الملاحق

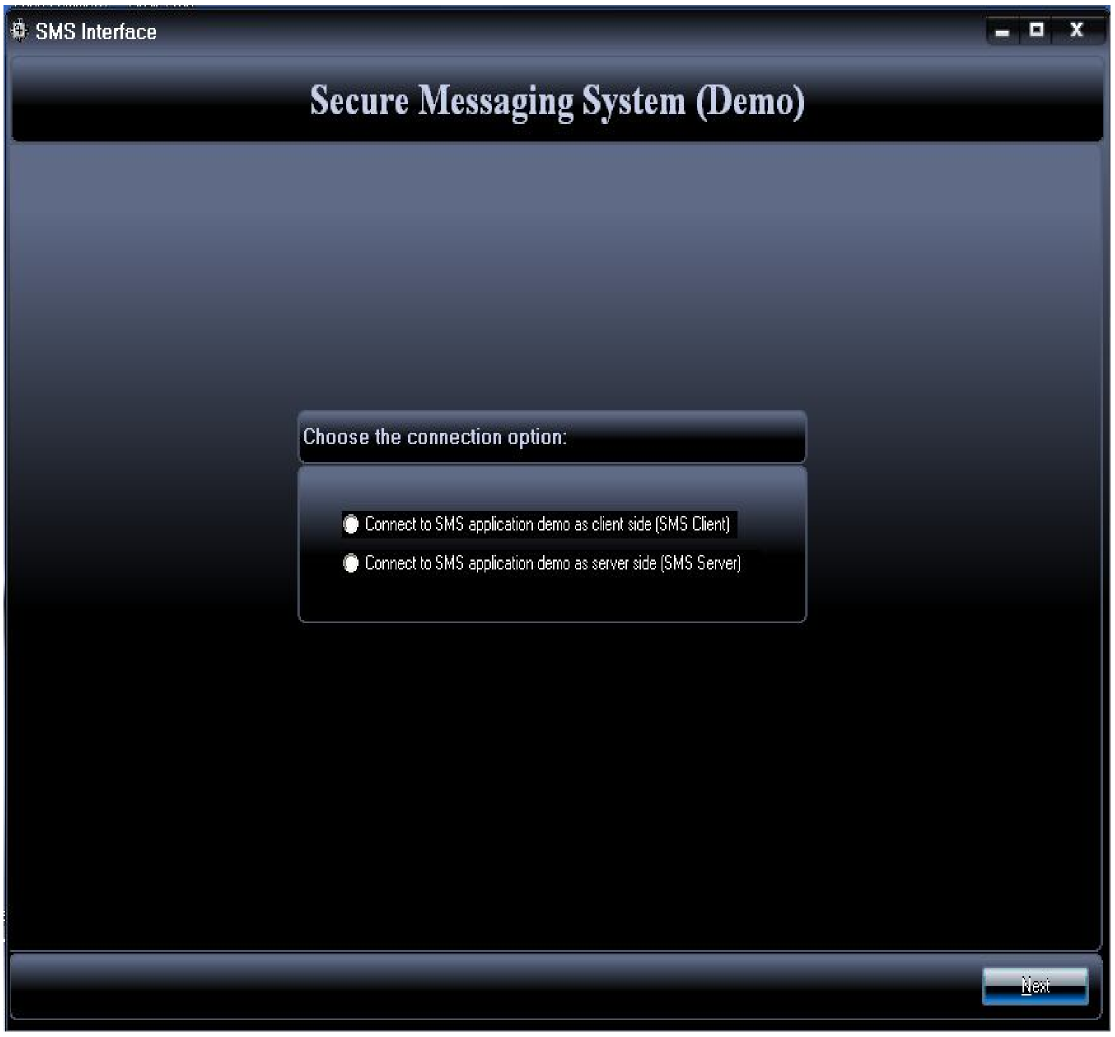
Appendences

الملحق (أ)
واجهات النظام الأساسية

فيما يلي مجموعة أشكال لواجهات النظام الأساسية وهي تنقسم إلى ثلاثة أجزاء:

1. الواجهة الرئيسية للنظام
2. الواجهات الخاصة بجزئية الزبون في النظام
3. الواجهات الخاصة بجزئية المخدّم في النظام

1. الواجهة الرئيسية للنظام


SMS Interface

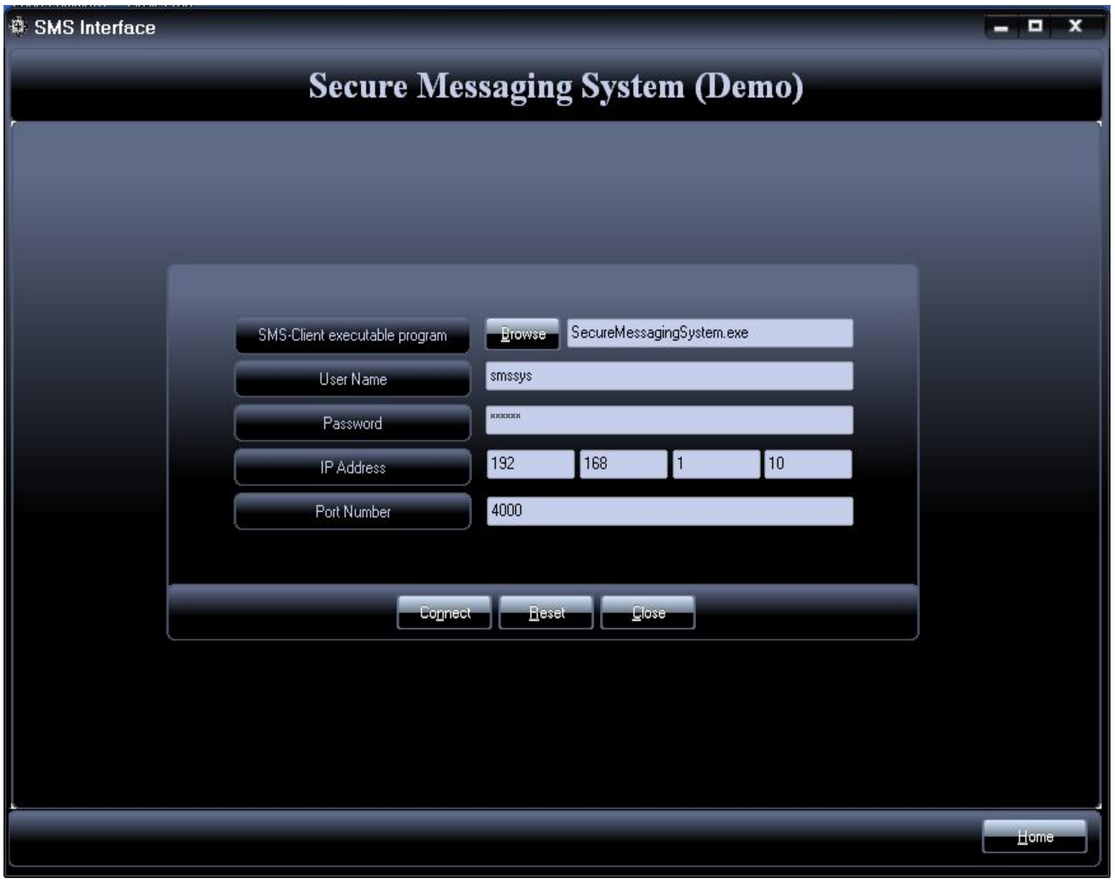
Secure Messaging System (Demo)

Choose the connection option:

- Connect to SMS application demo as client side (SMS Client)
- Connect to SMS application demo as server side (SMS Server)

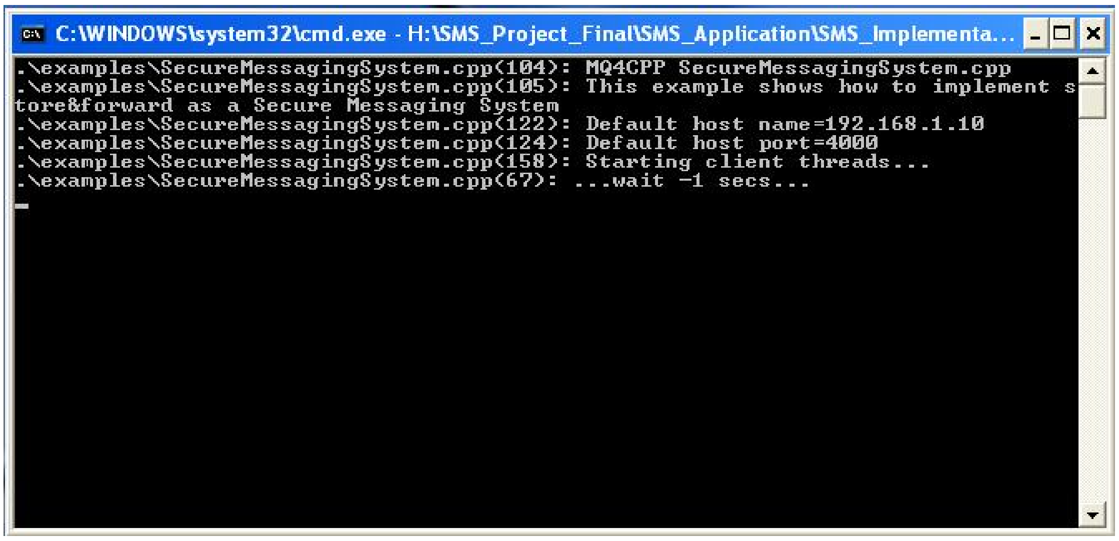
يتم أولاً إختيار طريقة الإتصال وهى إما الإتصال كزبون (الخيار الأول) أو الإتصال كمخدّم (الخيار الثاني).

2. الواجهات الخاصة بجزئية الزبون في النظام



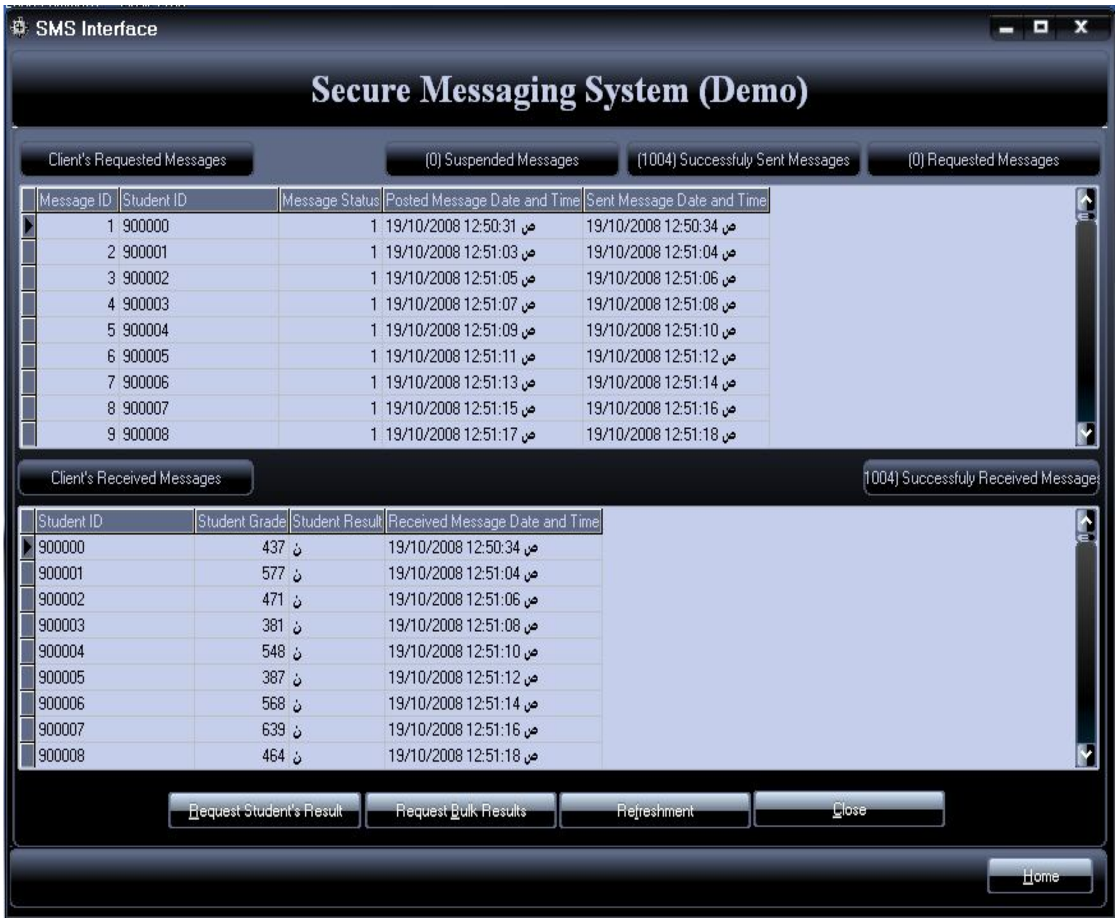
SMS-Client executable program	SecureMessagingSystem.exe
User Name	smssys
Password	xxxxxx
IP Address	192 168 1 10
Port Number	4000

عند إختيار الإتصال كزبون من واجهة النظام الأساسية تظهر هذه الواجهة وعلى المستخدم إدخال كل المعلومات الضرورية لإتمام تشغيل نظام الزبون.


C:\WINDOWS\system32\cmd.exe - H:\SMS_Project_Final\SMS_Application\SMS_Implementa...

```
.\examples\SecureMessagingSystem.cpp(104): MQ4CPP SecureMessagingSystem.cpp
.\examples\SecureMessagingSystem.cpp(105): This example shows how to implement s
tore&forward as a Secure Messaging System
.\examples\SecureMessagingSystem.cpp(122): Default host name=192.168.1.10
.\examples\SecureMessagingSystem.cpp(124): Default host port=4000
.\examples\SecureMessagingSystem.cpp(158): Starting client threads...
.\examples\SecureMessagingSystem.cpp(67): ...wait -1 secs...
_
```

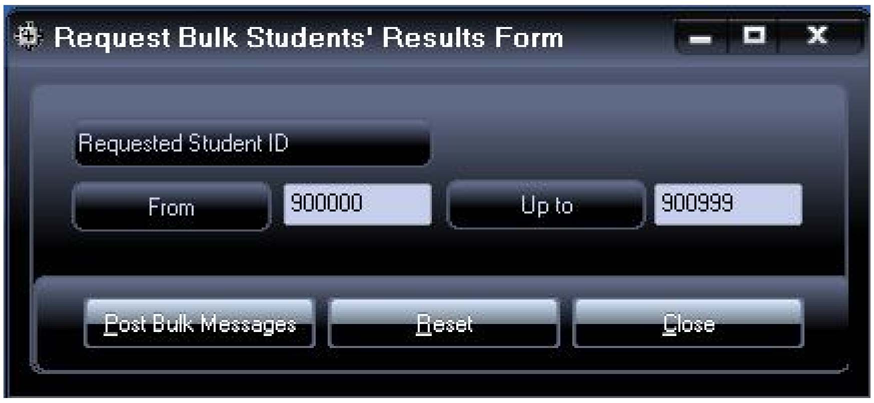
تظهر هذه الشاشة في خلفية واجهة الزبون، وذلك عندما يتم تشغيل نظام الزبون الخاص بنظام التراسل الآمن بنجاح.


SMS Interface
Secure Messaging System (Demo)
Client's Requested Messages
(0) Suspended Messages
(1004) Successfuly Sent Messages
(0) Requested Messages
Client's Received Messages
1004) Successfuly Received Message
Request Student's Result
Request Bulk Results
Refreshment
Close
Home

Message ID	Student ID	Message Status	Posted Message Date and Time	Sent Message Date and Time
1	900000	1	19/10/2008 12:50:31 ص	19/10/2008 12:50:34 ص
2	900001	1	19/10/2008 12:51:03 ص	19/10/2008 12:51:04 ص
3	900002	1	19/10/2008 12:51:05 ص	19/10/2008 12:51:06 ص
4	900003	1	19/10/2008 12:51:07 ص	19/10/2008 12:51:08 ص
5	900004	1	19/10/2008 12:51:09 ص	19/10/2008 12:51:10 ص
6	900005	1	19/10/2008 12:51:11 ص	19/10/2008 12:51:12 ص
7	900006	1	19/10/2008 12:51:13 ص	19/10/2008 12:51:14 ص
8	900007	1	19/10/2008 12:51:15 ص	19/10/2008 12:51:16 ص
9	900008	1	19/10/2008 12:51:17 ص	19/10/2008 12:51:18 ص

Student ID	Student Grade	Student Result	Received Message Date and Time
900000	437	ن	19/10/2008 12:50:34 ص
900001	577	ن	19/10/2008 12:51:04 ص
900002	471	ن	19/10/2008 12:51:06 ص
900003	381	ن	19/10/2008 12:51:08 ص
900004	548	ن	19/10/2008 12:51:10 ص
900005	387	ن	19/10/2008 12:51:12 ص
900006	568	ن	19/10/2008 12:51:14 ص
900007	639	ن	19/10/2008 12:51:16 ص
900008	464	ن	19/10/2008 12:51:18 ص

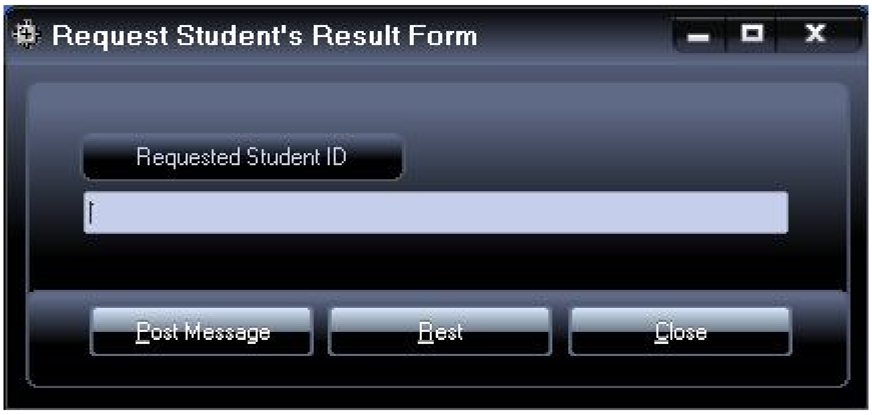
تظهر هذه الشاشة مباشرة بعد تشغيل نظام الزبون الخاص بنظام التراسل الآمن، وهي تعمل كواجهة لجداول قاعدة بيانات نظام الزبون. تقوم هذه الواجهة بإظهار تقارير تفصيلية وتجميعية متجددة مع كل حدث يحدث في قاعدة البيانات.



Requested Student ID

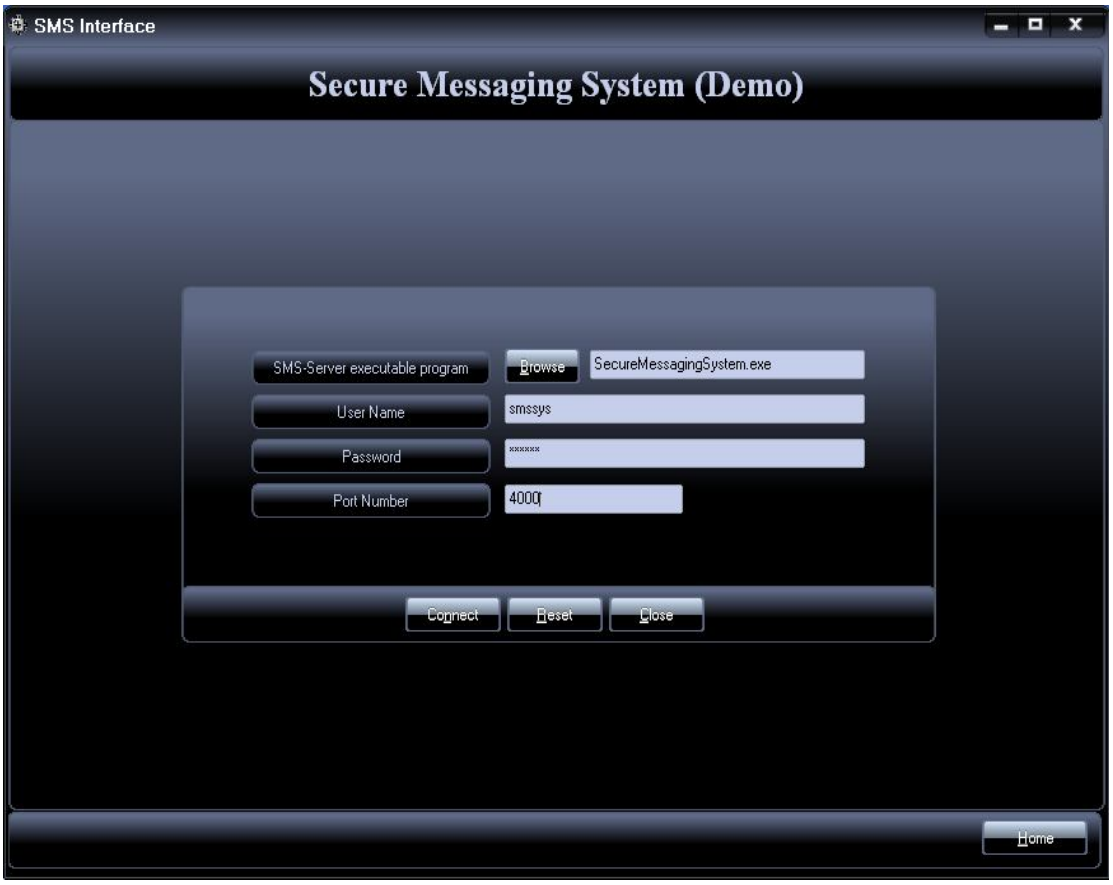
From: 900000 Up to: 900999

تظهر هذه الواجهة عند الضغط على ذر (Request Bulk Results) وهي تفيد في إدخال مجموعة رسائل طلبات متواصلة، وذلك بهدف إختبار نظام التراسل الآمن.



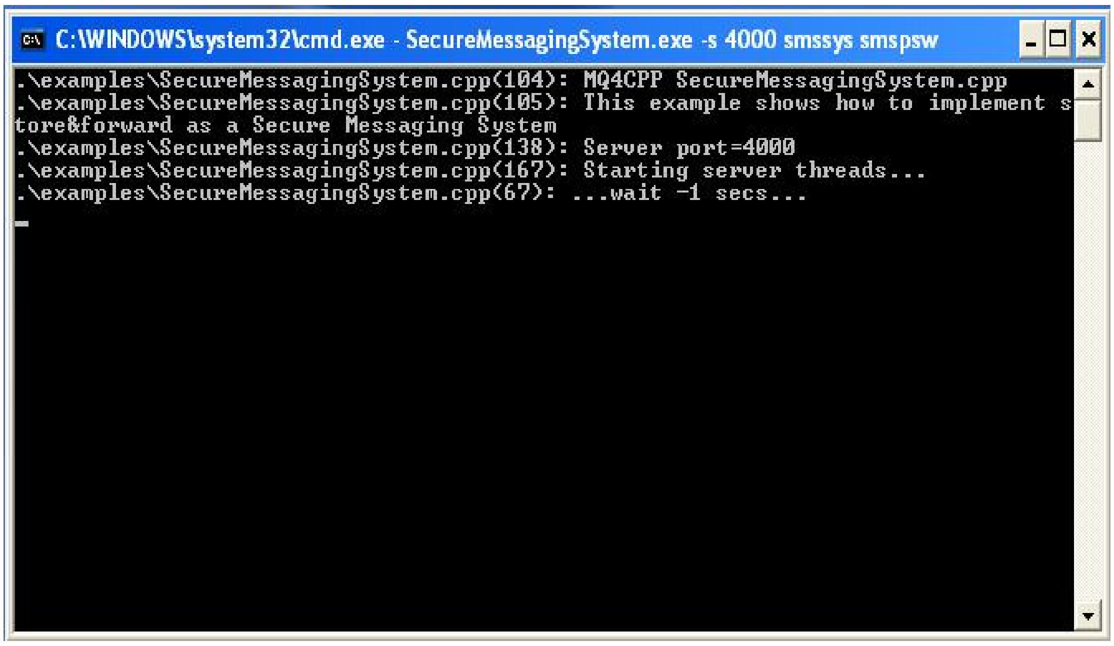
تظهر هذه الواجهة عند الضغط على ذر (Request Student’s Results) وهي تفيد في إدخال رسالة طلب واحدة معينة، وذلك بهدف إختبار نظام التراسل الآمن.

3. الواجهات الخاصة بجزئية المخدّم في النظام



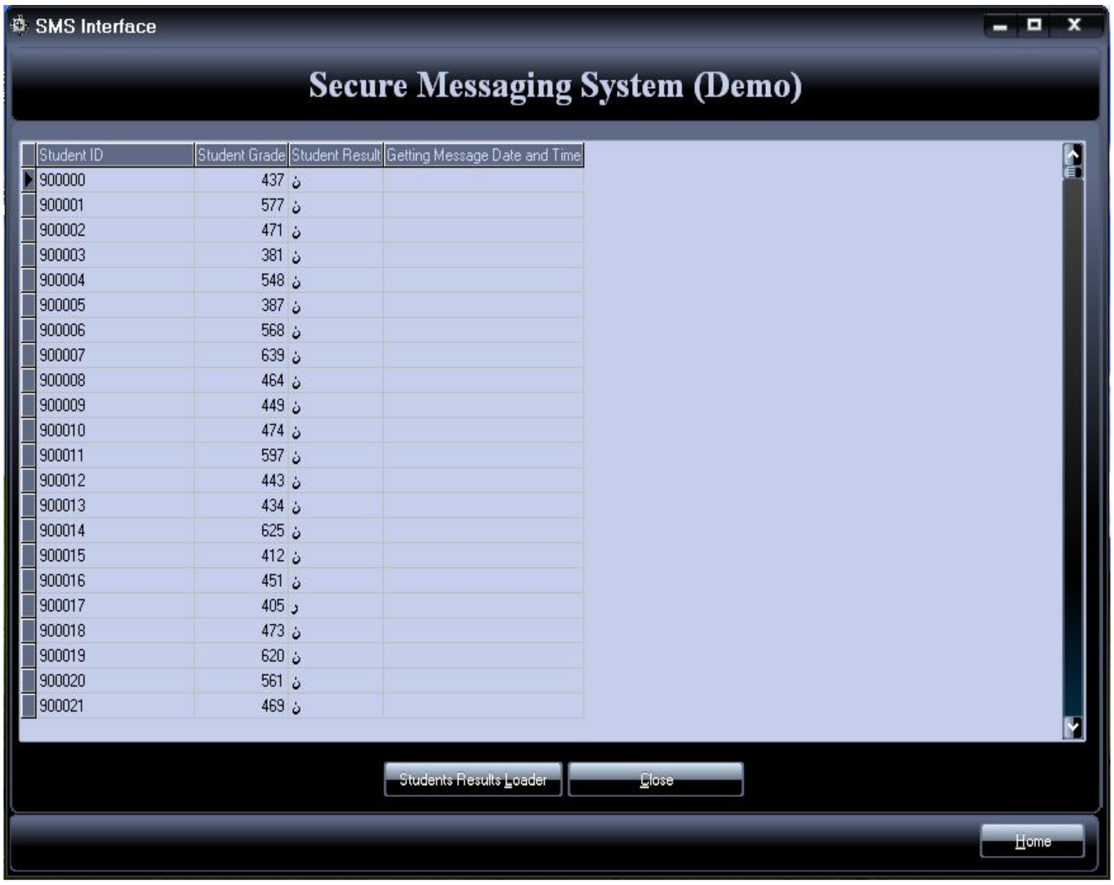
SMS-Server executable program: SecureMessagingSystem.exe
User Name: smssys
Password: ******
Port Number: 4000

عند إختيار الإتصال كمخدّم من واجهة النظام الرئيسية تظهر هذه الواجهة وعلى المستخدم إدخال كل المعلومات الضرورية لإتمام تشغيل نظام المخدّم.


C:\WINDOWS\system32\cmd.exe - SecureMessagingSystem.exe -s 4000 smssys smspsw

```
.\examples\SecureMessagingSystem.cpp(104): MQ4CPP SecureMessagingSystem.cpp
.\examples\SecureMessagingSystem.cpp(105): This example shows how to implement s
tore&forward as a Secure Messaging System
.\examples\SecureMessagingSystem.cpp(138): Server port=4000
.\examples\SecureMessagingSystem.cpp(167): Starting server threads...
.\examples\SecureMessagingSystem.cpp(67): ...wait -1 secs...
_
```

تظهر هذه الشاشة في خلفية واجهة المخدّم، وذلك عندما يتم تشغيل نظام الزبون الخاص بنظام التراسل الآمن بنجاح.



Student ID	Student Grade	Student Result	Getting Message Date and Time
900000	437	ن	
900001	577	ن	
900002	471	ن	
900003	381	ن	
900004	548	ن	
900005	387	ن	
900006	568	ن	
900007	639	ن	
900008	464	ن	
900009	449	ن	
900010	474	ن	
900011	597	ن	
900012	443	ن	
900013	434	ن	
900014	625	ن	
900015	412	ن	
900016	451	ن	
900017	405	ر	
900018	473	ن	
900019	620	ن	
900020	561	ن	
900021	469	ن	

تظهر هذه الشاشة مباشرة بعد تشغيل نظام المخدّم الخاص بنظام التراسل الآمن، وهي تعمل كواجهة لجداول قاعدة بيانات نظام المخدّم. تقوم هذه الواجهة بإظهار تقرير تفصيلي لمحتوى جدول نتائج الطلاب الذي يتم الإستفسار عنه من قبل الزبائن (Clients).

الملحق (ب)
الملفات المستخدمة في النظام

هنالك مجموعة من الملفات التي تم إستخدامها في النظام وهي مقسّمة كما يلي:

1. **الملفات المساعدة:** وهي تتكون من ملفين؛ الأول خاص بالمستخدمين (users.txt) حيث يحوي اسم المستخدم وكلمة المرور الخاصة به على التوالي، بينما الملف الثاني يحوي مفتاح التشفير السرّي (key.txt).

```
Users.txt
smssys,smspsw

Key.txt
1234567890123456
```

2. **ملفات جداول نظام التراسل:** تمت إضافة ثلاثة جداول خاصة بنظام التراسل وهي كما يلي:

RequestsMessagesQueue	
اسم الحقل	نوع البيانات
ID	Int
StdID	nvarchar(20)
Posted	Tinyint
PostedMessageDateTime	Datetime
SentMessageDateTime	datetime

جدول رسائل الطلبات – يتبع لقاعدة بيانات نظام الرسائل القصيرة (الزبون)

StudentResults	
اسم الحقل	نوع البيانات
ID	Int
StudentID	nvarchar(20)
StudentGrade	int
StudentResult	nvarchar(5)
ReceivedMessageDateTime	datetime

جدول رسائل الرد على الطلبات – يتبع لقاعدة بيانات نظام الرسائل القصيرة (الزبون)

StudentResults	
اسم الحقل	**نوع البيانات**
ID	int
StdID	nvarchar(20)
StdGrade	int
StdResult	nvarchar(5)
GetingDateTime	datetime

جدول نتائج الطلاب – يتبع لقاعدة بيانات نظام النتيجة (المخدّم)

3. **ملفات الأحداث:** وهي التي تقوم بتسجيل حركة الرسائل من جهتي الزبون والمخدّم، حيث كل جهة تحوي ملفين لتسجيل الأحداث وذلك حسب السيناريو المقدّم في كلٍ من الشكلين 5.5 و6.5

الجدول التالي يوضح مختصر عن تلك الملفات.

اسم الملف	وصف الملف
SMSClient.log	يقوم بتسجيل حركة الرسائل الصادرة من نظام الزبون وذلك قبل إرسالها إلى مصفوف الرسائل.
SMSClientRe.log	يقوم بتسجيل حركة الرسائل الواردة إلى نظام الزبون وذلك بعد إستلامها من قناة التراسل الآمنة.
SMSServer.log	يقوم بتسجيل حركة الرسائل الصادرة من نظام المخدّم وذلك قبل إرسالها قناة التراسل الآمنة.
SMSServerRe.log	يقوم بتسجيل حركة الرسائل الواردة إلى نظام المخدّم وذلك بعد إستلامها من قناة التراسل الآمنة.

وفيما يلي عرض لجزء يسير من محتوى تلك الملفات عبر الجدولين الآتيين.

نظام الزبون - ملفات الأحداث الخاصة بالزبون									
SMSClientRe.log					SMSClient.log				
زمن إستقبال رسالة الرد على الطلب	طول الرسالة بعد فك التشفير	طول الرسالة المضغوطة	طول الرسالة الأصلية المهيكلة	رقم الرسالة الوحيد	زمن إرسال رسالة الطلب	طول الرسالة بعد التشفير	طول الرسالة المضغوطة	طول الرسالة الأصلية المهيكلة	رقم الرسالة الوحيد
في عدم وجود المشفّر									
180720	71	71	81	9	180353	43	43	44	9
649172	74	74	83	477	648749	46	46	46	477
690223	74	74	83	518	689808	46	46	46	518
1052619	73	73	83	880	1052343	45	45	46	880
1073120	71	71	81	900	1072364	44	44	46	900
1073677	73	73	83	901	1073317	44	44	46	901
1094602	74	74	83	922	1094311	45	45	46	922
في وجود المشفّر (طول مفتاح التشفير السرّي = 128 بت)									
724452	80	73	84	1009	724038	48	44	47	1009
1192423	80	75	84	1477	1192019	48	46	47	1477
33583	80	74	84	1518	33141	48	46	47	1518
395300	80	74	84	1880	394599	48	45	47	1880
415293	80	71	82	1900	414615	48	44	47	1900
416300	80	73	84	1901	415564	48	44	47	1901
438365	80	75	84	1922	437486	48	45	47	1922
في وجود المشفّر (طول مفتاح التشفير السرّي = 192 بت)									
78351	80	73	84	2009	77507	48	44	47	2009
546488	80	75	84	2477	545728	48	46	47	2477
587468	80	75	84	2518	586733	48	46	47	2518
949574	80	74	84	2880	948457	48	45	47	2880
969523	80	71	82	2900	968445	48	44	47	2900
970542	80	73	84	2901	969383	48	44	47	2901
991506	80	74	84	2922	990355	48	45	47	2922
في وجود المشفّر (طول مفتاح التشفير السرّي = 256 بت)									
277400	80	73	84	3009	276535	48	44	47	3009
13536	80	75	84	3477	12738	48	46	47	3477
387182	80	75	84	3518	386175	48	46	47	3518
1111366	80	74	84	3880	1110171	48	45	47	3880
1151298	80	71	82	3900	1150214	48	44	47	3900
1153386	80	73	84	3901	1152185	48	44	47	3901
1195414	80	75	84	3922	1194191	48	45	47	3922

							نظام المخدّم - ملفات الأحداث الخاصة بالمخدّم
SMSServer.log				SMSServerRe.log			
رقم الرسالة الوحيد	طول الرسالة الأصلية المهيكلة	طول الرسالة المضغوطة	طول الرسالة بعد التشفير	رقم الرسالة الوحيد	طول الرسالة الأصلية المهيكلة	طول الرسالة المضغوطة	طول الرسالة بعد فك التشفير
							في عدم وجود المشفّر
9	44	43	43	9	81	71	71
477	46	46	46	477	83	74	74
518	46	46	46	518	83	74	74
880	46	45	45	880	83	73	73
900	46	44	44	900	81	71	71
901	46	44	44	901	83	73	73
922	46	45	45	922	83	74	74
							في وجود المشفّر (طول مفتاح التشفير السرّي = 128 بت)
1009	47	44	48	1009	84	73	80
1477	47	46	48	1477	84	75	80
1518	47	46	48	1518	84	74	80
1880	47	45	48	1880	84	74	80
1900	47	44	48	1900	82	71	80
1901	47	44	48	1901	84	73	80
1922	47	45	48	1922	84	75	80
							في وجود المشفّر (طول مفتاح التشفير السرّي = 192 بت)
2009	47	44	48	2009	84	73	80
2477	47	46	48	2477	84	75	80
2518	47	46	48	2518	84	75	80
2880	47	45	48	2880	84	74	80
2900	47	44	48	2900	82	71	80
2901	47	44	48	2901	84	73	80
2922	47	45	48	2922	84	74	80
							في وجود المشفّر (طول مفتاح التشفير السرّي = 256 بت)
3009	47	44	48	3009	84	73	80
3477	47	46	48	3477	84	75	80
3518	47	46	48	3518	84	75	80
3880	47	45	48	3880	84	74	80
3900	47	44	48	3900	82	71	80
3901	47	44	48	3901	84	73	80
3922	47	45	48	3922	84	75	80

الملحق (ج)
ملفات شيفرة النظام

بالإضافة إلى الملفات والمكتبات المستخدمة ضمن نظام مصفوف الرسائل مفتوح المصدر، تم إنجاز الآتي:

1. تعديل شيفرة نموذج التخزين والتمرير لتتوائم مع نظام التراسل الآمن وذلك ضمن الملف SecureMessagingSystem.cpp
2. إضافة ملف الإعلان (Header File) والملف التنفيذي (Implementation File) لخوارزمية Rijndael القياسية.
3. إنشاء ملف إعلان وملف تنفيذي يقومان بتقديم كل الدوال والإجراءات التي يحتاج إليها نظام التراسل الآمن وذلك ضمن الملفين SMS.h و SMS.cpp

SecureMessagingSystem.cpp

```cpp
///////////////////////////////////////////////////////////////////////////
// MQ4CPP - Message queuing for C++
// Copyright (C) 2004-2007  Riccardo Pompeo (Italy)
//
// This library is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 2.1 of the License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA


#include "StoreForward.h"
#include "Logger.h"
#include "SMS.h"

class MyClient : public SMSClient
{
private:
    unsigned itsCnt;

public:
    MyClient(const char* theName, const char* theWorkingPath, const char* theHost, short thePort,const char* theRemoteService)
        :SMSClient(theName,theWorkingPath,theHost,thePort,theRemoteService)
    {
        itsCnt=0;
        SCHEDULE(this,100);
    };

    virtual ~MyClient() {};

protected:
    virtual void onWakeup(Wakeup* theMessage)
    {
        GetNewMessageFromDB();
    };
};

class MyServer : public SMSServer
{
public:
    MyServer(const char* theName) : SMSServer(theName)
    {
    };

    virtual ~MyServer()
    {
    };

protected:
    string service(string theBuffer)
    {
        string ReplyMessage = GetMessageFromSecureChannel(theBuffer);
        return ReplyMessage;
    };
};

void main_sleep(int val)
{
    DISPLAY("...wait " << val << " secs...")
    Thread::sleep(val*1000);
}

bool CheckUserAuthorization(char *UserName, char *Password)
{
    FILE * pFile;
    char buffer[100]="";
    string LoginInfo, LogUserName, LogPassword, Uname, Psw;
    int i, pos;
    pFile = fopen ("users.txt","r");
    LoginInfo.erase();
    LogUserName.erase();
    LogPassword.erase();
    if (pFile!=NULL)
    {
        fgets (buffer, sizeof(buffer), pFile);
        for (i = 0; i < sizeof(buffer); i++)
            if (buffer[i] != NULL)
                LoginInfo.append(1, buffer[i]);
        pos = LoginInfo.find(',');
        LogUserName.append(LoginInfo, 0, pos);
        LogPassword.append(LoginInfo, pos + 1, LoginInfo.length());
        fclose (pFile);
    }
    Uname.erase();
    Psw.erase();
    Uname.assign(UserName);
    Psw.assign(Password);
    if ((LogUserName.compare(Uname) == 0) && (LogPassword.compare(Psw) == 0))
        return true;
    else
        return false;
}

int main(int argv,char* argc[])
{
    DISPLAY("MQ4CPP SecureMessagingSystem.cpp")
    DISPLAY("This example shows how to implement store&forward as a Secure Messaging System")

    bool client=false;
    char* host=NULL;
    int hport=0;

    if(argv < 5)
    {
        DISPLAY("Client usage: SecureMessagingSystem -c hostip port username password")
        DISPLAY("Server usage: SecureMessagingSystem -s port username password")
        return 0;
    }
    else if(string(argc[1]).compare("-c")==0 && argv==6)
    {
        if (CheckUserAuthorization(argc[4], argc[5]))
        {
            client=true;
            DISPLAY("Default host name=" << argc[2])
            host=argc[2];
            DISPLAY("Default host port=" << argc[3])
            hport=atoi(argc[3]);
        }
        else
        {
            TRACE("This user has no sufficient access permissions, please check your username or password and try again.");
            return 0;
        }
    }
    else if(string(argc[1]).compare("-s")==0 && argv==5)
    {
        if (CheckUserAuthorization(argc[3], argc[4]))
        {
            client=false;
            DISPLAY("Server port=" << argc[2])
            hport=atoi(argc[2]);
        }
        else
        {
            TRACE("This user has no sufficient access permissions, please check your username or password and try again.");
            return 0;
        }
    }
    else
    {
        DISPLAY("Client usage: SecureMessagingSystem -c hostip port username password")
        DISPLAY("Server usage: SecureMessagingSystem -s port username password")
        return 0;
    }

    try
    {
        if(client==true)
        {
            DISPLAY("Starting client threads...")
            STARTLOGGER("client.log")
            LOG("!!!!!!! SecureMessagingSystem.cpp - client !!!!!!!")
            MessageForwarder* aForwarder=new MessageForwarder("MyForwarder","tlog");
            MyClient* aStorer=new MyClient("MyClient" , "tlog", host, hport, "MyServer");
            main_sleep(-1);
        }
        else
        {
            DISPLAY("Starting server threads...")
            STARTLOGGER("server.log")
            LOG("!!!!!!! SecureMessagingSystem.cpp - server !!!!!!!")
            MessageProxyFactory aFactory("MyFactory",hport);
            MyServer* aServer=new MyServer("MyServer");
            main_sleep(-1);
        }

        DISPLAY("...stopping threads...")
        Thread::shutdownInProgress();
        STOPLOGGER()
        STOPREGISTRY()
        STOPTIMER()
    }
    catch(Exception& ex)
    {
        TRACE(ex.getMessage().c_str())
    }
    catch(...)
    {
        TRACE("Unhandled exception")
    }

    DISPLAY("...done!")
    if(client)
        DISPLAY("See client.log for details")
    else
        DISPLAY("See server.log for details")
    return 0;
}
```

SMS.h
==

```cpp
#ifndef _SMS_H_
#define _SMS_H_

#include "StoreForward.h"
#include "Logger.h"
#include <sql.h>

class SMS
{
public:
    SMS();
    ~SMS();

protected:
    int print_error (SQLHENV    henv,
                     SQLHDBC    hdbc,
                     SQLHSTMT   hstmt);
    int check_error (SQLHENV    henv,
                     SQLHDBC    hdbc,
                     SQLHSTMT   hstmt,
                     SQLRETURN  frc);
    void TerminateDBConnection(SQLHENV hEnv, SQLHDBC hDBC, SQLHSTMT hStmt);
    DWORD TimeMilliSeconds(LARGE_INTEGER theTime, LARGE_INTEGER theFrequencies);
    void WriteToLogFile(char* FileName, string MessageID, int OriginalMsgLen, int CompressedMsgLen,
                         int EncryptedMsgLen, DWORD ProcessTime);
    string GetEncDecKey();
    string ConvertToStr(UCHAR Str[], int StrLength);
    void ConvertToHex(int Dec, string &Hex);
    void ConvertToDec(char* Hex, int &Dec);
    string EncryptMessage(string Message);
    string DecryptMessage(string Message);

protected:
    LARGE_INTEGER Frequencies;
    LARGE_INTEGER Process_StartTime;
    LARGE_INTEGER Process_EndTime;
    string MessagID;
    int OriginalMsgLen;
    int CompressedMsgLen;
    int EncryptedMsgLen;
    DWORD ProcessTime;
};

class SMSClient : protected MessageStorer, SMS
{
public:
    SMSClient(const char* theName, const char* theWorkingPath, const char* theHost, short thePort,const char* theRemoteService);
    ~SMSClient();
    void GetNewMessageFromDB();

protected:
    string generateServerBXMLMSG(string studentID, string MessageID);
};

class SMSServerReply : protected Observer, SMS
{
public:
    SMSServerReply(const char* theName);
    ~SMSServerReply();
    void ProcessFeedbackMessage(string theBuffer);

protected:
    void reverseServerBXMLMSG(string theBuffer, string StdInfo[]);
};

class SMSServer : protected Server, SMS
{
public:
    SMSServer(const char* theName);
    ~SMSServer();
    string GetMessageFromSecureChannel(string theBuffer);

protected:
    string generateClientBXMLMSG(string studentID, string studentGrade, char
    studentResult, string MessageID);
    void reverseClientBXMLMSG(string theBuffer, string StdInfo[]);
};
#endif // _SMS_H_

```

SMS.cpp

```cpp
#include "StoreForward.h"
#include "Logger.h"
#include "Compression.h"
#include "examples/rijndael.cpp"
#include "Session.h"
#include "sms.h"
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <stdlib.h>
#include <memory.h>
#include <sql.h>
#include <sqlext.h>
#include <time.h>
using namespace std;

SMS::SMS()
{
}

SMS::~SMS()
{
}

/*******************************************************************
**   - print_error   - call SQLError(), display SQLSTATE and message
*******************************************************************/
int SMS::print_error(SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt)
{
    SQLCHAR     buffer[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR     sqlstate[SQL_SQLSTATE_SIZE + 1];
    SQLINTEGER  sqlcode;
    SQLSMALLINT length;
    while ( SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
                     SQL_MAX_MESSAGE_LENGTH + 1, &length) == SQL_SUCCESS )
    {
        TRACE("**** ERROR *****");
        TRACE("         SQLSTATE: "<<sqlstate);
        TRACE("Native Error Code: "<<sqlcode);
        TRACE(buffer);
    };
    return ( SQL_ERROR);
} /* end print_error */

/*******************************************************************
**   - check_error   - call print_error(), checks severity of return code
*******************************************************************/
int SMS::check_error(SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc)
{
    SQLRETURN   rc;
    print_error(henv, hdbc, hstmt);
    switch (frc){
    case SQL_SUCCESS : break;
    case SQL_ERROR :
    case SQL_INVALID_HANDLE:
        TRACE(" ** FATAL ERROR, Attempting to rollback transaction **");
        rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
        if (rc != SQL_SUCCESS)
            TRACE("Rollback Failed, Exiting application")
        else
            TRACE("Rollback Successful, Exiting application");
        exit(frc);
        break;
    case SQL_SUCCESS_WITH_INFO :
        TRACE(" ** Warning Message, application continuing");
        break;
    case SQL_NO_DATA_FOUND :
        TRACE(" ** No Data Found ** ");
        break;
    default :
        TRACE(" ** Invalid Return Code ** ");
        TRACE(" ** Attempting to rollback transaction **");
        SQLTransact(henv, hdbc, SQL_ROLLBACK);
        exit(frc);
        break;
    }
    return(SQL_SUCCESS);

} /* end check_error */

/*******************************************************************
**  - TerminateDBConnection   - call TerminateDBConnection(), terminate the database connection
*******************************************************************/
void SMS::TerminateDBConnection(SQLHENV hEnv, SQLHDBC hDBC, SQLHSTMT hStmt)
{
    RETCODE retcode;
    // Free the allocated statement handle
    retcode = SQLFreeStmt (hStmt, SQL_DROP);
    if (retcode != SQL_SUCCESS )
        check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
    // Disconnect from datasource
    retcode = SQLDisconnect (hDBC);
    if (retcode != SQL_SUCCESS )
        print_error(hEnv, hDBC, SQL_NULL_HSTMT);
    // Free the allocated connection handle
    retcode = SQLFreeConnect (hDBC);
    if (retcode != SQL_SUCCESS )
        print_error(hEnv, hDBC, SQL_NULL_HSTMT);
    // Free the allocated ODBC environment handle
    retcode = SQLFreeEnv (hEnv);
    if (retcode != SQL_SUCCESS )
        print_error(hEnv, hDBC, SQL_NULL_HSTMT);
}

DWORD SMS::TimeMilliSeconds(LARGE_INTEGER theTime,
                            LARGE_INTEGER theFrequencies)
{
    DWORD ClockTime    = theTime.u.LowPart;
    DWORD freq         = theFrequencies.u.LowPart;
    DWORD MilliSeconds = ((double)ClockTime/(double)freq) * 1000;
    return MilliSeconds;
}

void SMS::WriteToLogFile(char* FileName, string MessageID, int OriginalMsgLen_, int CompressedMsgLen_,
                         int EncryptedMsgLen_, DWORD ProcessTime_)
{
    FILE * pFile;
    char buffer[65]="", buffer1[65]="", buffer2[65]="";
    pFile = fopen (FileName,"a+t");
    if (pFile!=NULL)
    {
        string TraceLine;
        TraceLine.append(MessageID, 0, MessageID.length());
        TraceLine += ", ";
        TraceLine.append(itoa (OriginalMsgLen_, buffer, 10), strlen(itoa (OriginalMsgLen_, buffer, 10)));
        TraceLine += ", ";
        TraceLine.append(itoa (CompressedMsgLen_, buffer1, 10), strlen(itoa (CompressedMsgLen_, buffer1, 10)));
        TraceLine += ", ";
        TraceLine.append(itoa (EncryptedMsgLen_, buffer2, 10), strlen(itoa (EncryptedMsgLen_, buffer2, 10)));
        if (ProcessTime_ != NULL)
        {
            TraceLine += ", ";
            char ProcessTimeStr[65];
            _i64toa(ProcessTime_, ProcessTimeStr, 10);
            TraceLine.append(ProcessTimeStr, strlen(ProcessTimeStr));
        }
        TraceLine += "\n";
        fputs (TraceLine.c_str(), pFile);
```

```cpp
        fclose (pFile);
    }
}

string SMS::GetEncDecKey()
{
    FILE * pFile;
    char buffer[70]="";
    string theKey;
    pFile = fopen ("key.txt","r");
    if (pFile!=NULL)
    {
        fgets (buffer, sizeof(buffer), pFile);
        theKey.erase();
        for (int i=0; i<sizeof(buffer); i++)
            if (buffer[i] != NULL)
                theKey.append(1, buffer[i]);
        fclose (pFile);
    }
    return theKey;
}

string SMS::ConvertToStr(UCHAR Str[], int StrLength)
{
    int i;
    string StrResult;
    StrResult.erase();
    for (i = 0; i < StrLength; i++)
    {
        if (Str[i] != NULL)
            StrResult.append(1, Str[i]);
    }
    return StrResult;
}

void SMS::ConvertToHex(int Dec, string &Hex)
{
  char Hexa[10];
  switch(Dec)
  {
   case 10:
       Hex="A";
       break;
   case 11:
       Hex="B";
       break;
   case 12:
       Hex="C";
       break;
   case 13:
       Hex="D";
       break;
   case 14:
       Hex="E";
       break;
   case 15:
       Hex="F";
       break;
   default :
       Hex=itoa(Dec, Hexa, 10);
  }
}

void SMS::ConvertToDec(char* Hex, int &Dec)
{
      switch(Hex[0])
  {
   case 'A':
       Dec=10;
       break;
   case 'B':
       Dec=11;
       break;
   case 'C':
       Dec=12;
       break;
   case 'D':
       Dec=13;
       break;
   case 'E':
       Dec=14;
       break;
   case 'F':
       Dec=15;
       break;
   default :
       Dec = atoi(Hex);
  }
}

string SMS::EncryptMessage(string Message)
{
        string msg="";
        string blockMsg="";
        string Hex;
        unsigned char cipherMsg[16]="";
        unsigned char plainMsg[16]="";
        Rijndael rijnd;
        char ch;
        int counter=0, i=0, j=0, k=0, l=0, m=0, x=0;
        // Get the encryption/decryption key
        //=======================================================================================
        string KeyStr = GetEncDecKey();
        unsigned char *key = new (unsigned char[KeyStr.length()+1]);

        for (i=0; i<(int)KeyStr.length(); i++)
            key[i] = KeyStr.at(i);
        int KeySize = KeyStr.length();
        if (KeyStr.length() == 16)
            x = rijnd.init(Rijndael::CBC, Rijndael::Encrypt,  key,Rijndael::Key16Bytes, 0);
        else
            if (KeyStr.length() == 24)
                x = rijnd.init(Rijndael::CBC, Rijndael::Encrypt,  key,Rijndael::Key24Bytes, 0);
            else
                x = rijnd.init(Rijndael::CBC, Rijndael::Encrypt,  key,Rijndael::Key32Bytes, 0);
        //=======================================================================================
        int lastBlockSize = (Message.length() % 16);
        ConvertToHex(lastBlockSize, Hex);
        msg.erase();
        memset(plainMsg, '\0', 16);
        memset(cipherMsg, '\0', 16);
        blockMsg.erase();
        msg.append(1, Hex.at(0));
        for (k=0; k<(int)Message.length();k++)
        {
            msg.append(1, Message.at(k));
        }
        if (msg.length()<16)
        {
            ch=' ';
            for (k=0; k<(int)msg.length();k++)
            {
                plainMsg[k]=msg.at(k);
            }
            for (k=msg.length(); k<16;k++)
            {
                plainMsg[k]= ch;
            }
            l = rijnd.blockEncrypt(plainMsg, KeySize*8, cipherMsg);
            for (m=0; m<16; m++)
            {
                blockMsg.append(1, cipherMsg[m]);
            }
```

```cpp
        }
        else
        for (i=0; i<(int)msg.length(); i++)
        {
            counter++;
            plainMsg[j]=msg.at(i);
            j++;
            if (j == 16)
            {
                l = rijnd.blockEncrypt(plainMsg, 16*8, cipherMsg);
                for (m=0; m<16; m++)
                {
                    blockMsg.append(1, cipherMsg[m]);
                }
                j=0;
            }
            if (counter == msg.length() && j != 0)
            {
                ch=' ';
                for (k=j; k<16;k++)
                {
                    plainMsg[k]= ch;
                }
                l = rijnd.blockEncrypt(plainMsg, (sizeof(key)-1)*8, cipherMsg);
                for ( m=0; m<16; m++)
                {
                    blockMsg.append(1, cipherMsg[m]);
                }
            }
        }
        return blockMsg;
}

string SMS::DecryptMessage(string Message)
{
    string blockMsg="";
    unsigned char cipherMsg[16];
    unsigned char plainMsg[16];
    Rijndael rijnd;
    int x;
    // Get the encryption/decryption key
    //======================================================================================
    string KeyStr = GetEncDecKey();
    unsigned char *key = new (unsigned char[KeyStr.length()+1]);
    for (int i=0; i<(int)KeyStr.length(); i++)
        key[i] = KeyStr.at(i);
    int KeySize = KeyStr.length();
    if (KeyStr.length() == 16)
        x = rijnd.init(Rijndael::CBC, Rijndael::Decrypt,  key,Rijndael::Key16Bytes, 0);
    else
        if (KeyStr.length() == 24)
            x = rijnd.init(Rijndael::CBC, Rijndael::Decrypt,  key,Rijndael::Key24Bytes, 0);
        else
            x = rijnd.init(Rijndael::CBC, Rijndael::Decrypt,  key,Rijndael::Key32Bytes, 0);
    //======================================================================================
    memset(plainMsg, '\0', 16);
    memset(cipherMsg, '\0', 16);
    int counter=0, i=0, j=0, k=0, l=0, m=0;
    int firstBlock=0, lastBlockSize=0, start=0;
    char Hex[1]="";
    blockMsg.erase();
    for (i=0; i<(int)Message.length(); i++)
    {
        counter++;
        cipherMsg[j]=Message.at(i);
        j++;
        if (j == 16)
        {
            l = rijnd.blockDecrypt(cipherMsg, KeySize*8, plainMsg);
            if (firstBlock == 0)
            {
                start = 1;
                firstBlock = 1;
                Hex[0] = plainMsg[0];
                ConvertToDec(Hex, lastBlockSize);
            }
            else
            {
                start = 0;
            }
            if (counter == Message.length())
            {
                for ( m=start; m<=lastBlockSize; m++)
                {
                    blockMsg.append(1, plainMsg[m]);
                }
            }
            else
            {
                for ( m=start; m<16; m++)
                {
                    blockMsg.append(1, plainMsg[m]);
                }
            }
            j=0;
        }
    }
    return blockMsg;
}

SMSClient::SMSClient(const char* theName, const char* theWorkingPath, const char* theHost, short thePort,const char* theRemoteService)
    :MessageStorer(theName, theWorkingPath, theHost, thePort, theRemoteService)
{
}

SMSClient::~SMSClient()
{
}

void SMSClient::GetNewMessageFromDB()
{
    QueryPerformanceFrequency(&Frequencies);
    QueryPerformanceCounter(&Process_StartTime);
    bool foundNewRec = false;
    char* aString;
    string Message, StudentID, MessageID;
    SQLHENV hEnv = NULL; // Env Handle from SQLAllocEnv()
    SQLHDBC hDBC = NULL; // Connection handle
    SQLHSTMT hStmt = NULL;// Statement handle
    RETCODE retcode;
    UCHAR ID[5]="";// Model buffer
    UCHAR StdID[10]="";// Model buffer
    // Get the new incomming message from database
    //========================================================================
    UCHAR szDSN[SQL_MAX_DSN_LENGTH] = "SMS_DSN";// Data Source Name buffer
    UCHAR szUID[13] = "";// User ID buffer
    UCHAR szPasswd[7] = "";// Password buffer
    SDWORD cbModel, cbModel1;// Model buffer bytes recieved
    UCHAR szSqlStr[128]= "SELECT * FROM RequestsMessagesQueue where Posted=0";
    // Initialize connection for selection
    //========================================================================
    // Allocate memory for ODBC Environment handle
    retcode = SQLAllocEnv (&hEnv);
    if (retcode != SQL_SUCCESS )
        check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
    // Allocate memory for the connection handle
    retcode = SQLAllocConnect (hEnv, &hDBC);
    if (retcode != SQL_SUCCESS )
        check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
    // Connect to the data source "szDSN" using userid and password.
    retcode = SQLConnect (hDBC, szDSN, SQL_NTS, szUID, SQL_NTS, szPasswd,SQL_NTS);
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
    {
        // Allocate memory for the statement handle
        retcode = SQLAllocStmt (hDBC, &hStmt);
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
        // Prepare the SQL statement by assigning it to the statement handle
        retcode = SQLPrepare (hStmt, szSqlStr, sizeof (szSqlStr));
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, hStmt, retcode);
        // Execute the SQL statement handle
        retcode = SQLExecDirect(hStmt, szSqlStr, SQL_NTS);
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, hStmt, retcode);
        // Project columns
        retcode = SQLBindCol (hStmt, 1, SQL_C_CHAR, ID, sizeof(ID),&cbModel);
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, hStmt, retcode);
        retcode = SQLBindCol (hStmt, 2, SQL_C_CHAR, StdID, sizeof(StdID),&cbModel1);
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, hStmt, retcode);
    }
    while ((retcode = SQLFetch(hStmt)) == SQL_SUCCESS)
    {
        ProcessTime = TimeMilliSeconds(Process_StartTime, Frequencies);
        foundNewRec = true;
        // Convert the message into binaryXML format (Translator Filter)
        StudentID    = ConvertToStr(StdID, sizeof(StdID));
        MessageID = ConvertToStr(ID, sizeof(ID));
        Message = generateServerBXMLMSG(StudentID, MessageID);
        OriginalMsgLen = Message.length();
        // Compress the binaryXML message (Compression Filter)
        PacketCompression compression(true);
        Message = compression.deflate(Message);
        CompressedMsgLen = Message.length();
        // Encrypt the message using Rijndael algorithm (Encryption Filter)
        Message = EncryptMessage(Message);
        EncryptedMsgLen = Message.length();
        // Send message via messaging channel (Message Multiplexer Filter)
        send(Message);
        // Update the selected recodes
        SQLHENV hEnvUpdt = NULL; // Env Handle from SQLAllocEnv()
        SQLHDBC hDBCUpdt = NULL; // Connection handle
        SQLHSTMT hStmtUpdt = NULL; // Statement handle
        RETCODE retcd;
        UCHAR szSqlStrUpdate[128]="";
        string SqlStr = "Update RequestsMessagesQueue Set Posted=2 Where ID = ";
        SqlStr.append(MessageID, 0, MessageID.length());
        int i;
        for (i=0; i<(int)SqlStr.length(); i++)
            szSqlStrUpdate[i]=SqlStr.at(i);
        szSqlStrUpdate[i]=NULL;
        // Initialize connection for update
        //==========================================================================
        // Allocate memory for ODBC Environment handle
        retcd = SQLAllocEnv (&hEnvUpdt);
        if (retcd != SQL_SUCCESS )
            check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
        // Allocate memory for the connection handle
        retcd = SQLAllocConnect (hEnvUpdt, &hDBCUpdt);
        if (retcd != SQL_SUCCESS )
            check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
        // Connect to the data source "szDSN" using userid and password.
        retcd = SQLConnect (hDBCUpdt, szDSN, SQL_NTS, szUID, SQL_NTS, szPasswd,SQL_NTS);
        if (retcd == SQL_SUCCESS || retcd == SQL_SUCCESS_WITH_INFO)
        {
            // Allocate memory for the statement handle
            retcd = SQLAllocStmt (hDBCUpdt, &hStmtUpdt);
            if (retcd != SQL_SUCCESS )
                check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
            // Prepare the SQL statement by assigning it to the statement handle
            retcd = SQLPrepare (hStmtUpdt, szSqlStrUpdate, sizeof (szSqlStrUpdate));
            if (retcd != SQL_SUCCESS )
                check_error(hEnvUpdt, hDBCUpdt, hStmtUpdt, retcd);
            // Execute the SQL statement handle
            retcd = SQLExecDirect(hStmtUpdt, szSqlStrUpdate, SQL_NTS);
            if (retcd != SQL_SUCCESS )
                check_error(hEnvUpdt, hDBCUpdt, hStmtUpdt, retcd);
            TerminateDBConnection(hEnvUpdt, hDBCUpdt, hStmtUpdt);
        }
        for (i=0; i<sizeof(StdID); i++)
            ID[i]=NULL;
        for (i=0; i<sizeof(StdID); i++)
            StdID[i]=NULL;
    }
    if (retcode != SQL_NO_DATA_FOUND)
        check_error(hEnv, hDBC, hStmt, retcode);
    // Terminate the database connection
    TerminateDBConnection(hEnv, hDBC, hStmt);
    if  (foundNewRec)
    // Trace process
        WriteToLogFile("SMSClient.log", MessageID, OriginalMsgLen, CompressedMsgLen,    ↙
    EncryptedMsgLen,
                        ProcessTime);
}

string SMSClient::generateServerBXMLMSG(string studentID, string MessageID)
{
    ListProperty itsTxStructure;

    TRACE("Populating TX structure")
    itsTxStructure.free();

    StringProperty* aStringProperty_studentID=new StringProperty("studentID");
    aStringProperty_studentID->set(studentID);
    itsTxStructure.add(aStringProperty_studentID);

    StringProperty* aStringProperty_MessageID=new StringProperty("MessageID");
    aStringProperty_MessageID->set(MessageID);
    itsTxStructure.add(aStringProperty_MessageID);

    string aString;
    encodeProperties(itsTxStructure,aString);
    return aString;
}


SMSServerReply::SMSServerReply(const char* theName):Observer(theName)
{
}

SMSServerReply::~SMSServerReply()
{
}

void SMSServerReply::ProcessFeedbackMessage(string theBuffer)
{
    //============================================================================
    // Insert the received student's result into StudentResults table.
    //============================================================================
    string Message, StdGrade, StdResult;
    string StdInfo[4];
    // Get message from the messaging channel
    Message.append(theBuffer, 0, theBuffer.length());
    EncryptedMsgLen = Message.length();
    // Decrypt the received message
    Message = DecryptMessage(Message);
    CompressedMsgLen = Message.length();
    // Decompress the message
    PacketCompression compression(true);
    Message = compression.inflate(Message);
    OriginalMsgLen = Message.length();
    reverseServerBXMLMSG(Message, StdInfo);
    //============================================================================
    SQLHENV hEnv = NULL; // Env Handle from SQLAllocEnv()
    SQLHDBC hDBC = NULL; // Connection handle
    SQLHSTMT hStmt = NULL;// Statement handle
    RETCODE retcode;
    int i;
    UCHAR szDSN[SQL_MAX_DSN_LENGTH] = "SMS_DSN";// Data Source Name buffer
    UCHAR szUID[13] = "";// User ID buffer
    UCHAR szPasswd[7] = "";// Password buffer
    UCHAR szSqlStr[128];
    string SqlStr = "Insert INTO StudentResults (StudentID, StudentGrade, StudentResult) Values (";
    SqlStr.append(StdInfo[0], 0, StdInfo[0].length());
    SqlStr += ", ";
    SqlStr.append(StdInfo[1], 0, StdInfo[1].length());
    SqlStr += ", '";
    SqlStr.append(StdInfo[2], 0, StdInfo[2].length());
    SqlStr += "')";
    for (i=0; i<(int)SqlStr.length(); i++)
        szSqlStr[i]=SqlStr.at(i);
    szSqlStr[i]=NULL;
    // Allocate memory for ODBC Environment handle
    retcode = SQLAllocEnv (&hEnv);
    if (retcode != SQL_SUCCESS )
        check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
    // Allocate memory for the connection handle
    retcode = SQLAllocConnect (hEnv, &hDBC);
    if (retcode != SQL_SUCCESS )
        check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
    // Connect to the data source "szDSN" using userid and password.
    retcode = SQLConnect (hDBC, szDSN, SQL_NTS, szUID, SQL_NTS, szPasswd,SQL_NTS);
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
    {
        // Allocate memory for the statement handle
        retcode = SQLAllocStmt (hDBC, &hStmt);
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
        // Prepare the SQL statement by assigning it to the statement handle
        retcode = SQLPrepare (hStmt, szSqlStr, sizeof (szSqlStr));
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, hStmt, retcode);
        // Execute the SQL statement handle
        retcode = SQLExecDirect(hStmt, szSqlStr, SQL_NTS);
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, hStmt, retcode);
    }
    // Terminate the database connection
    TerminateDBConnection(hEnv, hDBC, hStmt);

    //==========================================================================
    // Update the RequestsMessagesQueue table.
    //==========================================================================
    SQLHENV hEnvUpdt = NULL; // Env Handle from SQLAllocEnv()
    SQLHDBC hDBCUpdt = NULL; // Connection handle
    SQLHSTMT hStmtUpdt = NULL;// Statement handle
    RETCODE retcd;
    UCHAR szSqlStrUpdate[128]="";
    string SqlStrUpdt = "Update RequestsMessagesQueue Set Posted=1, SentMessageDateTime=getdate() Where ID =";
    SqlStrUpdt.append(StdInfo[3], 0, StdInfo[3].length());
    for (i=0; i<(int)SqlStrUpdt.length(); i++)
        szSqlStrUpdate[i]=SqlStrUpdt.at(i);
    szSqlStrUpdate[i]=NULL;
    // Initialize connection for update
    //=========================================================================
    // Allocate memory for ODBC Environment handle
    retcd = SQLAllocEnv (&hEnvUpdt);
    if (retcd != SQL_SUCCESS )
        check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
    // Allocate memory for the connection handle
    retcd = SQLAllocConnect (hEnvUpdt, &hDBCUpdt);
    if (retcd != SQL_SUCCESS )
        check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
    // Connect to the data source "szDSN" using userid and password.
    retcd = SQLConnect (hDBCUpdt, szDSN, SQL_NTS, szUID, SQL_NTS, szPasswd,SQL_NTS);
    if (retcd == SQL_SUCCESS || retcd == SQL_SUCCESS_WITH_INFO)
    {
        // Allocate memory for the statement handle
        retcd = SQLAllocStmt (hDBCUpdt, &hStmtUpdt);
        if (retcd != SQL_SUCCESS )
            check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
        // Prepare the SQL statement by assigning it to the statement handle
        retcd = SQLPrepare (hStmtUpdt, szSqlStrUpdate, sizeof (szSqlStrUpdate));
        if (retcd != SQL_SUCCESS )
            check_error(hEnvUpdt, hDBCUpdt, hStmtUpdt, retcd);
        // Execute the SQL statement handle
        retcd = SQLExecDirect(hStmtUpdt, szSqlStrUpdate, SQL_NTS);
        if (retcd != SQL_SUCCESS )
            check_error(hEnvUpdt, hDBCUpdt, hStmtUpdt, retcd);
        TerminateDBConnection(hEnvUpdt, hDBCUpdt, hStmtUpdt);
        QueryPerformanceFrequency(&Frequencies);
        QueryPerformanceCounter(&Process_EndTime);
        ProcessTime = TimeMilliSeconds(Process_EndTime, Frequencies);
        // Trace process
        WriteToLogFile("SMSClientRe.log", StdInfo[3], OriginalMsgLen, CompressedMsgLen, ↙
     EncryptedMsgLen,
                        ProcessTime);
    }
}

void SMSServerReply::reverseServerBXMLMSG(string theBuffer, string StdInfo[])
{
    ListProperty itsRxStructure;

    decodeProperties(theBuffer,itsRxStructure);
    TRACE("Retrieving data from structure");

    Property* aProperty_studentID=itsRxStructure.get("studentID");
    if(aProperty_studentID!=NULL && aProperty_studentID->is(PROPERTY_STRING))
    {
        StdInfo[0]=((StringProperty*)aProperty_studentID)->get();
    }

    Property* aProperty_studentGrade=itsRxStructure.get("studentGrade");
    if(aProperty_studentGrade!=NULL && aProperty_studentGrade->is(PROPERTY_STRING))
    {
        StdInfo[1]=((StringProperty*)aProperty_studentGrade)->get();
    }

    Property* aProperty_studentResult=itsRxStructure.get("studentResult");
    if(aProperty_studentResult!=NULL && aProperty_studentResult->is(PROPERTY_CHAR))
    {
        StdInfo[2]=((CharProperty*)aProperty_studentResult)->get();
    }

    Property* aProperty_MessageID=itsRxStructure.get("MessageID");
    if(aProperty_MessageID!=NULL && aProperty_MessageID->is(PROPERTY_STRING))
    {
        StdInfo[3]=((StringProperty*)aProperty_MessageID)->get();
    }

    TRACE("Storing received structure")
    try
    {
        ofstream aStream("rxstruct.log");
        itsRxStructure.serialize(aStream);
        aStream.close();
    }
    catch(...)
    {
        TRACE("Exception during storing of file ")
    }
}

SMSServer:: SMSServer(const char* theName) : Server(theName)
{
}

SMSServer::~SMSServer()
{
}

string SMSServer::GetMessageFromSecureChannel(string theBuffer)
```

```
{
    QueryPerformanceFrequency(&Frequencies);
    ostrstream aStream;
    string Message, StudentID, StudentGrade, StudentResult, MessageID;
    string StdInfo[2];
    Message.erase();
    // Get message from the messaging channel
    Message.append(theBuffer, 0, theBuffer.length());
    EncryptedMsgLen = Message.length();
    // Decrypt the received message
    Message = DecryptMessage(Message);
    CompressedMsgLen = Message.length();
    // Decompress the message
    PacketCompression compression(true);
    Message = compression.inflate(Message);
    OriginalMsgLen = Message.length();
    // Trace process
    reverseClientBXMLMSG(Message, StdInfo);
    WriteToLogFile("SMSServer.log", StdInfo[1], OriginalMsgLen, CompressedMsgLen, EncryptedMsgLen,
                    NULL);
    //============================================================================
    SQLHENV hEnv = NULL; // Env Handle from SQLAllocEnv()
    SQLHDBC hDBC = NULL; // Connection handle
    SQLHSTMT hStmt = NULL;// Statement handle
    RETCODE retcode;
    UCHAR StdID[10]="";// Model buffer
    UCHAR StdGrade[4]="";// Model buffer
    UCHAR StdResult[2]="";// Model buffer
    // Get the new incomming message from database
    //===========================================================================
    UCHAR szDSN[SQL_MAX_DSN_LENGTH] = "STDRESULTS_DSN";// Data Source Name buffer
    UCHAR szUID[13] = "";// User ID buffer
    UCHAR szPasswd[7] = "";// Password buffer
    SDWORD cbModel, cbModel1, cbModel2;// Model buffer bytes recieved
    UCHAR szSqlStr[128]= "";
    string SqlStr = "SELECT * FROM StudentResults Where StdID = ";
    SqlStr.append(StdInfo[0], 0, StdInfo[0].length());
    int i;
    for (i=0; i<(int)SqlStr.length(); i++)
        szSqlStr[i]=SqlStr.at(i);
    szSqlStr[i]=NULL;
    // Initialize connection for selection
    //===========================================================================
    // Allocate memory for ODBC Environment handle
    retcode = SQLAllocEnv (&hEnv);
    if (retcode != SQL_SUCCESS )
        check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
    // Allocate memory for the connection handle
    retcode = SQLAllocConnect (hEnv, &hDBC);
    if (retcode != SQL_SUCCESS )
        check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
    // Connect to the data source "szDSN" using userid and password.
    retcode = SQLConnect (hDBC, szDSN, SQL_NTS, szUID, SQL_NTS, szPasswd,SQL_NTS);
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
    {
        // Allocate memory for the statement handle
        retcode = SQLAllocStmt (hDBC, &hStmt);
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
        // Prepare the SQL statement by assigning it to the statement handle
        retcode = SQLPrepare (hStmt, szSqlStr, sizeof (szSqlStr));
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, hStmt, retcode);
        // Execute the SQL statement handle
        retcode = SQLExecDirect(hStmt, szSqlStr, SQL_NTS);
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, hStmt, retcode);
        // Project columns
        retcode = SQLBindCol (hStmt, 2, SQL_C_CHAR, StdID, sizeof(StdID),&cbModel);
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, hStmt, retcode);
        retcode = SQLBindCol (hStmt, 3, SQL_C_CHAR, StdGrade, sizeof(StdGrade),&cbModel1);
        if (retcode != SQL_SUCCESS )
            check_error(hEnv, hDBC, hStmt, retcode);
        retcode = SQLBindCol (hStmt, 4, SQL_C_CHAR, StdResult, sizeof(StdResult),&cbModel2);
        if (retcode != SQL SUCCESS )
            check_error(hEnv, hDBC, hStmt, retcode);
    }
    bool ResultExist = false;
    StudentID.erase();
    StudentGrade.erase();
    StudentResult.erase();
    MessageID.erase();
    while ((retcode = SQLFetch(hStmt)) == SQL_SUCCESS)
    {
        ResultExist = true;
        StudentID     = ConvertToStr(StdID, sizeof(StdID));
        StudentGrade  = ConvertToStr(StdGrade, sizeof(StdGrade));
        StudentResult = ConvertToStr(StdResult, sizeof(StdResult));
        MessageID.append(StdInfo[1], 0, StdInfo[1].length());
    }
    // Terminate the database connection
    TerminateDBConnection(hEnv, hDBC, hStmt);
    // If no results exist, then return the ID with zeros
    if (ResultExist == false)
    {
        StudentID.append(StdInfo[0], 0, StdInfo[0].length());
        StudentGrade.append(1, '0');
        StudentResult.append(1, 'N');
        MessageID.append(StdInfo[1], 0, StdInfo[1].length());
    }
    // Convert the message into binaryXML format (Translator Filter)
    string ReplyMessage = generateClientBXMLMSG(StudentID, StudentGrade, StudentResult.at(0), MessageID);
    OriginalMsgLen = ReplyMessage.length();
    // Compress the binaryXML message (Compression Filter)
    ReplyMessage = compression.deflate(ReplyMessage);
    CompressedMsgLen = ReplyMessage.length();
    // Encrypt the message using Rijndael algorithm (Encryption Filter)
    ReplyMessage = EncryptMessage(ReplyMessage);
    EncryptedMsgLen = ReplyMessage.length();
    // Trace process
    WriteToLogFile("SMSServerRe.log", MessageID, OriginalMsgLen, CompressedMsgLen, EncryptedMsgLen,
                    NULL);

    return ReplyMessage;
}

string SMSServer::generateClientBXMLMSG(string studentID, string studentGrade, char studentResult, string MessageID)
{
    ListProperty itsTxStructure;

    TRACE("Populating TX structure")
    itsTxStructure.free();

    StringProperty* aStringProperty_studentID=new StringProperty("studentID");
    aStringProperty_studentID->set(studentID);
    itsTxStructure.add(aStringProperty_studentID);

    StringProperty* aStringProperty_studentGrade=new StringProperty("studentGrade");
    aStringProperty studentGrade->set(studentGrade);
    itsTxStructure.add(aStringProperty_studentGrade);

    CharProperty* aCharProperty_studentResult=new CharProperty("studentResult");
    aCharProperty_studentResult->set(studentResult);
    itsTxStructure.add(aCharProperty_studentResult);

    StringProperty* aStringProperty MessageID=new StringProperty("MessageID");
    aStringProperty_MessageID->set(MessageID);
    itsTxStructure.add(aStringProperty_MessageID);

    string aString;
    encodeProperties(itsTxStructure,aString);
    return aString;
}

void SMSServer::reverseClientBXMLMSG(string theBuffer, string StdInfo[])
{
    ListProperty itsRxStructure;

    decodeProperties(theBuffer,itsRxStructure);
    TRACE("Retrieving data from structure")

    Property* aProperty_studentID=itsRxStructure.get("studentID");
    if(aProperty_studentID!=NULL && aProperty_studentID->is(PROPERTY_STRING))
    {
        StdInfo[0]=((StringProperty*)aProperty_studentID)->get();
    }

    Property* aProperty_MessageID=itsRxStructure.get("MessageID");
    if(aProperty_MessageID!=NULL && aProperty_MessageID->is(PROPERTY_STRING))
    {
        StdInfo[1]=((StringProperty*)aProperty_MessageID)->get();
    }

    TRACE("Storing received structure")
    try
    {
        ofstream aStream("rxstruct.log");
        itsRxStructure.serialize(aStream);
        aStream.close();
    }
    catch(...)
    {
        TRACE("Exception during storing of file ")
    }
}


```