SECURE COMMUNICATION OVER SSL

P-1216

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF

**BACHELOR OF ENGINEERING IN
COMPUTER SCIENCE AND ENGINEERING**

OF THE BHARATHIAR UNIVERSITY, COIMBATORE

SUBMITTED BY

AKSHAY CHANDU	**0027K0154**
BALUMAHENDRAN.V	**0027K0166**
MANJU.H	**0027K0181**

UNDER THE GUIDANCE OF

Mrs D.Chandrakala M.E.

MARCH 2004

DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

(AFFILIATED TO BHARATHIAR UNIVERSITY)

COIMBATORE – 641 006

KUMARAGURU COLLEGE OF TECHNOLOGY
(AFFILIATED TO BHARATHIAR UNIVERSITY)
COIMBATORE – 641 006, TAMIL NADU, INDIA
APPROVED BY AICTE, NEW DELHI - ACCREDITED BY NBA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

THIS IS TO CERTIFY THAT THE PROJECT ENTITLED

"SECURE COMMUNICATION OVER SSL"

HAS BEEN SUBMITTED BY

AKSHAY CHANDU, BALUMAHENDRAN.V & MANJU.H

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF DEGREE OF

BACHELOR OF ENGINEERING

DURING THE ACADEMIC YEAR 2003-2004

[illegible]
Head of the Department

[illegible] 16/3/'04
Project Guide

Submitted for the university examination held on 23-3-2004

[illegible] 23/3/'04
Internal Examiner

[illegible]
External Examiner

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We are greatly indebted to our revered Principal **Dr.K.K.Padmanabhan**, who has been the motivating force behind all our deeds.

We earnestly express our sincere thanks to our beloved Head of the Department **Prof.S.Thangasamy Ph.D.**, for his immense encouragement and help and for being our source of inspiration all through our course of study.

We are much obliged to express our sincere thanks and gratitude to our beloved guide , **Mrs.D.Chandrakala, M.E.**, for her valuable suggestions, construction criticisms and encouragement which has enabled us to complete our project successfully.

We gratefully thank **Mrs.M.S.Hema, B.E.**, for extending her most appreciative and timely help to us.

We also thank all staff members of the Department of Computer Science and Engineering for all their encouragement and moral support.

We also extend our heartiest thanks to all our friends for their continuous help and encouragement throughout the course of study.

SYNOPSIS

SYNOPSIS

Security is important on the network since data sensitivity on the network has to be taken care at all levels, particularly with credit transaction involving a lot of money and with huge volume of confidential data.

This project work entitled "**Secure Communication Over SSL**" is used for providing secure communication over Local Area Network (LAN) for file transaction. SSL is application independent .The file to be transmitted is encrypted using a standard encryption algorithm namely RSA .The file tends to increase in size after encryption thereby slowing down the process of transaction of files across network. To avoid this, a compression phase is added which uses 3 algorithms, viz. Run Length Encoding, Burrows-Wheeler Transform and Move-To-Front to significantly compress the file size. The file is then sent over the Secure Socket Layer residing on top of the TCP/IP layer.

At the receiving end, decompression and decryption are done in the specified order to get back the original file. Thus both security and fast transmission over the network are achieved.

CONTENTS

CONTENTS

INTRODUCTION

1. INTRODUCTION

Computer Networks are typically a shared resource used by many applications for many different purposes .Though ,there are lots of benefits which a network may provide, security of data is a key issue to be addressed. The data transmitted in the network between various applications should be highly confidential.Hence eavesdropping may be dangerous. Therefore, users sometime want to encrypt the messages they send, with the goal of keeping any one who is eavesdropping on the channel from being unable to read the contents of the message.

In this project, Secure Socket Layer (SSL) , an encryption method for Transmission Control Protocol/Internet Protocol(TCP/IP) sockets , is used for secure file transaction over a Local Area Network .The file is both encrypted and compressed before transmission, thereby providing security along with agility.

1.1 EXISTING SYSTEM AND ITS LIMITATIONS:

Existing system is client / server model .On one side there is the customer client and on the other side there is a server. The client sends confidential data over the TCP/IP connection to the server. File transfer takes place using a program like FTP.

DISADVANTAGES:

- ➢ No encryption.
- ➢ Interception of a file transfer.
- ➢ No control over sending a partial file.

1.2 PROPOSED SYSTEM AND ITS ADVANTAGES :

SSL is provided in between the Transport layer and Application layer. On the sending side, SSL receives data, encrypts it, and is compressed and directs the data to a TCP socket. At the receiving side, data is received from socket, decompressed and then decrypted.

SSL is protected from dictionary attacks by having very large keys spaces for all its ciphers. Even if the attacks are run against the export ciphers, they become prohibitively large. Because the dictionary would have to be produced for each session, the hackers have to supply the session ID each time for breaking the information, which is practically impossible.

Nor is SSL vulnerable to man-in-the middle type of attacks providing that the server uses private key to decrypt the message.

SOFTWARE REQUIREMENTS SPECIFICATION

2. SOFTWARE REQUIREMENTS SPECIFICATION

2.1. INTRODUCTION:

PURPOSE:

To provide secure and fast communication using SSL.

OVERVIEW:

This project aims at providing quick and safe transfer of files using the secure socket layer, which is provided between the transport layer and the application layer.

SCOPE:

This project can serve as a means of providing secure file transfer in LAN.

DEFINITIONS ,ACRONYMS:

SSL:	Secure Socket Layer
RSA:	Rivest-Shamir-Adelman
LAN:	Local Area Network
TCP .	Transmission Control Protocol
IP:	Internet Protocol

2.2OVERALL DESCRIPTION:

SSL is provided in between the Transport layer and Application layer. On the sending side, SSL receives data, encrypts it , and is compressed and directs the data to a TCP socket. At the receiving side, data is received from socket, decompressed and then decrypted.

2.3.FUNTIONAL SPECIFICATION:

1) ENCRYPTION AND DECRYPTION:

Introduction:

This module deals with providing security for transfer of data.

Input:

At Transmission Side:

The user gives in the name of the file that has to be encrypted before it is transferred .

At Reception Side:

For decryption, the input is the decompressed file .

Process:

Encryption and decryption of data using RSA.

Output:

At Transmission Side:

The encrypted file is obtained. The content of it is called the cipher text.

At Reception side:

The original file is recovered.

2) COMPRESSION AND DECOMPRESSION:

Introduction:

This module deals with reduction in size of file for quick transfer

Input:

At transmission End:

The encrypted file for compression.

At Reception End:

The transferred file for decompression

Process:

Compression is performed using a sequence of algorithms viz.. Run Length Encoding, Burrows-Wheeler Transform, Move To Front and Run Length Encoding again.The reverse is used for decompression.

Output:

At Transmission Side: Compressed File.

At Reception Side: Original File.

3) CREATION OF SSL AND TRANSMISSION:

Introduction:

This involves creation of sockets and transmission of data.

Input:

Encrypted and compressed file.

Process:

Transmission of file across to the other system.

Output:

Message describing the status of file transfer at the transmission side and reception of file at the receiving end.

2.4.ASSUMPTIONS AND DEPENDENCIES:

CONSTRAINTS AND DEPENDENCIES:

- Communication between not more than two systems at a time.
- Transfer of text files only.

ASSUMPTIONS:

Both systems are connected through the LAN.

2.5.EXTERNAL INTERFACE REQUIREMENTS:

Dialog boxes for encryption, compression, connection, decompression and decryption.

2.6.DESIGN CONSTRAINTS:

HARDWARE LIMITATIONS:

- Communication is limited to LAN.
- Communication speed depends on file size.

SOFTWARE LIMITATIONS:

This works only on any one of the windows platforms.

2.7.ACCEPTANCE TESTING REQUIREMENT:

Two systems of a LAN and text files to be transferred.

2.8.OTHER REQUIREMENTS:

Hardware Specification:

Processor	– Pentium III and above.
RAM	– 64 MB

Software Specification:

Operating System	– Windows 98
Language	– Microsoft VC++

DETAILS OF PROPOSED METHODOLOGY

3. DETAILS OF PROPOSED METHODOLOGY

3.1CRYPTOGRAPHY:

Cryptography deals with algorithms for encryption and decryption for the purpose of ensuring secrecy and authenticity of messages.

Plain Text:

This is the readable message or data that is fed into the Encryption Algorithm as input or obtained as output from a Decryption Algorithm.

Key:

This is the value independent of the plain text that is used by the encryption algorithm for producing the cipher text or by decryption algorithm to retrieve the plain text.

Encryption Algorithm:

The Encryption Algorithm performs various transformations on the plain text and converts it into an unintelligible form.

Cipher Text:

This is the scrambled message produced as output .It depends on the plain text and the key .For a given message, two different keys will produce two different cipher texts.

Decryption Algorithm:

This algorithm accepts the cipher text and with the help of the decryption key the original plain text is retrieved.

3.1.1 ENCRYPTION METHODS:

There are two types of encryption namely, Symmetric Encryption (secret key encryption) And Asymmetric Encryption (public key encryption).

Symmetric Encryption:

It is also called secret key encryption because the secret key must be shared between the sender and receiver. The symmetric encryption can be depicted as a mathematical formula:

Encrypted Data=Function (data, key)

With an inverse function existing of the form

Plain text data=Inverse function (Encrypted data, key)

Asymmetric Encryption:

In this type of encryption, two keys are involved. The first key (the public key) is published and is used by the entities to send data securely The second key (the private key) is known only to the owner .This means that the private key can be used to encrypt data that can be decrypted only by using the public key and vice versa.

3.1.2 RSA Algorithm:

This Algorithm is used for the encryption and decryption process. It is named after its inventors: Ron Rivest, Adi Shamir and Leonard Ad leman founding members of RSA Data security.

Description Of The Algorithm:

Plain text is encrypted in blocks, with each block having a binary value less than some number n. Encryption and decryption are of the following form, for some plain text block M and cipher text block C:

$C = M \wedge e \bmod n$

$M = C \wedge d \bmod n$

$N = (M) \wedge ed \bmod n$

Both sender and receiver must know the value of n. The sender knows the value e, and the receiver knows the value of d. Thus this is the public key encryption algorithm with a public key of KU= {e, n} and a private key of KR= {d, n}.

Key Generation:

To Create Public Key:

1. Select 2 prime numbers p, q

   p=7, q=17

2. Calculate n=pq

   7*17=119

3. Calculate f (n) = (p-1) (q-1)

   6*16=96

4. Select 'e' such that e is relatively prime to f (n) and less than f (n)

   e=5

5. Public key is then 'e' concatenated with 'n'

Public Key is (5,119)

To Create Private Key:

6. Determine'd' such that MOD (de, 96)=1 and d<96.

   Correct value of d is 77.

7. Private key is then 'd' concatenated with'n'

Private Key is (77,119)

ENCRYPTION:

Plain text: M<n

Plain text: 19 (P is treated as a numerical value)

Cipher text: C= M ^ e (mod n)

19 POW 5=2476099/119=20807 REM 66.

Cipher text = 66

<u>DECRYPTION:</u>

Plain text=C

Cipher Text; M=C ^d (mod n)

66 POW 77=value/119=value rem 19

19=Plain text.

3.2 COMPRESSION & DECOMPRESSION

•RUN LENGTH ENCODING

•BURROWS WHEELER TRANSFORM

•MOVE TO FRONT ALGORITHM

3.2.1 RUN LENGTH ENCODING

This program performs the Run Length Encoding function on an input file, and sends the result to an output file or stream. In the output stream, any two consecutive characters with the same value flag a run. A byte following those two characters gives the count of *additional* repeat characters, which can be anything from 0 to 255.

RLE ALGORITHM

•Get two bytes

•Loop

•Are they equal?

•Yes

–Output both of them

–Count how many bytes repeated we have

–Output that value

–Update pointer to the input file

–Get next two bytes.

–Repeat.

•No

–Output the first byte

– -put the second, as first

– -get a byte for the second one

– -update pointer to input file

– -repeat.

–A little example, we have the file: "aaaaaabcddccc" The encoder should output that: a,a,4,b,c,d,d,0,c,c,1 And the decoder will be able to decompress

UNRLE ALGORITHM

•Get one byte, put it to the output file, and now it's the 'last' byte.

•Loop

•Get one byte

•Is the current byte equal to last?

•Yes

–Now get another byte, this is 'counter'

–Put current byte in the output file

–Copy 'counter' times the 'last' byte to the output file

–Put last copied byte in 'last' (or leave it alone)

–Repeat.

•No

–Put current byte to the output file

–Now 'last' is the current byte

–Repeat.

ADVANTAGES

1. A front end to BWT avoids pathologically slow sorts that occur when the input stream has long sequences of identical characters
2. The RLE has also been used as a heuristic postprocessor from the MTF program

3.2.2 BURROWS WHEELER TRANSFORM

The BWT is an algorithm that takes a block of data and rearranges it using a sorting algorithm. Consider the block

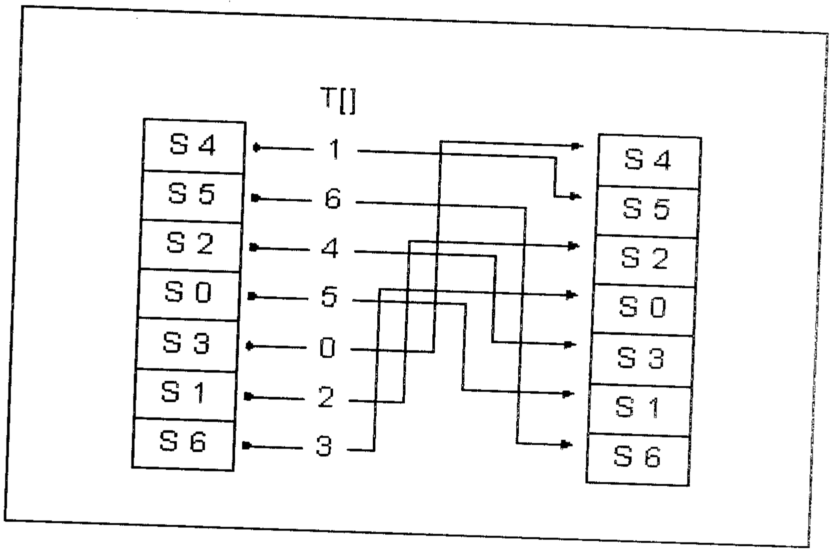
D	R	D	O	B	B	S

COMBINATIONS:

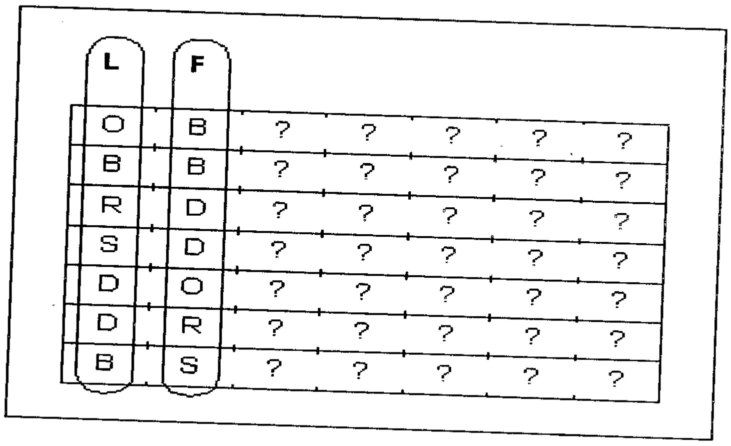
String 0	D	R	D	O	B	B	S
S 1	R	D	O	B	B	S	D
S 2	D	O	B	B	S	D	R
S 3	O	B	B	S	D	R	D
S 4	B	B	S	D	R	D	O
S 5	B	S	D	R	D	O	B
S 6	S	D	R	D	O	B	B

TRANSFORM VECTOR:

The transformation vector routes S[i] to S [i + 1]



REGENERATION OF ORIGINAL DATA SET



The output consists of a copy of last column of sorted strings and an integer indicating which row contains the original first character of buffer

INVERSE BWT:

The transformation vector T is calculated from the copy of L.The vector routes s [I] to s [I+1].By using the primary index from transform vector, the original string can be retrieved.BWT normally compresses large blocks of data as the output depends on future bytes.

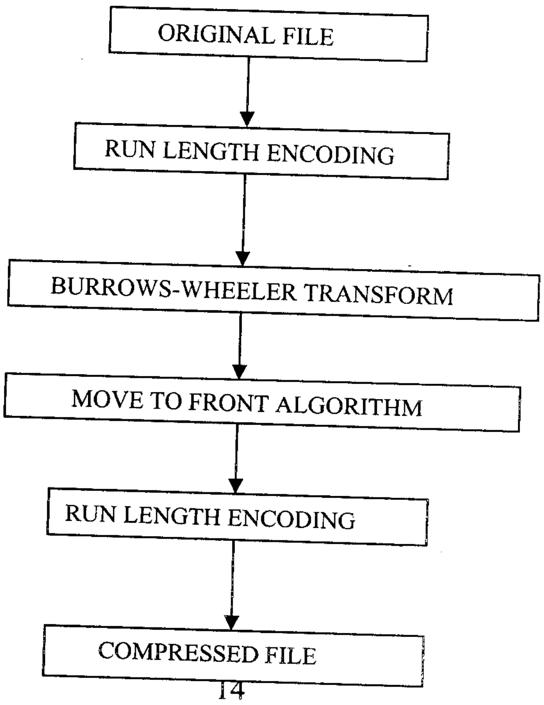
3.2.3 MOVE TO FRONT ALGORITHM

Move To Front is a transformation algorithm which does not compress data but can help to reduce redundancy some times, like after a BWT transformation, where a symbol which has recently seen, appears again

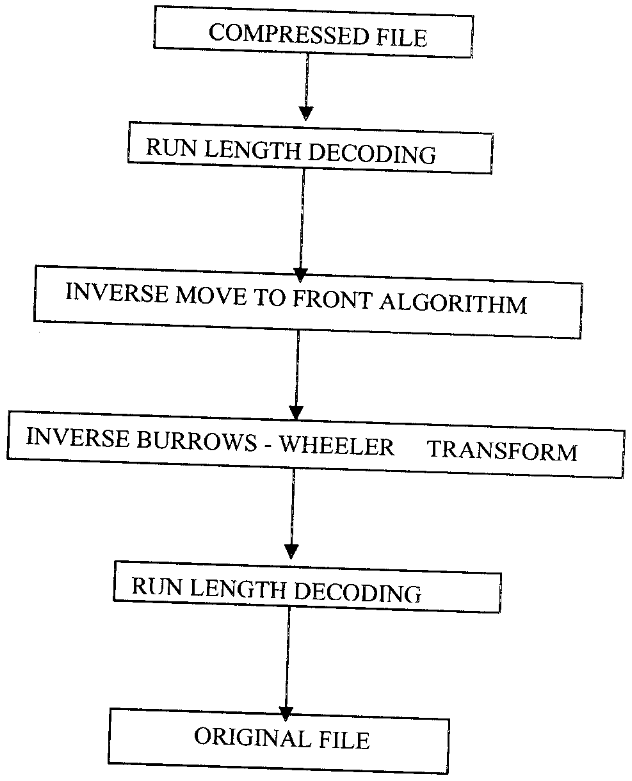
PROCEDURE

MTF instead of outputting the symbol (byte), outputs a code which refers to the position of the symbol in a table with all the symbols, thus the length of the code is the same as the length of the symbol. These simple schemes assign codes with lower values for more redundant symbols.

SEQUENCE FOR COMPRESSION



SEQUENCE FOR DECOMPRESSION


COMPRESSED FILE
RUN LENGTH DECODING
INVERSE MOVE TO FRONT ALGORITHM
INVERSE BURROWS - WHEELER TRANSFORM
RUN LENGTH DECODING
ORIGINAL FILE

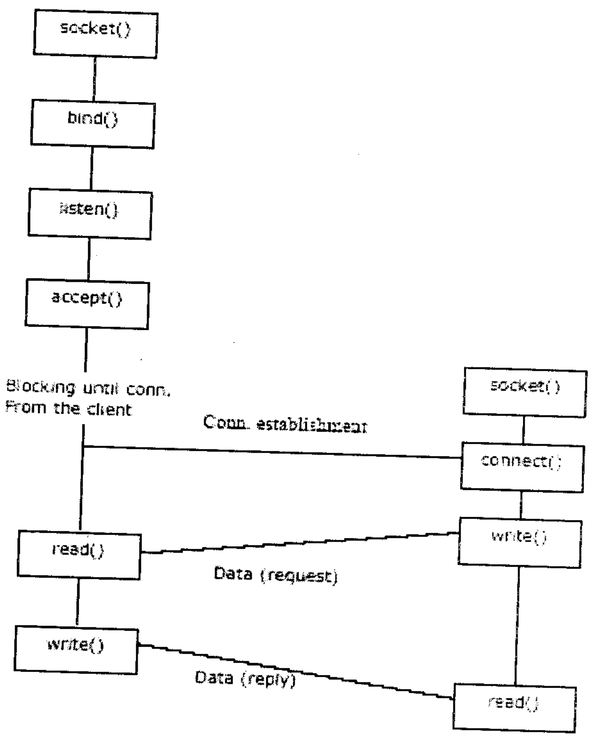
3.3. CREATION OF SECURE SOCKET LAYER

A socket is an interface between the application layer and transport layer within a host. It is also referred to as API (Application Programmers Interface) between the application and the network. The communication between the application layer, secure socket layer and transport layer is established by port. The port values are statically allocated by the program.

3.3.1CONNECTION ORIENTED COMMUNICATION:

A client and server must select the transport protocol that supports a connection-oriented service before sending data. To establish a connection, the applications interact with transport protocol software on the local computer and the two transport protocol module exchange messages across the network. After both sides agree that the connection has been established, the applications can send data.

3.3.2 SEQUENCE OF SOCKET PROCEDURE CALLS

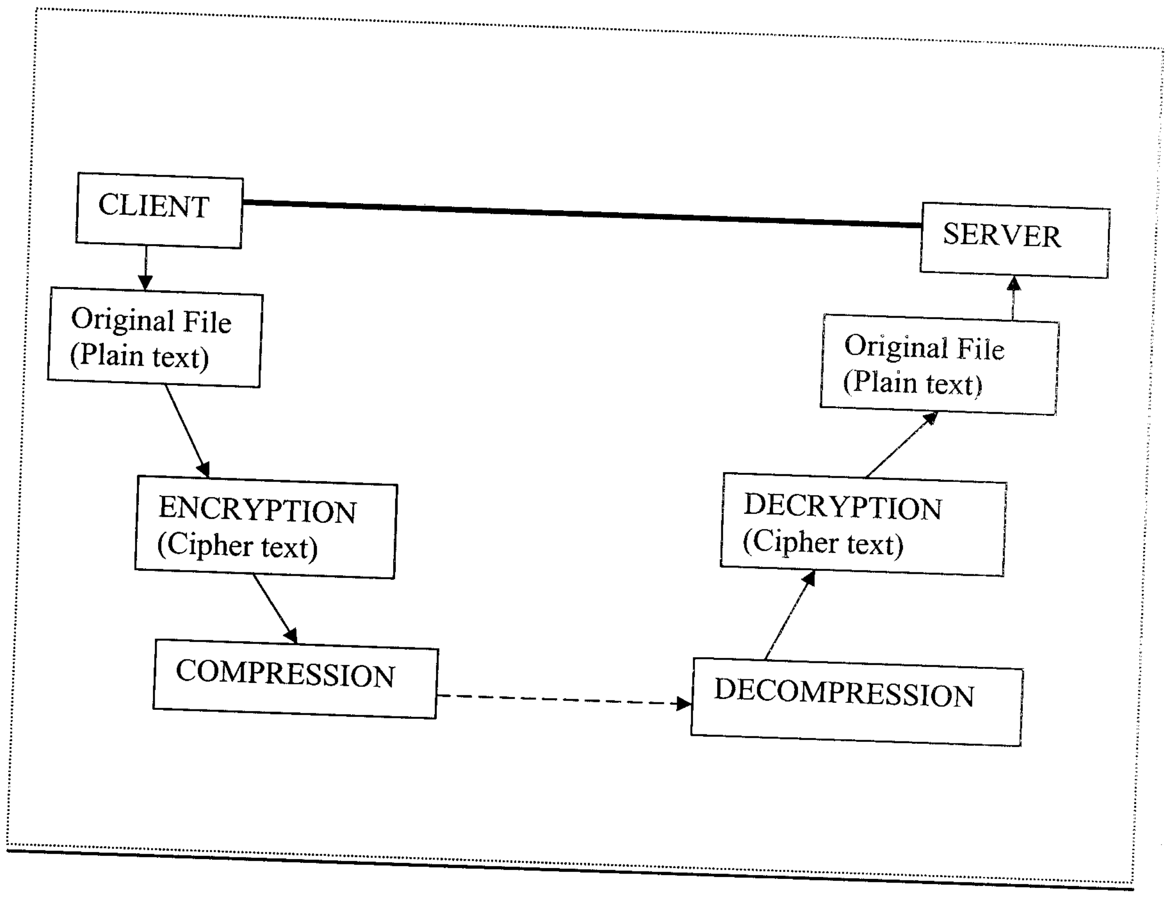


On the client side, a structure is initialized with information such as port number, address family and a socket is created in stream mode by calling *socket*. Now connection is done with the server and a thread is created. Inside the thread the send function is repeatedly called until all the data is sent. Now the socket is released by calling *close*.

On the server side, a socket is created in stream mode by calling *socket*. The server calls *bind* to specify a local port for the socket and listen to place the socket in passive mode. The server then enters an infinite loop in which it calls *accept* to accept the next incoming request. A thread is created and *recv* is called until all the data has been received. Now the socket is released by calling *close*.

DESIGN DETAILS

4. DESIGN DETAILS



FLOW DIAGRAM

MODULE 1:

ENCRYPTION AND DECRYPTION:

Public and private keys are generated on the server side, every time the text file has to be sent from the client to the server. The prime numbers used to generate these keys are got as input from the user.On the client side the public key generated by the server is used to encrypt the plain text using RSA algorithm. On the server side, the private key is used to regenerate the original plain text from the cipher text.

MODULE 2:

COMPRESSION AND DECOMPRESSION:

The encrypted file is usually of a larger size, when compared to the original text file. For ease of transmission across the network, the file is compressed by using a series of algorithms. The sequence followed is

1. RUN LENGTH ENCODING
2. BURROWS-WHEELER TRANSFORM
3. MOVE TO FRONT ALGORITHM
4. RUN LENGTH ENCODING

On the receiving side, the sequence of the algorithms applied is reversed to get back the file from the compressed format.

MODULE 3:

CREATION OF SECURE SOCKET LAYER AND TRANSMISSION

A socket is an interface between the application layer and transport layer within a host. It is also referred to as API (Application Programmers Interface) between the application and the network.

On the client side, a structure is initialized with information such as port number, address family and a socket is created in stream mode by calling *socket*. Now connection is done with the server and a thread is created. Inside the thread the send function is repeatedly called until all the data is sent. Now the socket is released by calling *close*.

On the server side, a socket is created in stream mode by calling *socket*. The server calls *bind* to specify a local port for the socket and listen to place the socket in passive mode. The server then enters an infinite loop in which it calls *accept* to accept the next incoming request. A thread is created and *recv* is called until all the data has been received. Now the socket is released by calling *close*.

SYSTEM TESTING

5. SYSTEM TESTING

Testing is an activity to verify that a correct system is being built and is performed with the intent of finding faults in the system. Testing is an activity, however not restricted to being performed after the development phase is complete. But this is to be carried out in parallel with all stages of system development, starting with requirement specification. Testing results once gathered and evaluated, provide a qualitative indication of software quality and reliability and serve as a basis for design modification if required.

System Testing is a process of checking whether the development system is working according to the original objectives and requirements. The system should be tested experimentally with test data so as to ensure that the system works according to the required specification. When the system is found working, test it with actual data and check performance.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. The increasing visibility of software as a system element and the attendant "cost" associated with a software failure are the motivation forces for a well planned, thorough testing.

5.1 TESTING OBJECTIVES:

The testing objectives are summarized in the following three steps. Testing is the process of executing a program with the intent of finding an error. A good test case is one that has high probability of finding an error. A successful test is one that uncovers as –yet-undiscovered errors.

5.2 TESTING PRINCIPLES:

All tests should be traceable to customer requirements. Tests should be planned long before testing begins, that is, the test planning can begin as soon as the requirements model is complete. Testing should begin "in the small" and progress towards resting "in large". The focus of testing will shift progressively from programs to individual modules and finally to the entire project. Exhaustive testing is not possible. To be more effective, testing should be one, which has highest probability of finding errors.

The following are the attributes of good tests:

- A good test has a high probability of finding an error.
- A good test is not redundant.
- A good test should be "best of breed"
- A good test should be neither too simple nor too complex.

"Software Testing is the process of uncovering the defects of the function, in logic, in implementation with the objective of assuring that the defined inputs will produce actual results that agree with the required results"

Testing is the important phase of Software development cycle and no software is complete without putting it to severe testing.

5.3 LEVELS OF TESTING:

The details of the software functionality tests are given below. The testing procedure that has been used is as follow:

- ➢ Unit Testing
- ➢ Integration Testing
- ➢ Validation Testing

Unit Testing:

- The compression and decompression modules were tested using text files of varying sizes to determine the amount of compression achieved .The test cases in this process were many text files of sizes 985KB, 1.2 MB which were compressed to 450 KB, 625 KB i.e. nearly 50%.
- The encryption and decryption modules were tested using client and server .And found that original file sent was retrieved at the other end as it was sent at the sender side.

Integrated Testing:

The main modules were integrated and the files to be securely transferred were compressed, sent, received, and decompressed, as required.

The modules to make the final system acceptable used the basic functionalities supported by the system .This was also found in good stead.

Validation Testing:

At the end of integration testing, software is completely assembled as a package, interfacing errors have been uncovered and correction testing begins.

Validation Test Criteria:

Software testing and validation is achieved through series of black box tests that demonstrate conformity with the requirements are achieved, documentation is correct and other requirements are met. Here the 3 modules which are checked in the integration testing are assembled and programmed in the code is testing for any correction.

CONCLUSION AND FUTURE OUTLOOK

6. CONCLUSION AND FUTURE OUTLOOK

"Security is a continuous evolving process and not a product"-Bruce Scheiner. Total security is an academics deal. Well-performed security involves risk. The level of risk involved depends on how well the technology is understood and applied. Protection guarantees should be equal to the level of need.

Information that travels from a client workstation and servers ,through out the network is susceptible to fraud and misuse by other parties. The network does not provide a built in mechanism (security system) that will prevent the intermediaries from deceiving us, eaves dropping, copying from, damaging the communication etc.

This project provides the required security for documents going around the network along with compression which is an enhanced feature.This project can be enhanced , to provide message authenticity and message integrity, which are key features when it comes to security of transmitted information on the local network.

Security is important on the Internet. Whether sharing financial, business or personal information, people want to know with whom they are communicating (authentication), to ensure that what is sent is what is received (integrity), and to prevent others from eaves dropping in their communications (privacy).This project can be enhanced to provide security on the Internet environment.

REFERENCES

7. REFERENCES

BOOKS:

1. William Stallings, "Cryptography and Network Security", Prentice Hall, second Edition ,1999.
2. Andrew S. Tanenbaum "Computer Networks", Prentice-Hall India ,Third Edition,2001.
3. David J. Kruglinski "Programming Visual VC++" First Edition ,WP Publishers India,1999.
4. Douglas E. Comer "Computer Networks and Internets" ,Prentice-Hall India, second edition ,2003.

JOURNALS:

George Apostolopolous ,Vinod Peris ,Debanjan Saha "Transport Layer Security;How much does it really cost? " IEEE Network Security July/August 2000
Paper by Mark Nelson on "Data Compression".

WEBSITES:

1. http:///openssl.com dt.25-11-03
2. http://rsa/security.com dt .26-11-03
3. http://www.netscape.com/assist/security/ssl/protocol.html dt.27-11-03
4. http://datacompression.info/algorithms/rle dt.8-12-03
5. http://autrocampos.com/ac.mtf.html dt.17-12-03
6. http://autrocampos.ac.rle.html dt.18-12-03
7. http://security.com dt.2-2-04

APPENDIX

8.APPENDIX

8.1 SAMPLE SOURCE CODE:

CLIENT SIDE

```
// SSL_CommDlg.cpp : implementation file
#include "stdafx.h"
#include "SSL_Comm.h"
#include "SSL_CommDlg.h"
#include "NewDecompDlg.h"
#include "math.h"
#define _WIN32_WINNT 0x0400
#include "bwt.h"
#include "windows.h"
#include "wincrypt.h"
#include "stdio.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
class CAboutDlg : public CDialog
{ public:    CAboutDlg();
// Dialog Data
    enum { IDD = IDD_ABOUTBOX };
    // ClassWizard generated virtual function overrides
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);
// Implementation
protected:  DECLARE_MESSAGE_MAP() };
```

```
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{ }
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{   CDialog::DoDataExchange(pDX);
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()
// CSSL_CommDlg dialog
CSSL_CommDlg::CSSL_CommDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CSSL_CommDlg::IDD, pParent)
{   m_e = 0; m_n = 0;     m_hIcon = AfxGetApp()-LoadIcon(IDR_MAINFRAME);
}
void CSSL_CommDlg::DoDataExchange(CDataExchange* pDX)
{   CDialog::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_EDIT_PASS, m_edPassWd);
    DDX_Control(pDX, IDC_EDIT2, m_edStat);
    DDX_Control(pDX, IDC_EDIT1, m_edIP);
    DDX_Control(pDX, IDC_ED_SRCFILENAME, m_ed_SrcFName);
    DDX_Control(pDX, IDC_ED_DSTFILENAME, m_ed_DstFName);
    DDX_Text(pDX, IDC_EDIT_PUBLIC, m_e);
    DDX_Text(pDX, IDC_EDIT_PUBLIC_TWO, m_n);}

BEGIN_MESSAGE_MAP(CSSL_CommDlg, CDialog)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_BUTTONOPEN, OnButtonopen)
    ON_BN_CLICKED(IDC_BUTTON_BROWSE, OnButtonBrowse)
    ON_BN_CLICKED(IDC_BTN_COMPRESS, OnBtnCompress)
    ON_BN_CLICKED(IDC_BTN_DECOMP, OnBtnDecomp)
    ON_BN_CLICKED(IDC_BUTTON2, OnButton2)
    ON_BN_CLICKED(IDC_BUTTONENCRYPT, OnButtonencrypt)
    END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
void CSSL_CommDlg::OnButtonopen()
{   CString strFilePath(""); TCHAR szPath[MAX_PATH] = "\0";
    g_strCurrPath = ""; ::GetCurrentDirectory (MAX_PATH,szPath);
    g_strCurrPath = szPath; BOOL bFileOpen = TRUE; CFileDialog
cf(bFileOpen,NULL,NULL,OFN_HIDEREADONLY,NULL,NULL);
    cf.DoModal ();  strFilePath = cf.GetPathName ();
    m_strSrcFName = strFilePath; m_ed_SrcFName.SetWindowText (strFilePath);}

void CSSL_CommDlg::OnButtonBrowse()
{   CString strFilePath(""); BOOL bFileOpen = FALSE;
    CFileDialogcf(bFileOpen,NULL,NULL,OFN_HIDEREADONLY,NULL,NULL)
;   cf.DoModal ();strFilePath = cf.GetPathName ();
    m_strDstFName = strFilePath;m_ed_DstFName.SetWindowText (strFilePath);
    m_strTransFName = strFilePath; }
void CSSL_CommDlg::OnBtnCompress()
{   CWaitCursor wc;  Compress(m_strSrcForComp,m_strDstFName);}
void CSSL_CommDlg::OnButton1()
{   char ipaddress[35]; m_edIP.GetWindowText(ipaddress,30);
    cli.sin_addr.s_addr=inet_addr(ipaddress);cli.sin_family=AF_INET;
    cli.sin_port=htons(5000); clisock=socket(AF_INET,SOCK_STREAM,0);
    ee=1;  AfxBeginThread(thread,0);
}
void CSSL_CommDlg::OnButton2() //send button
{   char buff[100]; CSize size;    size.cx=0;      size.cy=30;
    LPCTSTR lpctBuff = NULL; LPTSTR lpBuff;
    CString strTmp(""),strPath("\\\\"),strTmp1("\\"),strTmp2("\\");
    ::GetComputerName (strTmp.GetBuffer (dwLen),&dwLen);
    strTmp = strTmp.GetBuffer (strTmp.GetLength ()+1);
    strPath = strPath+strTmp; // computer name
    int j = m_strTransFName.ReverseFind ('\\');
    LPCTSTR lpFName = m_strTransFName;    CFile cf;
    DWORD dwFileLen = 0;        CString strFLen("");
    cf.Open (lpFName,CFile::modeRead);dwFileLen  = cf.GetLength ();
    cf.Close ();      strFLen.Format ("%d",dwFileLen);
    strTmp1 += m_strTransFName.Mid (j+1); // file
    m_strTransFName = m_strTransFName.Left (j);
    j = m_strTransFName.ReverseFind ('\\');
    strTmp2 += m_strTransFName.Mid (j+1); // folder
    strTmp.Empty ();        strTmp = strPath + strTmp2 + strTmp1;
    strTmp += "\t";         strFLen += "\t";
    strTmp  = strFLen + strTmp ;            AfxMessageBox(strTmp);
    lpctBuff = strTmp;      nSize = strTmp.GetLength();
    send(clisock,lpctBuff,nSize,0); }
void CSSL_CommDlg::BeginEncrypt (CString strSrcFName,CString strDstFName)
{
   CString strPassword(""); strSrcFName.TrimLeft ();    strSrcFName.TrimRight ();
  strDstFName.TrimLeft (); strDstFName.TrimRight ();strPassword = m_strPassWd;
  m_strSrcForComp = strDstFName;
  if(EncryptFile(strSrcFName, strDstFName))
  {    AfxMessageBox("Encryption Successful"); }
    else {   AfxMessageBox("Error encrypting file!");    return; }
}
BOOL CSSL_CommDlg::EncryptFile (CString strSource,CString strDst)
{
    UpdateData(TRUE);   int NO_BITS=32; double c=0,d=1;    char bits[100];
```

```
double n=(double)m_n; unsigned char ch;        double data;//19;
    long i,k=NO_BITS;   CString str;     int sizeof_d=sizeof(double);
    LPCTSTR szSource = "\0";   LPCTSTR szDst = "\0";
    szSource = strSource; szDst = strDst; FILE *inFP, *outFP;
    inFP=fopen(szSource,"rt");
    if(inFP==NULL)
    {AfxMessageBox("Couldn't open input file");return FALSE;}
    outFP=fopen(szDst,"wb");
    if(outFP==NULL)
    {AfxMessageBox("Couldn't create output file");fcloseall(); return FALSE;}
    //change integer 'm' to binary
    D_to_B(m_e,32,bits); GetOnlyProperBits(bits);k=NO_BITS = strlen(bits)-1;
    while(1)
    {       //read from ip file into 'data'
            if(fread(&ch,1,1,inFP)==0)break;data =(double)ch;
            //calculate ((data)^e mod n) .i.e the remainderc=0;d=1;
            for(i=k;i>=0;i--)
            {  c=2*c; d=fmod(d*d,n);
            if(bits[NO_BITS-i] == '1') {c=c+1;d=fmod(data*d,n);}
            }
            //write to op file from 'd'
            fwrite(&d,sizeof_d,1,outFP);
    }//end of while loop
    fcloseall(); return TRUE;
}
void CSSL_CommDlg::OnButtonencrypt()
{   CString strFile(""),strPassWd(""); int npos = -1;
    strFile = g_strTmpFName; strFile.TrimLeft ();strFile.TrimRight ();
    npos = strFile.ReverseFind ('\\');strFile = strFile.Left (npos);
```

```
strFile = "C:\\tmp5.txt";CString strSrcToEnc("");
    BeginEncrypt(strSrcToEnc,strFile);
}
```

SERVER SIDE

```
 SSL_Comm_ServDlg.cpp : implementation file
#include "stdafx.h"
#include "SSL_Comm_Serv.h"
#include "SSL_Comm_ServDlg.h"
#include "DeComp_New.h"
#include "math.h"
#define _WIN32_WINNT 0x0400
#include <stdio.h>
#include <windows.h>
CString m_strIncmgFPathClt(""); int nFLen = 0;
 CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{public:    CAboutDlg();
enum { IDD = IDD_ABOUTBOX };
protected:  virtual void DoDataExchange(CDataExchange* pDX);
protected:  DECLARE_MESSAGE_MAP() };
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()
CSSL_Comm_ServDlg::CSSL_Comm_ServDlg(CWnd* pParent /*=NULL*/)
: CDialog(CSSL_Comm_ServDlg::IDD, pParent)
{   m_prime1 = 0; m_prime2 = 0;
m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME); }
```

```
void CSSL_Comm_ServDlg::DoDataExchange(CDataExchange* pDX)
{CDialog::DoDataExchange(pDX);DDX_Control(pDX,IDC_PRIVATE_KEY,m_kr)
 DDX_Control(pDX, IDC_PUBLIC_KEY, m_ku);
   DDX_Control(pDX, IDC_EDIT_PASSWORD, m_edPassWd);
   DDX_Control(pDX, IDC_BUTTONDECOMP, m_btnDecomp);
   DDX_Control(pDX, IDC_LIST2, m_list);
   DDX_Control(pDX, IDC_ED_DECOMP_FILE1, m_ed_Decomp_File);
   DDX_Control(pDX, IDC_ED_COMP_FILE1, m_ed_Comp_File);
   DDX_Text(pDX, IDC_PRIME1, m_prime1);
   DDX_Text(pDX, IDC_PRIME2, m_prime2);}

BEGIN_MESSAGE_MAP(CSSL_Comm_ServDlg, CDialog)
   ON_WM_SYSCOMMAND()
   ON_WM_PAINT()
   ON_WM_QUERYDRAGICON()
   ON_BN_CLICKED(IDC_BUTTON_DECOMPFILE, OnButtonDecompfile)
   ON_BN_CLICKED(IDC_BUTTONSAVE, OnButtonsave)
   ON_BN_CLICKED(IDC_BUTTONDECOMP, OnButtondecomp)
   ON_BN_CLICKED(IDC_BUTTONDECRYPT, OnButtondecrypt)
   ON_BN_CLICKED(IDC_GEN_KEY, OnGenKey)
END_MESSAGE_MAP()
void CSSL_Comm_ServDlg::OnButtonDecompfile()
{
   CString strFile(""); BOOL bFileOpen = TRUE;
   CFileDialog
dlg(bFileOpen,NULL,NULL,OFN_HIDEREADONLY,NULL,NULL);
   dlg.DoModal ();strFile = dlg.GetPathName ();
   m_strCompFName = strFile ;}
void CSSL_Comm_ServDlg::OnButtonsave()
```

```
{   BOOL bFileOpen = FALSE;  CString strFilePath("");
    CFileDialog
dlg(bFileOpen,NULL,NULL,OFN_HIDEREADONLY,NULL,NULL);
    dlg.DoModal ();strFilePath = dlg.GetPathName ();
    m_strOrgFName = strFilePath;}

void CSSL_Comm_ServDlg::OnButtondecomp()
{   CWaitCursor cw;
    WriteToTmpFile(); }
void CSSL_Comm_ServDlg::WriteToTmpFile ()
{
    cf.Open (lpFName,CFile::modeRead|CFile::typeBinary|CFile::shareExclusive);
    cf1.Open (lpTmpFname,CFile::modeCreate|CFile::modeWrite);
    dwLen = nFLen;
    while(bFlag)
    { cf.Read(szBuff,1);  cf1.Write (szBuff,1);  i++;
    if(i >= dwLen)          bFlag = FALSE;}
    cf.Close ();cf1.Close ();
    DeCompressFile(strTmpFname,strDstForDecomp); }
void CSSL_Comm_ServDlg::OnButtondecrypt()
{
    CString strPassword("");
    m_edPassWd.GetWindowText (strPassword);
    CString strSrcForDecrp("C:\\TempDecomp.txt");
    if(DecryptFile(strSrcForDecrp,m_strOrgFName))
            AfxMessageBox("Deccryption successful");
    else
            AfxMessageBox("Deccryption Failed");
}
BOOL CSSL_Comm_ServDlg::DecryptFile(CStringstrSource, CString trDestination)
```

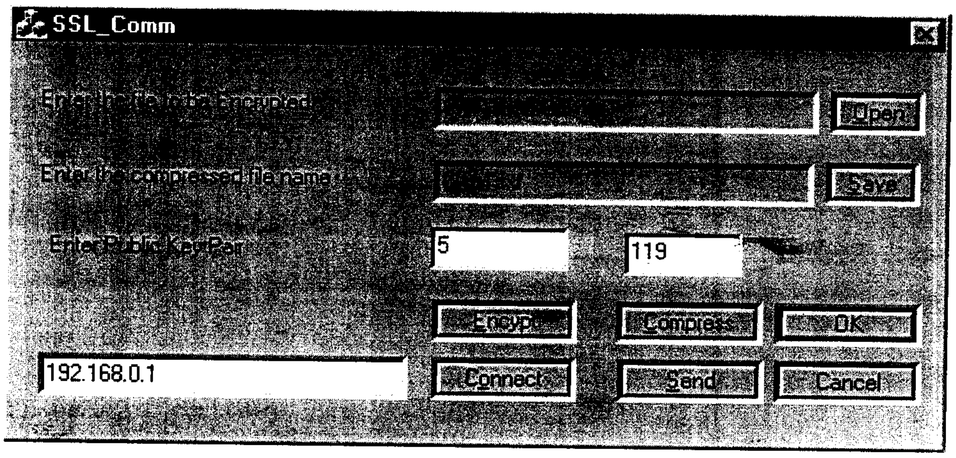
```
{
    int NO_BITS; double c=0,d=1;char bits[100] ; double n=(double)m_n;
    double data;//19; long i,k; CString str;int sizeof_d=sizeof(double);
    szSource = strSource; szDest = strDestination;
    FILE *inFP, *outFP;

    inFP = fopen(szSource,"rt+");
    if(inFP == NULL)
    {AfxMessageBox("Couldn't open input file");return FALSE;}
    outFP=fopen(szDest,"wt+");
   if(outFP==NULL){AfxMessageBox("Couldn't create output file");}
    //change integer 'd' to binary
    D_to_B(m_d,32,bits); GetOnlyProperBits(bits);k=NO_BITS = strlen(bits)-1;
    while(1)
    {//read from ip file into 'data'
            if(fread(&data,sizeof_d,1,inFP)==0) break;
            //calculate ((data)^e mod n) .i.e the remainder c=0;d=1;
            for(i=k;i>=0;i--)
            {c=2*c;d=fmod(d*d,n);if(bits[NO_BITS-i] == '1')
                    {c=c+1;d=fmod(data*d,n);}}
            //write to op file from 'd'      unsigned char ch=(unsigned char)d;
            fprintf(outFP,"%c",ch);}
    fcloseall();      return TRUE; }
void CSSL_Comm_ServDlg::OnGenKey()
{   Execute(); m_key_gen_done_flg = true;}
void CSSL_Comm_ServDlg::Execute ()
{
    CString str;    GeneratePrimeNumbers();    m_n = m_Prime1 * m_Prime2;
    m_Undef = (m_Prime1-1) * (m_Prime2-1);  SelectE();      CalculateD();
    m_ku.SetWindowText(str);   m_kr.SetWindowText(str); }
```

```
void CSSL_Comm_ServDlg::GeneratePrimeNumbers()
{
    CString str;    UpdateData(TRUE);
}
void CSSL_Comm_ServDlg::SelectE()
{
    CString str;
    ///algo: EUCLID
    for(float i=2;i<100000;i++)
    { if(IsRelativePrime((float)m_Undef,(float)i))
            {m_e=(long)i; WriteValues(str);return;}}}
bool CSSL_Comm_ServDlg::IsRelativePrime(float X,float Y)
{
    float R;///algo: EUCLID
    //X is always great than Y at this point
   for(;;)
    {if(Y==0)  break; R=fmod(X,Y);X=Y;Y=R;}
    if(X != 1) return false;
    else return true;//relatively prime }

void CSSL_Comm_ServDlg::CalculateD()
{
    float d; long d_dash;   CString str;
    for(float k=1;k<100000;k++)
    {       d=(m_Undef*k+1)/m_e;
            d_dash = (long)((m_Undef*k+1)/m_e);
            if(d == d_dash)
            {    m_d=d;
                    return;
            }
}}
```

8.2 SAMPLE OUTPUT:



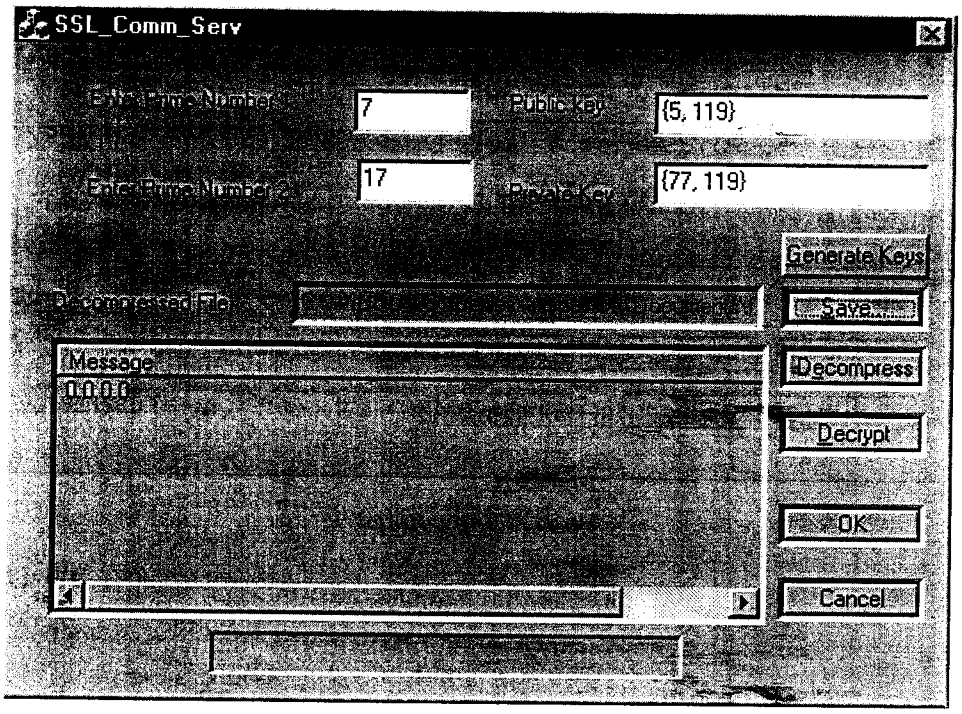
SSL_Comm

Enter the file to be Encrypted

Enter the compressed file name

Enter Public Key Pair: 5 119

192.168.0.1



SSL_Comm_Serv

Enter Prime Number 1: 7

Enter Prime Number 2: 17

Public key: {5, 119}

Private Key: {77, 119}

Decompressed File

Message

0.0.0.0