III B. Tech II SEM Regulation: R20

Laboratory Manual

For the course of

CRYPTOGRAPHY AND NETWORK SECURITY LAB

Branch: INFORMATION TECHNOLOGY

VIGNAN'S LARA
INSTITUTE OF TECHNOLOGY & SCIENCE
Approved by AICTE New Delhi & Affiliated to JNTUK Kakinada
Accredited by NAAC 'A+' and NBA | ISO 9001 : 2015
Vadlamudi - 522 213, Guntur District

DEPARTMENT OF INFORMATION TECHNOLOGY

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA
KAKINADA – 533 003, Andhra Pradesh, India

DEPARTMENT OF INFORMATION TECHNOLOGY

III Year – II Semester		L	T	P	C
		0	0	3	1.5
CRYPTOGRAPHY AND NETWORK SECURITY LAB					

Course Objectives:

- To learn basic understanding of cryptography, how it has evolved, and some key encryption techniques used today.
- To understand and implement encryption and decryption using Ceaser Cipher, Substitution Cipher, Hill Cipher.

Course Outcomes: At the end of the course, student will be able to

- Apply the knowledge of symmetric cryptography to implement encryption and decryption using Ceaser Cipher, Substitution Cipher, Hill Cipher
- Demonstrate the different algorithms like DES, BlowFish, and Rijndael, encrypt the text "Hello world" using Blowfish Algorithm.
- Analyze and implement public key algorithms like RSA, Diffie-Hellman Key Exchange mechanism, the message digest of a text using the SHA-1 algorithm

List of Experiments:

1. 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.
2. Write a C program that contains a string (char pointer) with a value \Hello World'. The program should AND or and XOR each character in this string with 127 and display the result
3. Write a Java program to perform encryption and decryption using the following algorithms:
   a) Ceaser Cipher
   b) Substitution Cipher
   c) Hill Cipher
4. Write a Java program to implement the DES algorithm logic
5. Write a C/JAVA program to implement the BlowFish algorithm logic
6. Write a C/JAVA program to implement the Rijndael algorithm logic.
7. Using Java Cryptography, encrypt the text "Hello world" using BlowFish. Create your own key using Java key tool.
8. Write a Java program to implement RSA Algorithm
9. 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).
10. Calculate the message digest of a text using the SHA-1 algorithm in JAVA.

EXP-1

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

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

EXP-2:

2. Write a C program that contains a string (char pointer) with a value \Hello World'. The program should AND or and XOR each character in this string with 127 and display the result

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>  // Needed for strlen()

int main() {
    char str[] = "Hello World";
    char str1[11];
    char str2[11] = "Hello World";  // Initialize str2 with "Hello World"
    char str3[11];  // Declare str3

    int i, len;
    len = strlen(str);  // Get the length of the string
// First loop: Modify str1 by applying bitwise AND with 127
    for (i = 0; i < len; i++) {
        str1[i] = str[i] & 127;
        printf("%c", str1[i]);
```

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

    // Second loop: Modify str3 by applying bitwise XOR with 127
    for (i = 0; i < len; i++) {
        str3[i] = str2[i] ^ 127;
        printf("%c", str3[i]);
    }
    printf("\n");

    return 0;  // Return 0 to indicate successful completion
 }
```

O/P: Original string: Hello World

Result after XOR operation with 127:
55 26 19 19 16 95 40 16 13 19 27

Result after XOR operation with 127 again (to restore):
Hello World

Experiment -3
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

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
public 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: Mjqqt Btwqi
Decrypted String is: Hello World

4. Java program for DES algorithm logic

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

```
import java.util.*;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.security.spec.KeySpec;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESedeKeySpec;

public class DES {
    private static final String UNICODE_FORMAT = "UTF8";
    public static final String DESEDE_ENCRYPTION_SCHEME = "DESede";
    private KeySpec myKeySpec;
    private SecretKeyFactory mySecretKeyFactory;
    private Cipher cipher;
    byte[] keyAsBytes;
    private String myEncryptionKey;
    private String myEncryptionScheme;
    SecretKey key;
    static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

```
public DES() throws Exception {
        myEncryptionKey = "ThisIsSecretEncryptionKey";
        myEncryptionScheme = DESEDE_ENCRYPTION_SCHEME;
        keyAsBytes = myEncryptionKey.getBytes(UNICODE_FORMAT);
        myKeySpec = new DESedeKeySpec(keyAsBytes);
        mySecretKeyFactory = SecretKeyFactory.getInstance(myEncryptionScheme);
        cipher = Cipher.getInstance(myEncryptionScheme);
        key = mySecretKeyFactory.generateSecret(myKeySpec);
    }

    public String encrypt(String unencryptedString) {
```

```
        public static String encode(byte[] data) {
            StringBuilder encodedString = new StringBuilder();
            int paddingCount = 0;
            int length = data.length;

            for (int i = 0; i < length; i += 3) {
                int byte1 = data[i] & 0xFF;
                int byte2 = (i + 1 < length) ? data[i + 1] & 0xFF : 0;
                int byte3 = (i + 2 < length) ? data[i + 2] & 0xFF : 0;
                int combined = (byte1 << 16) | (byte2 << 8) | byte3;
                encodedString.append(BASE64_CHARS.charAt((combined >> 18) & 0x3F));
                encodedString.append(BASE64_CHARS.charAt((combined >> 12) & 0x3F));
                encodedString.append(i + 1 < length ? BASE64_CHARS.charAt((combined >> 6) & 0x3F)
: '=');
                encodedString.append(i + 2 < length ? BASE64_CHARS.charAt(combined & 0x3F) : '=');
            }
            return encodedString.toString();
        }
    }
    static class Base64Decoder {
        private static final String BASE64_CHARS =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

        public static byte[] decode(String str) {
            int paddingCount = 0;
            if (str.endsWith("==")) paddingCount = 2;
            else if (str.endsWith("=")) paddingCount = 1;

            int length = str.length();
            int outputLength = (length * 3) / 4 - paddingCount;
            byte[] decodedBytes = new byte[outputLength];

            int j = 0;
            for (int i = 0; i < length; i += 4) {
                int byte1 = BASE64_CHARS.indexOf(str.charAt(i));
                int byte2 = BASE64_CHARS.indexOf(str.charAt(i + 1));
                int byte3 = (i + 2 < length) ? BASE64_CHARS.indexOf(str.charAt(i + 2)) : -1;
                int byte4 = (i + 3 < length) ? BASE64_CHARS.indexOf(str.charAt(i + 3)) : -1;

                int combined = (byte1 << 18) | (byte2 << 12) | ((byte3 == -1 ? 0 : byte3) << 6) | (byte4 == -
1 ? 0 : byte4);
```

```
    decodedBytes[j++] = (byte) ((combined >> 16) & 0xFF);
                if (byte3 != -1) decodedBytes[j++] = (byte) ((combined >> 8) & 0xFF);
                if (byte4 != -1) decodedBytes[j++] = (byte) (combined & 0xFF);
            }

            return decodedBytes;
        }
    }
}
```

OUTPUT:-

```
PS F:\shd\cns\output> javac  FILE_NAME.java
PS F:\shd\cns\output> java FILE_NAME

Enter the string:  Encripte Me
String To Encrypt:  Encripte Me
Encrypted Value:  0rhOoIVm7BwomLUohlntpw==
Decrypted Value:  Encripte Me
```

5. Program to implement BlowFish algorithm logic

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

PROGRAM:

```
import java.io.*;
import java.security.Key;
import javax.crypto.Cipher;
import javax.crypto.CipherOutputStream;
import javax.crypto.KeyGenerator;

public class BlowFish {
    public static void main(String[] args) throws Exception {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("Blowfish");
        keyGenerator.init(128); // Set the key size (128 bits)
        Key secretKey = keyGenerator.generateKey();
      Cipher cipherOut = Cipher.getInstance("Blowfish/CFB/NoPadding");
        cipherOut.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] iv = cipherOut.getIV();
        if (iv != null) {
            System.out.println("Initialization Vector of the Cipher: " + encodeBase64(iv));
        }
        FileInputStream fin = new FileInputStream("inputFile.txt");
        FileOutputStream fout = new FileOutputStream("outputFile.txt");
```

```
 // Set up CipherOutputStream to write encrypted data to the output file
        CipherOutputStream cout = new CipherOutputStream(fout, cipherOut);
        int input;
        // Read from input file and write encrypted data to output file
        while ((input = fin.read()) != -1) {
            cout.write(input);
        }

        // Close the streams
        fin.close();
        cout.close();
        fout.close();

        System.out.println("Encryption complete. Encrypted file written to outputFile.txt.");
    }

    // Custom Base64 encoder to encode byte data to a Base64 string
    private static String encodeBase64(byte[] data) {
        StringBuilder encodedString = new StringBuilder();
                                                        String          base64Chars           =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
        int paddingCount = 0;
        int length = data.length;

        // Encode the data byte by byte in Base64 format
        for (int i = 0; i < length; i += 3) {
            int byte1 = data[i] & 0xFF;
            int byte2 = (i + 1 < length) ? data[i + 1] & 0xFF : 0;
            int byte3 = (i + 2 < length) ? data[i + 2] & 0xFF : 0;

            int combined = (byte1 << 16) | (byte2 << 8) | byte3;

            encodedString.append(base64Chars.charAt((combined >> 18) & 0x3F));
            encodedString.append(base64Chars.charAt((combined >> 12) & 0x3F));
            encodedString.append(i + 1 < length ? base64Chars.charAt((combined >> 6) & 0x3F) : '=');
            encodedString.append(i + 2 < length ? base64Chars.charAt(combined & 0x3F) : '=');
        }

        return encodedString.toString();
    }
}
```

OUTPUT:-

Initialization Vector of the Cipher: GDFhV58xdfI=

Encryption complete. Encrypted file written to outputFile.txt.

6. Program to implement Rijndael algorithm logic

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

PROGRAM:

```
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;

public class AES {
   public static String asHex(byte[] buf) {
      StringBuilder strbuf = new StringBuilder(buf.length * 2);
      for (int 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 = "AES still rocks!!";

      // 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 for encryption
      Cipher cipher = Cipher.getInstance("AES");
      cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
      // Encrypt the message (use command line argument if available)
      byte[] encrypted = cipher.doFinal((args.length == 0 ? message : args[0]).getBytes());
      System.out.println("Encrypted string: " + asHex(encrypted));
      // Decrypt the message
      cipher.init(Cipher.DECRYPT_MODE, skeySpec);
      byte[] original = cipher.doFinal(encrypted);
      String originalString = new String(original);
      System.out.println("Original string: " + originalString + " " + asHex(original));
   }
}
```

OUTPUT:-
Encrypted string: ad9ec88c4399cb7e85fcf6a7b2f069263c08ac48b3d732b1faaa323fee1b459e
Original string: AES still rocks!! 414553207374696c6c20726f636b732121

Experiment – 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.

```
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
KeyGenerator keygenerator = KeyGenerator.getInstance("Blowfish");
// create a key
SecretKey secretkey = keygenerator.generateKey();
// create a cipher based upon Blowfish Cipher
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 :

Experiment - 8

AIM: Write a Java program to implement RSA Algoithm.

```
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 primenumber..
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).

PROGRAM:

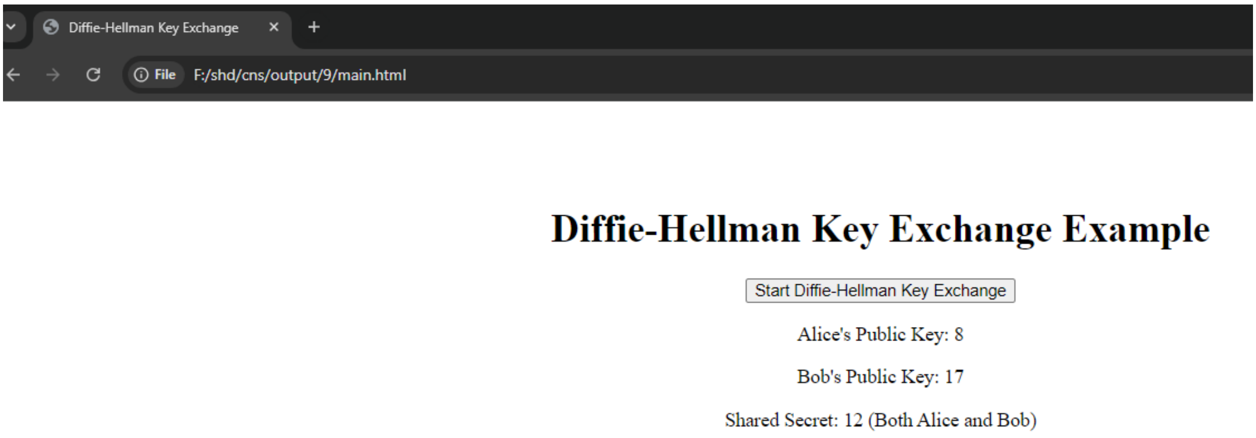
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Diffie-Hellman Key Exchange</title>
    <script>
        // Function to generate a random large prime number (simplified example)
        function generatePrimeNumber() {
```

```
// In real use, you would need a cryptographically secure prime number
    return 23;  // A simple small prime number for illustration purposes
  }

  // Function to generate a primitive root of the prime (simplified)
  function generatePrimitiveRoot(prime) {
    // In real use, we'd need to ensure the root is valid for the prime
    return 5;  // A primitive root of 23 for demonstration
  }

  // Function to perform modular exponentiation (for calculating public keys)
  function modExp(base, exponent, modulus) {
    let result = 1;
    base = base % modulus;
    while (exponent > 0) {
      if (exponent % 2 === 1) {
        result = (result * base) % modulus;
      }
```

```
exponent = Math.floor(exponent / 2);
      base = (base * base) % modulus;
    }
    return result;
  }

  // Alice (the user) generates their private key
  function generatePrivateKey() {
    return Math.floor(Math.random() * 10) + 1; // Random private key between 1 and 10
  }

  // Alice generates her public key
  function generatePublicKey(privateKey, prime, primitiveRoot) {
    return modExp(primitiveRoot, privateKey, prime);  // public key = (g^a) mod p
  }

  // Bob (the app) generates their private key and public key
  function generateBobPublicKey(prime, primitiveRoot) {
    let bobPrivateKey = Math.floor(Math.random() * 10) + 1;  // Random private key for Bob
    let bobPublicKey = modExp(primitiveRoot, bobPrivateKey, prime);  // Bob's public key
    return { bobPrivateKey, bobPublicKey };
```

```
}

        // Alice and Bob exchange public keys and compute shared secret
        function computeSharedSecret(publicKey, privateKey, prime) {
            return modExp(publicKey, privateKey, prime);  // Shared secret = (B^a) mod p
        }

        function startDiffieHellman() {
            // Prime number and primitive root (In a real scenario, these are carefully selected)
            const prime = generatePrimeNumber();
            const primitiveRoot = generatePrimitiveRoot(prime);
            // Alice generates a private key and public key
            let alicePrivateKey = generatePrivateKey();
            let alicePublicKey = generatePublicKey(alicePrivateKey, prime, primitiveRoot);
            let { bobPrivateKey, bobPublicKey } = generateBobPublicKey(prime, primitiveRoot);
            // Display Alice and Bob's public keys
                    document.getElementById('alicePublicKey').textContent = `Alice's Public Key: 
${alicePublicKey}`;
                     document.getElementById('bobPublicKey').textContent = `Bob's Public Key: 
${bobPublicKey}`;
            // Alice computes the shared secret using Bob's public key
            let aliceSharedSecret = computeSharedSecret(bobPublicKey, alicePrivateKey, prime);
            // Bob computes the shared secret using Alice's public key
            let bobSharedSecret = computeSharedSecret(alicePublicKey, bobPrivateKey, prime);
                        document.getElementById('sharedSecret').textContent = `Shared Secret: 
${aliceSharedSecret} (Both Alice and Bob)`;
        }
    </script>
</head>xcc
<body>
    <br> <br> <br>
    <center>
    <h1>Diffie-Hellman Key Exchange Example</h1>
    <button onclick="startDiffieHellman()">Start Diffie-Hellman Key Exchange</button>
    <div>
        <p id="alicePublicKey"></p>
        <p id="bobPublicKey"></p>
        <p id="sharedSecret"></p>
    </center>
    </div>
</body>
<html>d
```

OUTPUT:-



Experiment -10

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

```
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") = 32D10C7B8CF96570CA04CE37F2A19D84240D3A89
```