#include "stdafx.h"

#include "stdlib.h"

//#include "../include/CkCrypt2.h"

#include <Windows.h> // needed by CHILKAT e.g. for SystemTim

#include <CkRsa.h>

#include <CkCrypt2.h>

// added for CHILKAT

#include "..\include\ckimap.h"

//#include "..\include\tchar.h"

//

// need this for various things

extern "C" {

#include "../include/allC.h"

#include "../include/allUnicodeC.h"

}

#include "../include/allUnicode.h"

#include <CkSocket.h>

#include <conio.h>

// get strings, because the const char\* causes problems....

#include <iostream>

#include <string>

#include <vector>

#include <string.h>

#include <sstream>

//#include <boost/algorithm/string/split.hpp>

//#include <boost/algorithm/string/classification.hpp>

#include "SHA.h"

#include "RSA.h"

/\*

#include "RSApublica.h"

#include "RSApublicb.h"

#include "RSAprivate.h"

\*/

using namespace std;

//Spliited

std::vector<std::string> &split(const std::string &s, char delim, std::vector<std::string> &elems) {

std::stringstream ss(s);

std::string item;

while (std::getline(ss, item, delim)) {

elems.push\_back(item);

}

return elems;

}

//Used to split a string with a delimiter

std::vector<std::string> split(const std::string &s, char delim) {

std::vector<std::string> elems;

&split(s, delim, elems);

return elems;

}

// Sockets

void SocketSend (void)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Setup Sockets

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//CkSocket listenSocket;

CkSocket connectedSocket;

bool success;

success = connectedSocket.UnlockComponent("T06152014Socket\_r6KrKN4ekQNz");

if (success != true) {

printf("%s\n",connectedSocket.lastErrorText());

return;

}

/\*

bool success;

success = listenSocket.UnlockComponent("T06152014Socket\_r6KrKN4ekQNz");

if (success != true) {

printf("%s\n",listenSocket.lastErrorText());

return;

}

// Bind to a port and listen for incoming connections:

// This example will listen at port 5555 and allows for a backlog

// of 25 pending connection requests.

success = listenSocket.BindAndListen(5555,25);

if (success != true) {

printf("%s\n",listenSocket.lastErrorText());

return;

}

// Get the next incoming connection

// Wait a maximum of 20 seconds (20000 millisec)

CkSocket \*connectedSocket = 0;

connectedSocket = listenSocket.AcceptNextConnection(2000000000);

if (connectedSocket == 0 ) {

printf("%s\n",listenSocket.lastErrorText());

return;

}

\*/

// Set maximum timeouts for reading an writing (in millisec)

//

bool ssl;

ssl = false;

long maxWaitMillisec;

maxWaitMillisec = 20000;

success = connectedSocket.Connect("10.50.19.3",5555,ssl,maxWaitMillisec);

if (success != true) {

printf("%s\n",connectedSocket.lastErrorText());

return;

}

connectedSocket.put\_MaxReadIdleMs(10000);

connectedSocket.put\_MaxSendIdleMs(10000);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Send public key of b

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Generate public key of b

string bpublic = RSApublicb();

// Send a "Hello World!" message to the client:

success = connectedSocket.SendString(bpublic.c\_str());

/\*if (success != true) {

printf("%s\n",connectedSocket->lastErrorText());

delete connectedSocket;

return;

}\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Receive public key of a

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const char \* apublic;

apublic = connectedSocket.receiveString();

/\*if (receivedMsg == 0 ) {

printf("%s\n",socket->lastErrorText());

return;

}\*/

// Close the connection with the server

// Wait a max of 20 seconds (20000 millsec)

//socket.Close(20000);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//

// B,{Nb,{H(Nb)}Kb-1}Ka

//

//

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

H(Na)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

std::string nb = to\_string(rand() % 999999999999999 + 1000000000000000);

std::string B = "I am B";

std::string temp = SHA(nb);

printf("%s\n\n",temp);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{H(Nb)}Kb-1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

string temp2 = RSAprivate(temp);

printf("%s\n\n",temp2);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Concatenation: Nb,{H(Nb)}Kb-1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

std::string concat = nb + "," +temp2;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{Nb, {H(Nb)}Kb-1}Ka

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

string temp3 = RSApublica(concat, apublic);

printf("%s\n\n\n",temp3);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Concatenation: B, {Nb,{H(Nb)}Kb-1}Ka

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

std::string concat2 = B + "," +temp3;

printf("%s\n\n", concat2);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Send: B, {Nb,{H(Nb)}Kb-1}Ka

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Send a "Hello World!" message to the client:

success = connectedSocket.SendString(concat2.c\_str());

/\*if (success != true) {

printf("%s\n",connectedSocket->lastErrorText());

delete connectedSocket;

return;

}\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Receive: A, {Na,{H(Na)}Ka-1}Kb, {Nb}Kab

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const char \* receivedMsg;

receivedMsg = connectedSocket.receiveString();

/\*if (receivedMsg == 0 ) {

printf("%s\n",socket->lastErrorText());

return;

}\*/

//Spliiting String!!

std::vector<std::string> splitted = split(receivedMsg, ',');

std::string test = splitted[1];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Decrypting {Na, {H(Na)}Ka-1}Kb

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

std::string decryptNa = RSAdecryptb(test);

std::vector<std::string> splitted2 = split(decryptNa, ',');

std::string test2 = splitted2[0];

printf("%s", test2);

/\*int i;

for(i = 0; i < bEncryption.length(); i++)

{

temporary = temporary + bEncryprtion[i];

if(bEncryption[i] == "," | bEncryption[i] == "")

{

printf("%s", temporary);

temporary = "";

}

}\*/

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

SocketSend();

getch();

return 0;

}