s-boxes.

3. There are 5 sub key-arrays. One 18-entry p-array and four 256-entry s-boxes.

4. Every round r consists of 4 actions.
   a) XOR the left half of the data with the 'r'th p-array entry.

   b) Use the XORed data as input for Blowfish algorithm. c)F-function's output with the right half (R) of the data. d)Swap L and R.

5. The F-function splits the 32bits into four 8-bits quarters and uses the quarters as input to the s-boxes.

6. The s-boxes accept 8-bit input and produce 32-bit output.The outputs are added modulo 2 power 32 and XORed to produce the final 32-bit output.

PROGRAM:

```
import java.io.UnsupportedEncodingException;
import java.nio.charset.Charset;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

public class BlowfishDemo {

  public String encrypt(String password, String key) throws
              NoSuchAlgorithmException, NoSuchPaddingException,
               InvalidKeyException, IllegalBlockSizeException,
                BadPaddingException, UnsupportedEncodingException {
     byte[] KeyData = key.getBytes();
     SecretKeySpec KS = new SecretKeySpec(KeyData, "Blowfish");
   Cipher cipher = Cipher.getInstance("Blowfish");
     cipher.init(Cipher.ENCRYPT_MODE, KS);
     String encryptedtext = Base64.getEncoder().
        encodeToString(cipher.doFinal(password.getBytes("UTF-8")));
     return encryptedtext;

  }

  public String decrypt(String encryptedtext, String key)
       throws NoSuchAlgorithmException, NoSuchPaddingException,
```

```
        InvalidKeyException, IllegalBlockSizeException,
                BadPaddingException {
    byte[] KeyData = key.getBytes();
    SecretKeySpec KS = new SecretKeySpec(KeyData, "Blowfish");
    byte[] ecryptedtexttobytes = Base64.getDecoder().
                decode(encryptedtext);
    Cipher cipher = Cipher.getInstance("Blowfish");
    cipher.init(Cipher.DECRYPT_MODE, KS);
    byte[] decrypted = cipher.doFinal(ecryptedtexttobytes);
    String decryptedString =
            new String(decrypted, Charset.forName("UTF-8"));
    return decryptedString;
  }
   public static void main(String[] args) throws Exception {
    final String password = "Knf@123";
    final String key = "knowledgefactory";
    System.out.println("Password: " + password+'\n');
    BlowfishDemo obj = new BlowfishDemo();
    String enc_output = obj.encrypt(password, key);
    System.out.println("Encrypted text: " + enc_output+'\n');
    String dec_output = obj.decrypt(enc_output, key);
    System.out.println("Decrypted text: " + dec_output);
  }
}
```

OUTPUT:

```
Password: Knf@123
Encrypted text: 4DTHqnctCuk=
Decrypted text: Knf@123
```

6. Program to implement Rijndael algorithm logic

AIM: Write a C/JAVA program to implement the Rijndael algorithm logic.

DESCRIPTION:

The more popular and widely adopted symmetric encryption algorithm likely to be encountered now-a-days in the Advanced Encryption Standard(AES). It is performed at least 6 times faster than the triple DES.

ALGORITHM:

1.Derive the set of round keys from the cipher key

2.initialize the state array with the block data(plain text)

3.add the initial round key to the starting state array

4.perform nine rounds of state manipulation

5. perform the tenth and final round of state manipulation

6. copy the final state array out as the encrypted data(cipher text)

7. stop

PROGRAM:

```
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;
public class AES {
public static String  asHex  (byte  buf[])
{
  StringBuffer strbuf = new StringBuffer(buf.length * 2);
 int i;
for (i = 0; i < buf.length; i++)
{
 if (((int) buf[i] & 0xff) < 0x10) strbuf.append("0");
strbuf.append(Long.toString((int) buf[i] & 0xff, 16));
}
return strbuf.toString();
}
public static void main(String[] args) throws Exception
{
 String message="cns lab @%";

// Get the KeyGenerator
KeyGenerator kgen = KeyGenerator.getInstance("AES"); kgen.init(128);
// 192 and 256 bits may not be available

//  Generate  the  secret key  specs.
SecretKey skey = kgen.generateKey();
byte[] raw = skey.getEncoded();
SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
```

```
// Instantiate the cipher
Cipher cipher = Cipher.getInstance("AES"); cipher.init(Cipher.ENCRYPT_MODE, skeySpec);

byte [] encrypted = cipher.doFinal((args.length == 0 ? message :args[0]).getBytes());

System.out.println("encrypted string: " + asHex(encrypted)+'\n');

cipher.init(Cipher.DECRYPT_MODE, skeySpec);

byte[] original = cipher.doFinal(encrypted);

String originalString = new String(original);

System.out.println("Original string: " + originalString + " " +'\n' +asHex(original));
}
}
```

OUTPUT:

```
encrypted string: 873b980b3440f1cdc98e8c6f418879ce

Original string: cns lab @%
636e73206c6162204025
```

7.Encrypt a string using BlowFish algorithm

AIM: Using Java Cryptography, encrypt the text "Hello world" using BlowFish. Create your own key using Java keytool.

PROGRAM:

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey; import
javax.swing.JOptionPane;
public class BlowFishCipher {
public static void main(String[] args) throws Exception {

// create a key generator based upon the Blowfish cipher

KeyGeneratorkeygenerator = KeyGenerator.getInstance("Blowfish");

// create a key
SecretKeysecretkey = keygenerator.generateKey();

// create a cipher based upon Blowfish
Cipher cipher = Cipher.getInstance("Blowfish");

// initialise cipher to with secret key
cipher.init(Cipher.ENCRYPT_MODE, secretkey);

// get the text to encrypt
String inputText = JOptionPane.showInputDialog("Input your message: ");

// encrypt message
byte[] encrypted = cipher.doFinal(inputText.getBytes());

// re-initialise the cipher to be in decrypt mode

cipher.init(Cipher.DECRYPT_MODE, secretkey);

// decrypt message
byte[] decrypted = cipher.doFinal(encrypted);
```

```
// and display the results

JOptionPane.showMessageDialog(JOptionPane.getRootFrame(),
"\nEncrypted text: " + new String(encrypted) + "\n" + "\nDecrypted text: "
+ new String(decrypted));

System.exit(0);

}

}
```

OUTPUT:

Input your message: Hello world

Encrypted text: 3ooo&&(*&*4r4

Decrypted text: Hello world

8. RSA Algorithm

AIM: Write a Java program to implement RSA Algoithm.

DESCRIPTION:

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key. Cryptography because one of the keys can be given to anyone.

ALGORITHM:

1.select two prime numbers(p, q)

2.calculate 'n' value n=p*q 3.calculate

$\pi(n)=(p-1)(q-1)$

4.Generate public key by using the formula GCD(e, $\pi(n)$)=1

5.generate private key $D=e^{-1} \text{mod} \pi(n)$

6.Assume plain text is 1, cipher text(c)= m^emodn that is called as encryption. 7.Assume cipher text is c, the plain text is $m=c^d$modn. This is called as decryption.

PROGRAM:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.math.*;
import java.util.Random;
import java.util.Scanner;
public class RSA {
static Scanner sc = new Scanner(System.in);
public static void main(String[] args) {
// TODO code application logic here
System.out.print("Enter a Prime number: ");
BigInteger p = sc.nextBigInteger(); // Here's one prime number..
System.out.print("Enter another prime number: ");
BigInteger q = sc.nextBigInteger(); // ..and another.
BigInteger n = p.multiply(q);
BigInteger n2 =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
BigInteger e = generateE(n2);
BigInteger d = e.modInverse(n2); // Here's the multiplicative inverse
System.out.println("Encryption keys are: " + e + ", " + n);
System.out.println("Decryption keys are: " + d + ", " + n);
}
public static BigInteger generateE(BigInteger fiofn)
{
int y, intGCD;
BigInteger e;
BigInteger gcd;
Random x = new Random();
```

```
do {
y = x.nextInt(fiofn.intValue()-1);
String z = Integer.toString(y);
e = new BigInteger(z);
gcd = fiofn.gcd(e);
intGCD = gcd.intValue();
}
while(y <= 2 || intGCD != 1);
return e;
}
}
```

OUTPUT:

Enter a Prime number: 5
Enter another prime number: 11
Encryption keys are: 33, 55
Decryption keys are: 17, 55

9. Diffie-Hellman

AIM: Implement the Diffie-Hellman Key Exchange mechanism using HTML and JavaScript. Consider the end user as one of the parties (Alice) and the JavaScript application as other party (bob).

DESCRIPTION:

Diffie-Hellman key exchange(DH) is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols as originally conceptualized by Ralph Merkle and named after Whitfield Diffie and Martin Hellman.

ALGORITHM:

1.select prime number i.e. q 2.find the

primitive root of q i.e. x

3. Assume private key for user 'A' is x. Calculating public key for user A is $Y_A=\alpha^{XA}$modq

4. Assuming private key is X_B. Calculating public key is Y_B, $Y_B= \alpha^{XB}$modq

5.Generating secret key K, $K=(Y_A)^{XB}$modq, $K=(Y_B)^{XA}$modq

6.stop

PROGRAM:

```
import java.math.BigInteger;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security .SecureRandom;
import javax.crypto.spec.DHParameterSpec;
import javax.crypto.spec.DHPublicKeySpec;
```

```
public class DiffeHellman {
public final static int pValue = 47;
public final static int gValue = 71;
public final static int XaValue = 9;
public final static int XbValue = 14;
public static void main(String[] args) throws Exception {
// TODO code application logic here

BigInteger p = new BigInteger(Integer.toString(pValue));

BigInteger g = new BigInteger(Integer.toString(gValue));
BigIntegerXa = new BigInteger(Integer.toString(XaValue));
BigIntegerXb = new BigInteger(Integer.toString(XbValue));

createKey();
intbitLength = 512; // 512 bits
SecureRandomrnd = new SecureRandom();
p = BigInteger.probablePrime(bitLength, rnd);
g = BigInteger.probablePrime(bitLength, rnd);

createSpecificKey(p, g);
}

Public static void createKey throws  Exception()
KeypairGenerator kpg=KeyPairGenerator.getInstance(“DiffieHellman”);
kpg.initialize(512);
KeyPair kp = kpg.generateKeyPair();
KeyFactory kfactory = KeyFactory.getInstance("DiffieHellman");
DHPublicKeySpeckspec = (DHPublicKeySpec)
kfactory.getKeySpec(kp.getPublic(),
DHPublicKeySpec.class);
```

```
System.out.println("Public key is: " +kspec);

}

public static void createSpecificKey(BigInteger p, BigInteger g) throws Exception
{
KeyPairGeneratorkpg = KeyPairGenerator.getInstance("DiffieHellman");
DHParameterSpecparam = new DHParameterSpec(p, g);
kpg.initialize(param);
KeyPair kp = kpg.generateKeyPair();
KeyFactory kfactory = KeyFactory.getInstance("DiffieHellman");
DHPublicKeySpeckspec = (DHPublicKeySpec)
kfactory.getKeySpec(kp.getPublic(),DHPublicKeySpec.class);
System.out.println("\nPublic key is : " +kspec);
}
}
```

OUTPUT:

Public key is: javax.crypto.spec.DHPublicKeySpec@5afd29 Public key is: javax.crypto.spec.DHPublicKeySpec@9971ad

10. SHA-1

AIM: Calculate the message digest of a text using the SHA-1 algorithm in JAVA.

DESCRIPTION:

In cryptography, SHA-1(Secure Hashing Algorithm) is a cryptographic hash function which takes an input and produces a 160-bit(20-byte) hash value known as message digest typically rendered as a hexadecimal number, 40 digits long.

ALGORITHM:

1. Pad the bit 00… so that length of plain text is 128<Multiple of 1024 bits

2. Append 128 bits representing of original text such that length=Multiple of 1024 bits

3. initiate the buffers(a, b, c, d, e, f, g and h). Each buffer size=64 bits in hexadecimal

4. process each block of plain text in 80 rounds 5.output buffers is Hash code which has length of 1 bit.

PROGRAM:

```
import java.security.*;
public class SHA1 {
public static void main(String[] a) {
try {
MessageDigest md = MessageDigest.getInstance("SHA1");
System.out.println("Message digest object info: ");
System.out.println(" Algorithm = " +md.getAlgorithm());
System.out.println(" Provider = " +md.getProvider());
System.out.println(" ToString = " +md.toString());
String input = "";
md.update(input.getBytes());
byte[] output = md.digest();
System.out.println();
System.out.println("SHA1(\""+input+"\") = " +bytesToHex(output));
input = "abc";
md.update(input.getBytes());
output = md.digest();
System.out.println();
System.out.println("SHA1(\""+input+"\") = " +bytesToHex(output));
input = "abcdefghijklmnopqrstuvwxyz";
md.update(input.getBytes());
output = md.digest();
System.out.println();
System.out.println("SHA1(\"" +input+"\") = " +bytesToHex(output));
System.out.println(""); }
catch (Exception e) {
   System.out.println("Exception: " +e);
}
```

```
        }
        public static String bytesToHex(byte[] b) {
        char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
        StringBuffer buf = new StringBuffer();
        for (int j=0; j<b.length; j++) {
        buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
        buf.append(hexDigit[b[j] & 0x0f]); }
        return buf.toString(); }
}
```

OUTPUT:

Message digest object info:
Algorithm = SHA1 Provider =
SUN version 1.6
ToString = SHA1 Message Digest from SUN, <initialized> SHA1("") =
DA39A3EE5E6B4B0D3255BFEF95601890AFD80709
SHA1("abc") = A9993E364706816ABA3E25717850C26C9CD0D89D
SHA1("abcdefghijklmnopqrstuvwxyz")=32D10C7B8CF96570CA04CE37F2A19 D8424
0D3A89

Cryptography and Network Security Lab
III -II (IT)