

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 المخدم في النظام

1. الواجهة الرئيسية للنظام



يتم أولاً إختيار طريقة الإتصال وهي إما الإتصال كزبون (الخيار الأول) أو الإتصال كمخدم (الخيار الثاني).

2. الواجهات الخاصة بجزئية الزبون في النظام



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) Successfully Sent Messages (0) Requested Messages

| 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 |

Client's Received Messages (1004) Successfully Received Message:

| 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 |

Request Student's Result Request Bulk Results Refreshment Close

Home

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

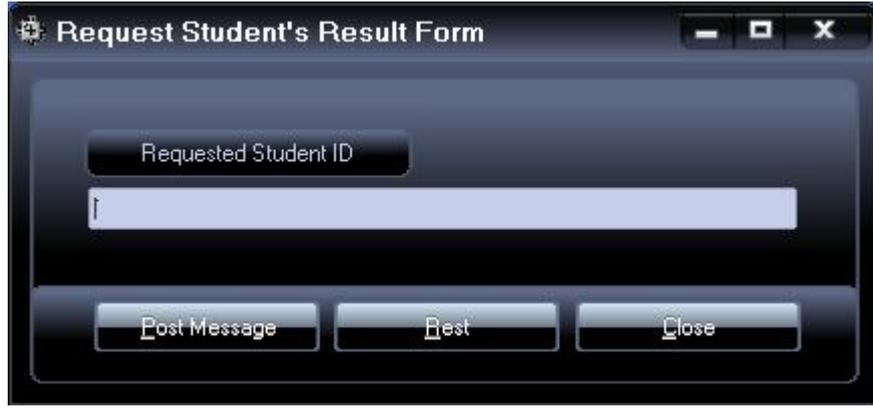
Request Bulk Students' Results Form

Requested Student ID

From 900000 Up to 900999

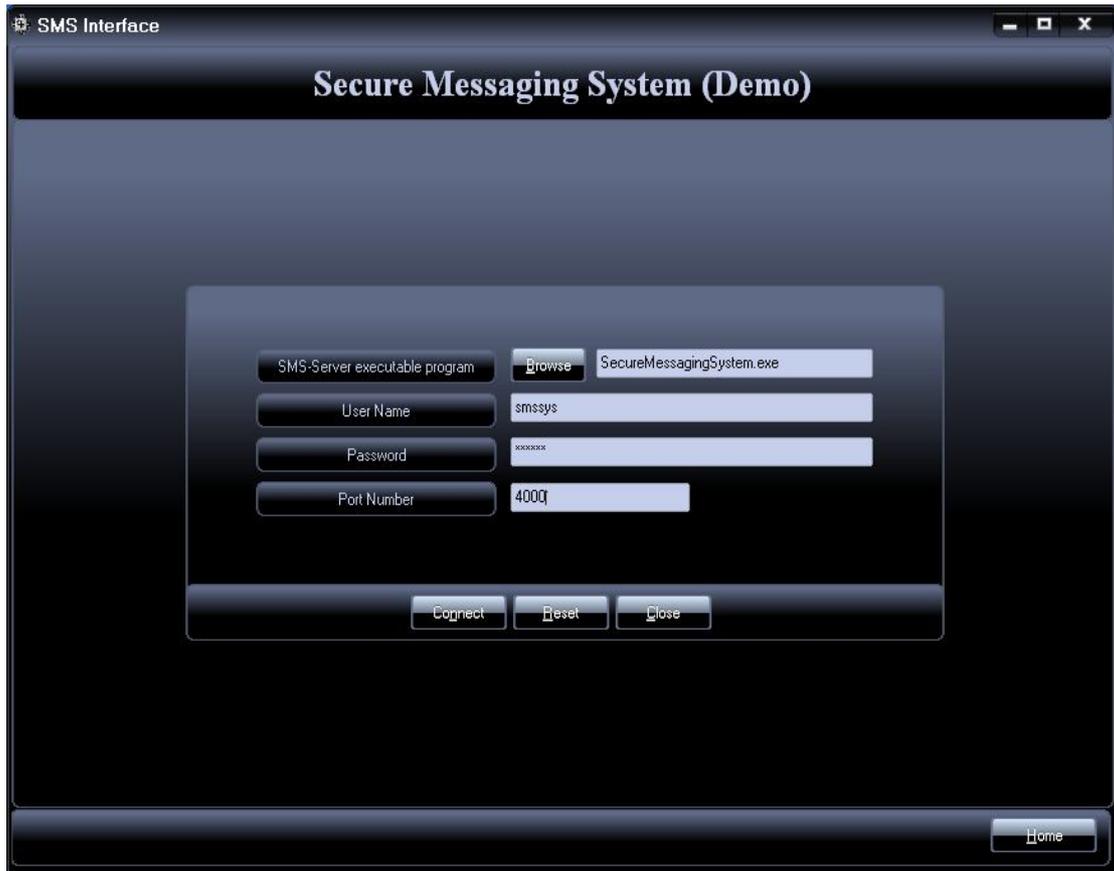
Post Bulk Messages Reset Close

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



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

3. الواجهات الخاصة بجزئية المخدم في النظام



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

```
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
core&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 المخدم، 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 | د | |
| 900018 | 473 | ن | |
| 900019 | 620 | ن | |
| 900020 | 561 | ن | |
| 900021 | 469 | ن | |

تظهر هذه الشاشة مباشرة بعد تشغيل نظام المخدم الخاص بنظام التراسل الآمن، وهي تعمل كواجهة لجداول قاعدة بيانات نظام المخدم. 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 مقسمة 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 السري (key.txt).

Users.txt

smssys,smsspw

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 المقدم 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 الأمانة. |
| SMSServer.log | يقوم بتسجيل حركة الرسائل الصادرة من نظام المخدّم وذلك قبل إرسالها قناة التراسل الأمانة. |
| SMSServerRe.log | يقوم بتسجيل حركة الرسائل الواردة إلى نظام المخدّم وذلك بعد إستلامها من قناة التراسل الأمانة. |

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

نظام الزبون - ملفات الأحداث الخاصة بالزبون

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

نظام المخدم - ملفات الأحداث الخاصة بالمخدم

| 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 المشفر | | | | | | | |
| 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 المشفر (طول مفتاح التشفير السري = 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 المشفر (طول مفتاح التشفير السري = 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 المشفر (طول مفتاح التشفير السري = 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 لتتواءم 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

```
1 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2 // MQ4CPP - Message queuing for C++
3 // Copyright (C) 2004-2007 Riccardo Pompeo (Italy)
4 //
5 // This library is free software; you can redistribute it and/or
6 // modify it under the terms of the GNU Lesser General Public
7 // License as published by the Free Software Foundation; either
8 // version 2.1 of the License, or (at your option) any later version.
9 //
10 // This library is distributed in the hope that it will be useful,
11 // but WITHOUT ANY WARRANTY; without even the implied warranty of
12 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
13 // Lesser General Public License for more details.
14 //
15 // You should have received a copy of the GNU Lesser General Public
16 // License along with this library; if not, write to the Free Software
17 // Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
18
19
20 #include "StoreForward.h"
21 #include "Logger.h"
22 #include "SMS.h"
23
24 class MyClient : public SMSClient
25 {
26 private:
27     unsigned itsCnt;
28
29 public:
30     MyClient(const char* theName, const char* theWorkingPath, const char* theHost,
31             short thePort, const char* theRemoteService)
32     {
33         itsCnt=0;
34         SCHEDULE(this,100);
35     };
36
37     virtual ~MyClient() {};
38
39 protected:
40     virtual void onWakeup(Wakeup* theMessage)
41     {
42         GetNewMessageFromDB();
43     };
44 };
45
46 class MyServer : public SMSServer
47 {
48 public:
49     MyServer(const char* theName) : SMSServer(theName)
50     {
51     };
52
53     virtual ~MyServer()
54     {
55     };
56
57 protected:
58     string service(string theBuffer)
59     {
60         string ReplyMessage = GetMessageFromSecureChannel(theBuffer);
61         return ReplyMessage;
62     };
63 };
64
65 void main_sleep(int val)
66 {
67     DISPLAY("...wait " << val << " secs...")
68     Thread::sleep(val*1000);
69 }
70
71 bool CheckUserAuthorization(char *UserName, char *Password)
72 {
73     FILE * pFile;
```

```

74     char buffer[100]="";
75     string LoginInfo, LogUserName, LogPassword, Uname, Psw;
76     int i, pos;
77     pFile = fopen ("users.txt", "r");
78     LoginInfo.erase();
79     LogUserName.erase();
80     LogPassword.erase();
81     if (pFile!=NULL)
82     {
83         fgets (buffer, sizeof(buffer), pFile);
84         for (i = 0; i < sizeof(buffer); i++)
85             if (buffer[i] != NULL)
86                 LoginInfo.append(1, buffer[i]);
87         pos = LoginInfo.find(',');
88         LogUserName.append(LoginInfo, 0, pos);
89         LogPassword.append(LoginInfo, pos + 1, LoginInfo.length());
90         fclose (pFile);
91     }
92     Uname.erase();
93     Psw.erase();
94     Uname.assign(Username);
95     Psw.assign>Password);
96     if ((LogUserName.compare(Uname) == 0) && (LogPassword.compare(Psw) == 0))
97         return true;
98     else
99         return false;
100 }
101
102 int main(int argv, char* argc[])
103 {
104     DISPLAY("MQ4CPP SecureMessagingSystem.cpp")
105     DISPLAY("This example shows how to implement store&forward as a Secure Messaging System")
106
107     bool client=false;
108     char* host=NULL;
109     int hport=0;
110
111     if(argv < 5)
112     {
113         DISPLAY("Client usage: SecureMessagingSystem -c hostip port username password")
114         DISPLAY("Server usage: SecureMessagingSystem -s port username password")
115         return 0;
116     }
117     else if(string(argc[1]).compare("-c")==0 && argv==6)
118     {
119         if (CheckUserAuthorization(argc[4], argc[5]))
120         {
121             client=true;
122             DISPLAY("Default host name=" << argc[2])
123             host=argc[2];
124             DISPLAY("Default host port=" << argc[3])
125             hport=atoi(argc[3]);
126         }
127         else
128         {
129             TRACE("This user has no sufficient access permissions, please check your
130             username or password and try again.");
131             return 0;
132         }
133     }
134     else if(string(argc[1]).compare("-s")==0 && argv==5)
135     {
136         if (CheckUserAuthorization(argc[3], argc[4]))
137         {
138             client=false;
139             DISPLAY("Server port=" << argc[2])
140             hport=atoi(argc[2]);
141         }
142         else
143         {
144             TRACE("This user has no sufficient access permissions, please check your
145             username or password and try again.");
146             return 0;

```

```

145     }
146 }
147 else
148 {
149     DISPLAY("Client usage: SecureMessagingSystem -c hostip port username password")
150     DISPLAY("Server usage: SecureMessagingSystem -s port username password")
151     return 0;
152 }
153
154 try
155 {
156     if(client==true)
157     {
158         DISPLAY("Starting client threads...")
159         STARTLOGGER("client.log")
160         LOG("!!!!!!! SecureMessagingSystem.cpp - client !!!!!!!")
161         MessageForwarder* aForwarder=new MessageForwarder("MyForwarder","tlog");
162         MyClient* aStorer=new MyClient("MyClient" , "tlog", host, hport, "MyServer"
163     );
164         main_sleep(-1);
165     }
166     else
167     {
168         DISPLAY("Starting server threads...")
169         STARTLOGGER("server.log")
170         LOG("!!!!!!! SecureMessagingSystem.cpp - server !!!!!!!")
171         MessageProxyFactory aFactory("MyFactory",hport);
172         MyServer* aServer=new MyServer("MyServer");
173         main_sleep(-1);
174     }
175     DISPLAY("...stopping threads...")
176     Thread::shutdownInProgress();
177     STOPLOGGER()
178     STOPREGISTRY()
179     STOPTIMER()
180 }
181 catch(Exception& ex)
182 {
183     TRACE(ex.getMessage().c_str())
184 }
185 catch(...)
186 {
187     TRACE("Unhandled exception")
188 }
189
190 DISPLAY("...done!")
191 if(client)
192     DISPLAY("See client.log for details")
193 else
194     DISPLAY("See server.log for details")
195 return 0;
196 }
197
198

```

SMS.h

```
=====
1  #ifndef _SMS_H_
2  #define _SMS_H_
3
4  #include "StoreForward.h"
5  #include "Logger.h"
6  #include <sql.h>
7
8  class SMS
9  {
10 public:
11     SMS();
12     ~SMS();
13
14 protected:
15     int print_error (SQLHENV     henv,
16                     SQLHDBC     hdbc,
17                     SQLHSTMT    hstmt);
18     int check_error (SQLHENV     henv,
19                     SQLHDBC     hdbc,
20                     SQLHSTMT    hstmt,
21                     SQLRETURN    frc);
22     void TerminateDBConnection(SQLHENV hEnv, SQLHDBC hDBC, SQLHSTMT hStmt);
23     DWORD TimeMilliseconds(LARGE_INTEGER theTime, LARGE_INTEGER theFrequencies);
24     void WriteToLogFile(char* FileName, string MessageID, int OriginalMsgLen, int CompressedMsgLen,
25                       int EncryptedMsgLen, DWORD ProcessTime);
26     string GetEncDecKey();
27     string ConvertToStr(UCHAR Str[], int StrLength);
28     void ConvertToHex(int Dec, string &Hex);
29     void ConvertToDec(char* Hex, int &Dec);
30     string EncryptMessage(string Message);
31     string DecryptMessage(string Message);
32
33 protected:
34     LARGE_INTEGER Frequencies;
35     LARGE_INTEGER Process_StartTime;
36     LARGE_INTEGER Process_EndTime;
37     string MessageID;
38     int OriginalMsgLen;
39     int CompressedMsgLen;
40     int EncryptedMsgLen;
41     DWORD ProcessTime;
42 };
43
44 class SMSClient : protected MessageStorer, SMS
45 {
46 public:
47     SMSClient(const char* theName, const char* theWorkingPath, const char* theHost,
48              short thePort, const char* theRemoteService);
49     ~SMSClient();
50     void GetNewMessageFromDB();
51
52 protected:
53     string generateServerBXMLMSG(string studentID, string MessageID);
54 };
55
56 class SMSServerReply : protected Observer, SMS
57 {
58 public:
59     SMSServerReply(const char* theName);
60     ~SMSServerReply();
61     void ProcessFeedbackMessage(string theBuffer);
62
63 protected:
64     void reverseServerBXMLMSG(string theBuffer, string StdInfo[]);
65 };
66
67 class SMSServer : protected Server, SMS
68 {
69 public:
70     SMSServer(const char* theName);
71     ~SMSServer();
72     string GetMessageFromSecureChannel(string theBuffer);
73

```

```
73 protected:
74     string generateClientBXMLMSG(string studentID, string studentGrade, char
studentResult, string MessageID);
75     void reverseClientBXMLMSG(string theBuffer, string StdInfo[]);
76 };
77 #endif // _SMS_H_
78
```



SMS.cpp

```
1 #include "StoreForward.h"
2 #include "Logger.h"
3 #include "Compression.h"
4 #include "examples/rijndael.cpp"
5 #include "Session.h"
6 #include "sms.h"
7 #include <string>
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <stdlib.h>
11 #include <memory.h>
12 #include <sql.h>
13 #include <sqlext.h>
14 #include <time.h>
15 using namespace std;
16
17 SMS::SMS ()
18 {
19 }
20
21 SMS::~SMS ()
22 {
23 }
24
25 /*****
26 ** - print_error - call SQLError(), display SQLSTATE and message
27 *****/
28 int SMS::print_error(SQLHENV henv,
29                     SQLHDBC hdbc,
30                     SQLHSTMT hstmt)
31 {
32     SQLCHAR buffer[SQL_MAX_MESSAGE_LENGTH + 1];
33     SQLCHAR sqlstate[SQL_SQLSTATE_SIZE + 1];
34     SQLINTEGER sqlcode;
35     SQLSMALLINT length;
36     while ( SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
37                    SQL_MAX_MESSAGE_LENGTH + 1, &length) == SQL_SUCCESS )
38     {
39         TRACE("**** ERROR ****");
40         TRACE("      SQLSTATE: "<<sqlstate);
41         TRACE("Native Error Code: "<<sqlcode);
42         TRACE(buffer);
43     };
44     return ( SQL_ERROR);
45 } /* end print_error */
46
47 /*****
48 ** - check_error - call print_error(), checks severity of return code
49 *****/
50 int SMS::check_error(SQLHENV henv,
51                     SQLHDBC hdbc,
52                     SQLHSTMT hstmt,
53                     SQLRETURN frc)
54 {
55     SQLRETURN rc;
56     print_error(henv, hdbc, hstmt);
57     switch (frc){
58     case SQL_SUCCESS : break;
59     case SQL_ERROR :
60     case SQL_INVALID_HANDLE:
61         TRACE(" ** FATAL ERROR, Attempting to rollback transaction **");
62         rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
63         if (rc != SQL_SUCCESS)
64             TRACE("Rollback Failed, Exiting application")
65         else
66             TRACE("Rollback Successful, Exiting application");
67         exit(frc);
68         break;
69     case SQL_SUCCESS WITH INFO :
70         TRACE(" ** Warning Message, application continuing");
71         break;
72     case SQL_NO_DATA_FOUND :
73         TRACE(" ** No Data Found ** ");
74         break;
```

```

75     default :
76         TRACE(" ** Invalid Return Code ** ");
77         TRACE(" ** Attempting to rollback transaction **");
78         SQLTransact(henv, hdbc, SQL_ROLLBACK);
79         exit(frc);
80         break;
81     }
82     return(SQL_SUCCESS);
83
84 } /* end check_error */
85
86 /*****
87 ** - TerminateDBConnection - call TerminateDBConnection(), terminate the database
88 connection
89 *****/
89 void SMS::TerminateDBConnection(SQLHENV hEnv, SQLHDBC hDBC, SQLHSTMT hStmt)
90 {
91     RETCODE retcode;
92     // Free the allocated statement handle
93     retcode = SQLFreeStmt (hStmt, SQL_DROP);
94     if (retcode != SQL_SUCCESS )
95         check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
96     // Disconnect from datasource
97     retcode = SQLDisconnect (hDBC);
98     if (retcode != SQL_SUCCESS )
99         print_error(hEnv, hDBC, SQL_NULL_HSTMT);
100    // Free the allocated connection handle
101    retcode = SQLFreeConnect (hDBC);
102    if (retcode != SQL_SUCCESS )
103        print_error(hEnv, hDBC, SQL_NULL_HSTMT);
104    // Free the allocated ODBC environment handle
105    retcode = SQLFreeEnv (hEnv);
106    if (retcode != SQL_SUCCESS )
107        print_error(hEnv, hDBC, SQL_NULL_HSTMT);
108 }
109
110 DWORD SMS::TimeMilliseconds(LARGE_INTEGER theTime,
111                             LARGE_INTEGER theFrequencies)
112 {
113     DWORD ClockTime    = theTime.u.LowPart;
114     DWORD freq         = theFrequencies.u.LowPart;
115     DWORD MilliSeconds = ((double)ClockTime/(double)freq) * 1000;
116     return MilliSeconds;
117 }
118
119 void SMS::WriteToLogFile(char* FileName, string MessageID, int OriginalMsgLen_, int
120                         CompressedMsgLen_,
121                         int EncryptedMsgLen_, DWORD ProcessTime_)
122 {
123     FILE * pFile;
124     char buffer[65]="", buffer1[65]="", buffer2[65]="";
125     pFile = fopen (FileName,"a+t");
126     if (pFile!=NULL)
127     {
128         string TraceLine;
129         TraceLine.append(MessageID, 0, MessageID.length());
130         TraceLine += ", ";
131         TraceLine.append(itoa (OriginalMsgLen_, buffer, 10), strlen(itoa
132 (OriginalMsgLen_, buffer, 10)));
133         TraceLine += ", ";
134         TraceLine.append(itoa (CompressedMsgLen_, buffer1, 10), strlen(itoa
135 (CompressedMsgLen_ , buffer1, 10)));
136         TraceLine += ", ";
137         TraceLine.append(itoa (EncryptedMsgLen_, buffer2, 10), strlen(itoa
138 (EncryptedMsgLen_, buffer2, 10)));
139         if (ProcessTime_ != NULL)
140         {
141             TraceLine += ", ";
142             char ProcessTimeStr[65];
143             _i64toa(ProcessTime_, ProcessTimeStr, 10);
144             TraceLine.append(ProcessTimeStr, strlen(ProcessTimeStr));
145         }
146         TraceLine += "\n";
147         fputs (TraceLine.c_str(), pFile);

```

```

144     fclose (pFile);
145 }
146 }
147
148 string SMS::GetEncDecKey()
149 {
150     FILE * pFile;
151     char buffer[70]="";
152     string theKey;
153     pFile = fopen ("key.txt","r");
154     if (pFile!=NULL)
155     {
156         fgets (buffer, sizeof(buffer), pFile);
157         theKey.erase();
158         for (int i=0; i<sizeof(buffer); i++)
159             if (buffer[i] != NULL)
160                 theKey.append(1, buffer[i]);
161         fclose (pFile);
162     }
163     return theKey;
164 }
165
166 string SMS::ConvertToStr(UCHAR Str[], int StrLength)
167 {
168     int i;
169     string StrResult;
170     StrResult.erase();
171     for (i = 0; i < StrLength; i++)
172     {
173         if (Str[i] != NULL)
174             StrResult.append(1, Str[i]);
175     }
176     return StrResult;
177 }
178
179 void SMS::ConvertToHex(int Dec, string &Hex)
180 {
181     char Hexa[10];
182     switch(Dec)
183     {
184     case 10:
185         Hex="A";
186         break;
187     case 11:
188         Hex="B";
189         break;
190     case 12:
191         Hex="C";
192         break;
193     case 13:
194         Hex="D";
195         break;
196     case 14:
197         Hex="E";
198         break;
199     case 15:
200         Hex="F";
201         break;
202     default :
203         Hex=itoa(Dec, Hexa, 10);
204     }
205 }
206
207 void SMS::ConvertToDec(char* Hex, int &Dec)
208 {
209     switch(Hex[0])
210     {
211     case 'A':
212         Dec=10;
213         break;
214     case 'B':
215         Dec=11;
216         break;
217     case 'C':

```

```

218     Dec=12;
219     break;
220 case 'D':
221     Dec=13;
222     break;
223 case 'E':
224     Dec=14;
225     break;
226 case 'F':
227     Dec=15;
228     break;
229 default :
230     Dec = atoi(Hex);
231 }
232 }
233
234 string SMS::EncryptMessage(string Message)
235 {
236     string msg="";
237     string blockMsg="";
238     string Hex;
239     unsigned char cipherMsg[16]="";
240     unsigned char plainMsg[16]="";
241     Rijndael rijnd;
242     char ch;
243     int counter=0, i=0, j=0, k=0, l=0, m=0, x=0;
244     // Get the encryption/decryption key
245     //=====
246     string KeyStr = GetEncDecKey();
247     unsigned char *key = new (unsigned char[KeyStr.length()+1]);
248
249     for (i=0; i<(int)KeyStr.length(); i++)
250         key[i] = KeyStr.at(i);
251     int KeySize = KeyStr.length();
252     if (KeyStr.length() == 16)
253         x = rijnd.init(Rijndael::CBC, Rijndael::Encrypt, key, Rijndael::Key16Bytes,
254 0);
255     else
256         if (KeyStr.length() == 24)
257             x = rijnd.init(Rijndael::CBC, Rijndael::Encrypt, key, Rijndael::
258 Key24Bytes, 0);
259         else
260             x = rijnd.init(Rijndael::CBC, Rijndael::Encrypt, key, Rijndael::
261 Key32Bytes, 0);
262     //=====
263     int lastBlockSize = (Message.length() % 16);
264     ConvertToHex(lastBlockSize, Hex);
265     msg.erase();
266     memset(plainMsg, '\0', 16);
267     memset(cipherMsg, '\0', 16);
268     blockMsg.erase();
269     msg.append(1, Hex.at(0));
270     for (k=0; k<(int)Message.length();k++)
271     {
272         msg.append(1, Message.at(k));
273     }
274     if (msg.length()<16)
275     {
276         ch=' ';
277         for (k=0; k<(int)msg.length();k++)
278         {
279             plainMsg[k]=msg.at(k);
280         }
281         for (k=msg.length(); k<16;k++)
282         {
283             plainMsg[k]= ch;
284         }
285         l = rijnd.blockEncrypt(plainMsg, KeySize*8, cipherMsg);
286         for (m=0; m<16; m++)
287         {
288             blockMsg.append(1, cipherMsg[m]);
289         }
290     }

```

```

287     }
288     else
289     for (i=0; i<(int)msg.length(); i++)
290     {
291         counter++;
292         plainMsg[j]=msg.at(i);
293         j++;
294         if (j == 16)
295         {
296             l = rijnd.blockEncrypt(plainMsg, 16*8, cipherMsg);
297             for (m=0; m<16; m++)
298             {
299                 blockMsg.append(l, cipherMsg[m]);
300             }
301             j=0;
302         }
303         if (counter == msg.length() && j != 0)
304         {
305             ch=' ';
306             for (k=j; k<16;k++)
307             {
308                 plainMsg[k]= ch;
309             }
310             l = rijnd.blockEncrypt(plainMsg, (sizeof(key)-1)*8, cipherMsg);
311
312             for ( m=0; m<16; m++)
313             {
314                 blockMsg.append(l, cipherMsg[m]);
315             }
316         }
317         return blockMsg;
318     }
319
320 string SMS::DecryptMessage(string Message)
321 {
322     string blockMsg="";
323     unsigned char cipherMsg[16];
324     unsigned char plainMsg[16];
325     Rijndael rijnd;
326     int x;
327     // Get the encryption/decryption key
328     //=====
329     string KeyStr = GetEncDecKey();
330     unsigned char *key = new (unsigned char[KeyStr.length()+1]);
331     for (int i=0; i<(int)KeyStr.length(); i++)
332         key[i] = KeyStr.at(i);
333     int KeySize = KeyStr.length();
334     if (KeyStr.length() == 16)
335         x = rijnd.init(Rijndael::CBC, Rijndael::Decrypt, key, Rijndael::Key16Bytes, 0);
336     else
337         if (KeyStr.length() == 24)
338             x = rijnd.init(Rijndael::CBC, Rijndael::Decrypt, key, Rijndael::Key24Bytes,
339 0);
340         else
341             x = rijnd.init(Rijndael::CBC, Rijndael::Decrypt, key, Rijndael::Key32Bytes,
342 0);
343     //=====
344     memset(plainMsg, '\0', 16);
345     memset(cipherMsg, '\0', 16);
346     int counter=0, i=0, j=0, k=0, l=0, m=0;
347     int firstBlock=0, lastBlockSize=0, start=0;
348     char Hex[1]="";
349     blockMsg.erase();
350     for (i=0; i<(int)Message.length(); i++)
351     {
352         counter++;
353         cipherMsg[j]=Message.at(i);
354         j++;
355         if (j == 16)
356         {
357             l = rijnd.blockDecrypt(cipherMsg, KeySize*8, plainMsg);

```

```

356         if (firstBlock == 0)
357         {
358             start = 1;
359             firstBlock = 1;
360             Hex[0] = plainMsg[0];
361             ConvertToDec(Hex, lastBlockSize);
362         }
363         else
364         {
365             start = 0;
366         }
367         if (counter == Message.length())
368         {
369             for ( m=start; m<=lastBlockSize; m++)
370             {
371                 blockMsg.append(1, plainMsg[m]);
372             }
373         }
374         else
375         {
376             for ( m=start; m<16; m++)
377             {
378                 blockMsg.append(1, plainMsg[m]);
379             }
380         }
381         j=0;
382     }
383 }
384 return blockMsg;
385 }
386
387 SMSClient::SMSClient(const char* theName, const char* theWorkingPath, const char*
theHost, short thePort, const char* theRemoteService)
:MessageStorer(theName, theWorkingPath, theHost, thePort, theRemoteService)
388 {
389 }
390
391
392 SMSClient::~SMSClient()
393 {
394 }
395
396 void SMSClient::GetNewMessageFromDB()
397 {
398     QueryPerformanceFrequency(&Frequencies);
399     QueryPerformanceCounter(&Process_StartTime);
400     bool foundNewRec = false;
401     char* aString;
402     string Message, StudentID, MessageID;
403     SQLHENV hEnv = NULL; // Env Handle from SQLAllocEnv()
404     SQLHDBC hDBC = NULL; // Connection handle
405     SQLHSTMT hStmt = NULL; // Statement handle
406     RETCODE retcode;
407     UCHAR ID[5]=""; // Model buffer
408     UCHAR StdID[10]=""; // Model buffer
409     // Get the new incoming message from database
410     //=====
411     UCHAR szDSN[SQL_MAX_DSN_LENGTH] = "SMS_DSN"; // Data Source Name buffer
412     UCHAR szUID[13] = ""; // User ID buffer
413     UCHAR szPasswd[7] = ""; // Password buffer
414     SDWORD cbModel, cbModel1; // Model buffer bytes recieved
415     UCHAR szSqlStr[128]= "SELECT * FROM RequestsMessagesQueue where Posted=0";
416     // Initialize connection for selection
417     //=====
418     // Allocate memory for ODBC Environment handle
419     retcode = SQLAllocEnv (&hEnv);
420     if (retcode != SQL_SUCCESS )
421         check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
422     // Allocate memory for the connection handle
423     retcode = SQLAllocConnect (hEnv, &hDBC);
424     if (retcode != SQL_SUCCESS )
425         check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
426     // Connect to the data source "szDSN" using userid and password.
427     retcode = SQLConnect (hDBC, szDSN, SQL_NTS, szUID, SQL_NTS, szPasswd, SQL_NTS);
428     if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)

```

```

429 {
430     // Allocate memory for the statement handle
431     retcode = SQLAllocStmt (hDBC, &hStmt);
432     if (retcode != SQL_SUCCESS )
433         check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
434     // Prepare the SQL statement by assigning it to the statement handle
435     retcode = SQLPrepare (hStmt, szSqlStr, sizeof (szSqlStr));
436     if (retcode != SQL_SUCCESS )
437         check_error(hEnv, hDBC, hStmt, retcode);
438     // Execute the SQL statement handle
439     retcode = SQLExecDirect(hStmt, szSqlStr, SQL_NTS);
440     if (retcode != SQL_SUCCESS )
441         check_error(hEnv, hDBC, hStmt, retcode);
442     // Project columns
443     retcode = SQLBindCol (hStmt, 1, SQL_C_CHAR, ID, sizeof(ID), &cbModel);
444     if (retcode != SQL_SUCCESS )
445         check_error(hEnv, hDBC, hStmt, retcode);
446     retcode = SQLBindCol (hStmt, 2, SQL_C_CHAR, StdID, sizeof(StdID), &cbModel1);
447     if (retcode != SQL_SUCCESS )
448         check_error(hEnv, hDBC, hStmt, retcode);
449 }
450 while ((retcode = SQLFetch(hStmt)) == SQL_SUCCESS)
451 {
452     ProcessTime = TimeMilliSeconds(Process_StartTime, Frequencies);
453     foundNewRec = true;
454     // Convert the message into binaryXML format (Translator Filter)
455     StudentID = ConvertToStr(StdID, sizeof(StdID));
456     MessageID = ConvertToStr(ID, sizeof(ID));
457     Message = generateServerBXMLMSG(StudentID, MessageID);
458     OriginalMsgLen = Message.length();
459     // Compress the binaryXML message (Compression Filter)
460     PacketCompression compression(true);
461     Message = compression.deflate(Message);
462     CompressedMsgLen = Message.length();
463     // Encrypt the message using Rijndael algorithm (Encryption Filter)
464     Message = EncryptMessage(Message);
465     EncryptedMsgLen = Message.length();
466     // Send message via messaging channel (Message Multiplexer Filter)
467     send(Message);
468     // Update the selected recodes
469     SQLHENV hEnvUpdt = NULL; // Env Handle from SQLAllocEnv()
470     SQLHDBC hDBCUpdt = NULL; // Connection handle
471     SQLHSTMT hStmtUpdt = NULL; // Statement handle
472     RETCODE retcd;
473     UCHAR szSqlStrUpdate[128] = "";
474     string SqlStr = "Update RequestsMessagesQueue Set Posted=2 Where ID = ";
475     SqlStr.append(MessageID, 0, MessageID.length());
476     int i;
477     for (i=0; i<(int)SqlStr.length(); i++)
478         szSqlStrUpdate[i]=SqlStr.at(i);
479     szSqlStrUpdate[i]=NULL;
480     // Initialize connection for update
481     //=====
482     // Allocate memory for ODBC Environment handle
483     retcd = SQLAllocEnv (&hEnvUpdt);
484     if (retcd != SQL_SUCCESS )
485         check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
486     // Allocate memory for the connection handle
487     retcd = SQLAllocConnect (hEnvUpdt, &hDBCUpdt);
488     if (retcd != SQL_SUCCESS )
489         check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
490     // Connect to the data source "szDSN" using userid and password.
491     retcd = SQLConnect (hDBCUpdt, szDSN, SQL_NTS, szUID, SQL_NTS, szPasswd, SQL_NTS) ✓
492 ;
493 if (retcd == SQL_SUCCESS || retcd == SQL_SUCCESS_WITH_INFO)
494 {
495     // Allocate memory for the statement handle
496     retcd = SQLAllocStmt (hDBCUpdt, &hStmtUpdt);
497     if (retcd != SQL_SUCCESS )
498         check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
499     // Prepare the SQL statement by assigning it to the statement handle
500     retcd = SQLPrepare (hStmtUpdt, szSqlStrUpdate, sizeof (szSqlStrUpdate));
501     if (retcd != SQL_SUCCESS )
502         check_error(hEnvUpdt, hDBCUpdt, hStmtUpdt, retcd);

```

```

502         // Execute the SQL statement handle
503         retcd = SQLExecDirect(hStmtUpdt, szSqlStrUpdate, SQL_NTS);
504         if (retcd != SQL_SUCCESS )
505             check_error(hEnvUpdt, hDBCUpdt, hStmtUpdt, retcd);
506         TerminateDBConnection(hEnvUpdt, hDBCUpdt, hStmtUpdt);
507     }
508     for (i=0; i<sizeof(StdID); i++)
509         ID[i]=NULL;
510     for (i=0; i<sizeof(StdID); i++)
511         StdID[i]=NULL;
512 }
513 if (retcode != SQL_NO_DATA_FOUND)
514     check_error(hEnv, hDBC, hStmt, retcode);
515 // Terminate the database connection
516 TerminateDBConnection(hEnv, hDBC, hStmt);
517 if (foundNewRec)
518     // Trace process
519     WriteToLogFile("SMSClient.log", MessageID, OriginalMsgLen, CompressedMsgLen,
520 EncryptedMsgLen,
521 ProcessTime);
522 }
523 string SMSClient::generateServerBXMLMSG(string studentID, string MessageID)
524 {
525     ListProperty itsTxStructure;
526
527     TRACE("Populating TX structure")
528     itsTxStructure.free();
529
530     StringProperty* aStringProperty studentID=new StringProperty("studentID");
531     aStringProperty_studentID->set(studentID);
532     itsTxStructure.add(aStringProperty_studentID);
533
534     StringProperty* aStringProperty_MessageID=new StringProperty("MessageID");
535     aStringProperty_MessageID->set(MessageID);
536     itsTxStructure.add(aStringProperty_MessageID);
537
538     string aString;
539     encodeProperties(itsTxStructure,aString);
540     return aString;
541 }
542
543
544 SMSServerReply::SMSServerReply(const char* theName):Observer(theName)
545 {
546 }
547
548 SMSServerReply::~SMSServerReply()
549 {
550 }
551
552 void SMSServerReply::ProcessFeedbackMessage(string theBuffer)
553 {
554     //=====
555     // Insert the received student's result into StudentResults table.
556     //=====
557     string Message, StdGrade, StdResult;
558     string StdInfo[4];
559     // Get message from the messaging channel
560     Message.append(theBuffer, 0, theBuffer.length());
561     EncryptedMsgLen = Message.length();
562     // Decrypt the received message
563     Message = DecryptMessage(Message);
564     CompressedMsgLen = Message.length();
565     // Decompress the message
566     PacketCompression compression(true);
567     Message = compression.inflate(Message);
568     OriginalMsgLen = Message.length();
569     reverseServerBXMLMSG(Message, StdInfo);
570     //=====
571     SQLHENV hEnv = NULL; // Env Handle from SQLAllocEnv()
572     SQLHDBC hDBC = NULL; // Connection handle
573     SQLHSTMT hStmt = NULL; // Statement handle
574     RETCODE retcode;

```

```

575     int i;
576     UCHAR szDSN[SQL_MAX_DSN_LENGTH] = "SMS_DSN";// Data Source Name buffer
577     UCHAR szUID[13] = "";// User ID buffer
578     UCHAR szPasswd[7] = "";// Password buffer
579     UCHAR szSqlStr[128];
580     string SqlStr = "Insert INTO StudentResults (StudentID, StudentGrade,
StudentResult) Values (";
581     SqlStr.append(StdInfo[0], 0, StdInfo[0].length());
582     SqlStr += ", ";
583     SqlStr.append(StdInfo[1], 0, StdInfo[1].length());
584     SqlStr += ", ";
585     SqlStr.append(StdInfo[2], 0, StdInfo[2].length());
586     SqlStr += ")";
587     for (i=0; i<(int)SqlStr.length(); i++)
588         szSqlStr[i]=SqlStr.at(i);
589     szSqlStr[i]=NULL;
590     // Allocate memory for ODBC Environment handle
591     retcode = SQLAllocEnv (&hEnv);
592     if (retcode != SQL_SUCCESS )
593         check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
594     // Allocate memory for the connection handle
595     retcode = SQLAllocConnect (hEnv, &hDBC);
596     if (retcode != SQL_SUCCESS )
597         check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
598     // Connect to the data source "szDSN" using userid and password.
599     retcode = SQLConnect (hDBC, szDSN, SQL_NTS, szUID, SQL_NTS, szPasswd,SQL_NTS);
600     if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
601     {
602         // Allocate memory for the statement handle
603         retcode = SQLAllocStmt (hDBC, &hStmt);
604         if (retcode != SQL_SUCCESS )
605             check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
606         // Prepare the SQL statement by assigning it to the statement handle
607         retcode = SQLPrepare (hStmt, szSqlStr, sizeof (szSqlStr));
608         if (retcode != SQL_SUCCESS )
609             check_error(hEnv, hDBC, hStmt, retcode);
610         // Execute the SQL statement handle
611         retcode = SQLExecDirect(hStmt, szSqlStr, SQL_NTS);
612         if (retcode != SQL_SUCCESS )
613             check_error(hEnv, hDBC, hStmt, retcode);
614     }
615     // Terminate the database connection
616     TerminateDBConnection(hEnv, hDBC, hStmt);
617
618     //=====
619     // Update the RequestsMessagesQueue table.
620     //=====
621     SQLHENV hEnvUpdt = NULL; // Env Handle from SQLAllocEnv()
622     SQLHDBC hDBCUpdt = NULL; // Connection handle
623     SQLHSTMT hStmntUpdt = NULL;// Statement handle
624     RETCODE retcd;
625     UCHAR szSqlStrUpdate[128]="";
626     string SqlStrUpdt = "Update RequestsMessagesQueue Set Posted=1, SentMessageDateTime
=getdate() Where ID =";
627     SqlStrUpdt.append(StdInfo[3], 0, StdInfo[3].length());
628     for (i=0; i<(int)SqlStrUpdt.length(); i++)
629         szSqlStrUpdate[i]=SqlStrUpdt.at(i);
630     szSqlStrUpdate[i]=NULL;
631     // Initialize connection for update
632     //=====
633     // Allocate memory for ODBC Environment handle
634     retcd = SQLAllocEnv (&hEnvUpdt);
635     if (retcd != SQL_SUCCESS )
636         check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
637     // Allocate memory for the connection handle
638     retcd = SQLAllocConnect (hEnvUpdt, &hDBCUpdt);
639     if (retcd != SQL_SUCCESS )
640         check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
641     // Connect to the data source "szDSN" using userid and password.
642     retcd = SQLConnect (hDBCUpdt, szDSN, SQL_NTS, szUID, SQL_NTS, szPasswd,SQL_NTS);
643     if (retcd == SQL_SUCCESS || retcd == SQL_SUCCESS_WITH_INFO)
644     {
645         // Allocate memory for the statement handle
646         retcd = SQLAllocStmt (hDBCUpdt, &hStmntUpdt);

```

```

647     if (retcd != SQL_SUCCESS )
648         check_error(hEnvUpdt, hDBCUpdt, SQL_NULL_HSTMT, retcd);
649     // Prepare the SQL statement by assigning it to the statement handle
650     retcd = SQLPrepare (hStmtUpdt, szSqlStrUpdate, sizeof (szSqlStrUpdate));
651     if (retcd != SQL_SUCCESS )
652         check_error(hEnvUpdt, hDBCUpdt, hStmtUpdt, retcd);
653     // Execute the SQL statement handle
654     retcd = SQLExecDirect(hStmtUpdt, szSqlStrUpdate, SQL_NTS);
655     if (retcd != SQL_SUCCESS )
656         check_error(hEnvUpdt, hDBCUpdt, hStmtUpdt, retcd);
657     TerminateDBCConnection(hEnvUpdt, hDBCUpdt, hStmtUpdt);
658     QueryPerformanceFrequency(&Frequencies);
659     QueryPerformanceCounter(&Process_EndTime);
660     ProcessTime = TimeMilliseconds(Process_EndTime, Frequencies);
661     // Trace process
662     WriteToLogFile("SMSSClientRe.log", StdInfo[3], OriginalMsgLen, CompressedMsgLen,
663     EncryptedMsgLen,
664         ProcessTime);
665 }
666
667 void SMSServerReply::reverseServerBXMLMSG(string theBuffer, string StdInfo[])
668 {
669     ListProperty itsRxStructure;
670
671     decodeProperties(theBuffer, itsRxStructure);
672     TRACE("Retrieving data from structure");
673
674     Property* aProperty_studentID=itsRxStructure.get("studentID");
675     if(aProperty_studentID!=NULL && aProperty_studentID->is(PROPERTY_STRING))
676     {
677         StdInfo[0]=((StringProperty*)aProperty_studentID)->get();
678     }
679
680     Property* aProperty_studentGrade=itsRxStructure.get("studentGrade");
681     if(aProperty_studentGrade!=NULL && aProperty_studentGrade->is(PROPERTY_STRING))
682     {
683         StdInfo[1]=((StringProperty*)aProperty_studentGrade)->get();
684     }
685
686     Property* aProperty_studentResult=itsRxStructure.get("studentResult");
687     if(aProperty_studentResult!=NULL && aProperty_studentResult->is(PROPERTY_CHAR))
688     {
689         StdInfo[2]=((CharProperty*)aProperty_studentResult)->get();
690     }
691
692     Property* aProperty_MessageID=itsRxStructure.get("MessageID");
693     if(aProperty_MessageID!=NULL && aProperty_MessageID->is(PROPERTY_STRING))
694     {
695         StdInfo[3]=((StringProperty*)aProperty_MessageID)->get();
696     }
697
698     TRACE("Storing received structure")
699     try
700     {
701         ofstream aStream("rxstruct.log");
702         itsRxStructure.serialize(aStream);
703         aStream.close();
704     }
705     catch(...)
706     {
707         TRACE("Exception during storing of file ")
708     }
709 }
710
711 SMSServer:: SMSServer(const char* theName) : Server(theName)
712 {
713 }
714
715 SMSServer::~SMSServer()
716 {
717 }
718
719 string SMSServer::GetMessageFromSecureChannel(string theBuffer)

```

```

720 {
721     QueryPerformanceFrequency(&Frequencies);
722     ostream aStream;
723     string Message, StudentID, StudentGrade, StudentResult, MessageID;
724     string StdInfo[2];
725     Message.erase();
726     // Get message from the messaging channel
727     Message.append(theBuffer, 0, theBuffer.length());
728     EncryptedMsgLen = Message.length();
729     // Decrypt the received message
730     Message = DecryptMessage(Message);
731     CompressedMsgLen = Message.length();
732     // Decompress the message
733     PacketCompression compression(true);
734     Message = compression.inflate(Message);
735     OriginalMsgLen = Message.length();
736     // Trace process
737     reverseClientBXMLMSG(Message, StdInfo);
738     WriteToLogFile("SMSServer.log", StdInfo[1], OriginalMsgLen, CompressedMsgLen,
739                 NULL);
740     //=====
741     SQLHENV hEnv = NULL; // Env Handle from SQLAllocEnv()
742     SQLHDBC hDBC = NULL; // Connection handle
743     SQLHSTMT hStmt = NULL; // Statement handle
744     RETCODE retcode;
745     UCHAR StdID[10]=""; // Model buffer
746     UCHAR StdGrade[4]=""; // Model buffer
747     UCHAR StdResult[2]=""; // Model buffer
748     // Get the new incoming message from database
749     //=====
750     UCHAR szDSN[SQL_MAX_DSN_LENGTH] = "STDRESULTS_DSN"; // Data Source Name buffer
751     UCHAR szUID[13] = ""; // User ID buffer
752     UCHAR szPasswd[7] = ""; // Password buffer
753     SDWORD cbModel, cbModell, cbModel2; // Model buffer bytes recieved
754     UCHAR szSqlStr[128] = "";
755     string SqlStr = "SELECT * FROM StudentResults Where StdID = ";
756     SqlStr.append(StdInfo[0], 0, StdInfo[0].length());
757     int i;
758     for (i=0; i<(int)SqlStr.length(); i++)
759         szSqlStr[i]=SqlStr.at(i);
760     szSqlStr[i]=NULL;
761     // Initialize connection for selection
762     //=====
763     // Allocate memory for ODBC Environment handle
764     retcode = SQLAllocEnv (&hEnv);
765     if (retcode != SQL_SUCCESS )
766         check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
767     // Allocate memory for the connection handle
768     retcode = SQLAllocConnect (hEnv, &hDBC);
769     if (retcode != SQL_SUCCESS )
770         check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
771     // Connect to the data source "szDSN" using userid and password.
772     retcode = SQLConnect (hDBC, szDSN, SQL_NTS, szUID, SQL_NTS, szPasswd, SQL_NTS);
773     if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
774     {
775         // Allocate memory for the statement handle
776         retcode = SQLAllocStmt (hDBC, &hStmt);
777         if (retcode != SQL_SUCCESS )
778             check_error(hEnv, hDBC, SQL_NULL_HSTMT, retcode);
779         // Prepare the SQL statement by assigning it to the statement handle
780         retcode = SQLPrepare (hStmt, szSqlStr, sizeof (szSqlStr));
781         if (retcode != SQL_SUCCESS )
782             check_error(hEnv, hDBC, hStmt, retcode);
783         // Execute the SQL statement handle
784         retcode = SQLExecDirect(hStmt, szSqlStr, SQL_NTS);
785         if (retcode != SQL_SUCCESS )
786             check_error(hEnv, hDBC, hStmt, retcode);
787         // Project columns
788         retcode = SQLBindCol (hStmt, 2, SQL_C_CHAR, StdID, sizeof(StdID), &cbModel);
789         if (retcode != SQL_SUCCESS )
790             check_error(hEnv, hDBC, hStmt, retcode);
791         retcode = SQLBindCol (hStmt, 3, SQL_C_CHAR, StdGrade, sizeof(StdGrade), &
cbModell);

```

```

792         if (retcode != SQL_SUCCESS )
793             check_error(hEnv, hDBC, hStmt, retcode);
794         retcode = SQLBindCol (hStmt, 4, SQL_C_CHAR, StdResult, sizeof(StdResult), &
cbModel2);
795         if (retcode != SQL SUCCESS )
796             check_error(hEnv, hDBC, hStmt, retcode);
797     }
798     bool ResultExist = false;
799     StudentID.erase();
800     StudentGrade.erase();
801     StudentResult.erase();
802     MessageID.erase();
803     while ((retcode = SQLFetch(hStmt)) == SQL_SUCCESS)
804     {
805         ResultExist = true;
806         StudentID = ConvertToStr(StdID, sizeof(StdID));
807         StudentGrade = ConvertToStr(StdGrade, sizeof(StdGrade));
808         StudentResult = ConvertToStr(StdResult, sizeof(StdResult));
809         MessageID.append(StdInfo[1], 0, StdInfo[1].length());
810     }
811     // Terminate the database connection
812     TerminateDBConnection(hEnv, hDBC, hStmt);
813     // If no results exist, then return the ID with zeros
814     if (ResultExist == false)
815     {
816         StudentID.append(StdInfo[0], 0, StdInfo[0].length());
817         StudentGrade.append(1, '0');
818         StudentResult.append(1, 'N');
819         MessageID.append(StdInfo[1], 0, StdInfo[1].length());
820     }
821     // Convert the message into binaryXML format (Translator Filter)
822     string ReplyMessage = generateClientBXMLMSG(StudentID, StudentGrade, StudentResult.
at(0), MessageID);
823     OriginalMsgLen = ReplyMessage.length();
824     // Compress the binaryXML message (Compression Filter)
825     ReplyMessage = compression.deflate(ReplyMessage);
826     CompressedMsgLen = ReplyMessage.length();
827     // Encrypt the message using Rijndael algorithm (Encryption Filter)
828     ReplyMessage = EncryptMessage(ReplyMessage);
829     EncryptedMsgLen = ReplyMessage.length();
830     // Trace process
831     WriteToLogFile("SMSServerRe.log", MessageID, OriginalMsgLen, CompressedMsgLen,
EncryptedMsgLen,
832                 NULL);
833
834     return ReplyMessage;
835 }
836
837 string SMSServer::generateClientBXMLMSG(string studentID, string studentGrade, char
studentResult, string MessageID)
838 {
839     ListProperty itsTxStructure;
840
841     TRACE("Populating TX structure")
842     itsTxStructure.free();
843
844     StringProperty* aStringProperty_studentID=new StringProperty("studentID");
845     aStringProperty_studentID->set(studentID);
846     itsTxStructure.add(aStringProperty_studentID);
847
848     StringProperty* aStringProperty_studentGrade=new StringProperty("studentGrade");
849     aStringProperty_studentGrade->set(studentGrade);
850     itsTxStructure.add(aStringProperty_studentGrade);
851
852     CharProperty* aCharProperty_studentResult=new CharProperty("studentResult");
853     aCharProperty_studentResult->set(studentResult);
854     itsTxStructure.add(aCharProperty_studentResult);
855
856     StringProperty* aStringProperty_MessageID=new StringProperty("MessageID");
857     aStringProperty_MessageID->set(MessageID);
858     itsTxStructure.add(aStringProperty_MessageID);
859
860     string aString;
861     encodeProperties(itsTxStructure, aString);

```

```
862     return aString;
863 }
864
865 void SMSServer::reverseClientBXMLMSG(string theBuffer, string StdInfo[])
866 {
867     ListProperty itsRxStructure;
868
869     decodeProperties(theBuffer,itsRxStructure);
870     TRACE("Retrieving data from structure")
871
872     Property* aProperty_studentID=itsRxStructure.get("studentID");
873     if(aProperty_studentID!=NULL && aProperty_studentID->is(PROPERTY_STRING))
874     {
875         StdInfo[0]=((StringProperty*)aProperty_studentID)->get();
876     }
877
878     Property* aProperty_MessageID=itsRxStructure.get("MessageID");
879     if(aProperty_MessageID!=NULL && aProperty_MessageID->is(PROPERTY_STRING))
880     {
881         StdInfo[1]=((StringProperty*)aProperty_MessageID)->get();
882     }
883
884     TRACE("Storing received structure")
885     try
886     {
887         ofstream aStream("rxstruct.log");
888         itsRxStructure.serialize(aStream);
889         aStream.close();
890     }
891     catch(...)
892     {
893         TRACE("Exception during storing of file ")
894     }
895 }
896
897
```