**Experiment No.10**

**public class Main {**

**public static String encryptThisString(String input)**

**{**

**Try{**

**byte[] messageDigest = md.digest(input.getBytes**

**BigInteger no = new BigInteger(1, messageDigest);**

**String hashtext = no.toString(1**

**while (hashtext.length() < 32) {**

**hashtext = "0" + hashtext;**

**}**

**return hashtext;**

**}**

**{**

**throw new RuntimeException(e);**

**}**

**{**

**System.out.println("HashCode Generated by SHA-1 for: "**

**String s1 = "GeeksForGeeks";**

**System.out.println("\n" + s1 + " : " + encryptThisString(s1));**

**String s2 = "hello world";**

**System.out.println("\n" + s2 + " : " + encryptThisString(s2));**

**HashCode Generated by SHA-1 for:**

**GeeksForGeeks : addf120b430021c36c232c99ef8d926aea2acd6b**

**hello world : 2aae6c35c94fcfb415dbe95f408b9ce91ee846e**

**Experiment..11**

**#include <iostream>**

**#include <string>**

**using namespace CryptoPP;**

**template <unsigned int HASH\_SIZE = 32>**

**class IdentityHash : public HashTransformation**

**{**

**public:**

**CRYPTOPP\_CONSTANT(DIGESTSIZE = HASH\_SIZE)**

**static const char \* StaticAlgorithmName()**

**{**

**return "IdentityHash";**

**}**

**IdentityHash() : m\_digest(HASH\_SIZE), m\_idx(0) {}**

**virtual unsigned int DigestSize() const**

**{**

**return DIGESTSIZE;**

**}**

**virtual void Update(const byte \*input, size\_t length)**

**{**

**size\_t s = STDMIN(STDMIN<size\_t>(DIGESTSIZE, length),**

**DIGESTSIZE - m\_idx);**

**if (s)**

**::memcpy(&m\_digest[m\_idx], input, s);**

**m\_idx += s;**

**}**

**virtual void TruncatedFinal(byte \*digest, size\_t digestSize)**

**{**

**if (m\_idx != DIGESTSIZE)**

**throw Exception(Exception::OTHER\_ERROR, "Input size must be " + IntToString(DIGESTSIZE));**

**ThrowIfInvalidTruncatedSize(digestSize);**

**if (digest)**

**::memcpy(digest, m\_digest, digestSize);**

**m\_idx = 0;**

**}**

**private:**

**SecByteBlock m\_digest;**

**size\_t m\_idx;**

**};**

**int main(int argc, char\* argv[])**

**{**

**AutoSeededRandomPool prng;**

**DSA::PrivateKey privateKey;**

**privateKey.Initialize(prng, 2048);**

**std::string message;**

**message.resize(IdentityHash<224/8>::DIGESTSIZE);**

**::memset(&message[0], 0xAA, message.size());**

**DSA::Signer signer(privateKey);**

**std::string signature;**

**StringSource ss(message, true,**

**new SignerFilter(prng, signer,**

**new HexEncoder(new StringSink(signature))**

**)**

**);**

**cout << "Signature: " << signature << std::endl;**

**return 0;**

**}**

**Signature: 51F37CA3F02A89CE25FBF815CA4D0EB8624BA58FBB87B0DDDEB194D77ECDDBBC9B246**

**CB69FFF18576FD9DD6D2BCF7AF9E7BA49FB382767BC**

**Experinent..9**

**#include<stdio.h>**

**#include<conio.h>**

**int main()**

**{**

**int i, cnt=0, p8[8]={6,7,8,9,1,2,3,4};**

**int p10[10]={6,7,8,9,10,1,2,3,4,5};**

**char input[11], k1[10], k2[10], temp[11];**

**char LS1[5], LS2[5];**

**printf("Enter 10 bits input:");**

**scanf("%s",input);**

**input[10]='\0';**

**for(i=0; i<10; i++)**

**{**

**cnt = p10[i];**

**temp[i] = input[cnt-1];**

**}**

**temp[i]='\0';**

**printf("\nYour p10 key is :");**

**for(i=0; i<10; i++)**

**{ printf("%d,",p10[i]); }**

**printf("\nBits after p10 :");**

**puts(temp);**

**for(i=0; i<5; i++)**

**{**

**if(i==4)**

**temp[i]=temp[0];**

**else**

**temp[i]=temp[i+1];**

**}**

**for(i=5; i<10; i++)**

**{**

**if(i==9)**

**temp[i]=temp[5];**

**else**

**temp[i]=temp[i+1];**

**}**

**printf("Output after LS-1 :");**

**puts(temp);**

**printf("\nYour p8 key is :");**

**for(i=0; i<8; i++)**

**{ printf("%d,",p8[i]); }**

**for(i=0; i<8; i++)**

**{**

**cnt = p8[i];**

**k1[i] = temp[cnt-1];**

**}**

**printf("\nYour key k1 is :");**

**puts(k1);**

**}**

**output**

**Enter 10 bits input:1100011100**

**Your p10 key is :6,7,8,9,10,1,2,3,4,5,**

**Bits after p10 :1110011000**

**Output after LS-1 :1100110001**

**Your p8 key is :6,7,8,9,1,2,3,4,**

**Your key k1 is :10001100**