#include "stdafx.h"

#include "stdlib.h"

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

// 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"

// for the AES Encryption & Hashing

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

// for RSA

#include "../include/CkRsa.h"

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

#include <iostream>

#include <string>

#include <conio.h>

using namespace std;

const char \* publicKey;

const char \* privateKey;

CkRsa rsa;

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

Generate Keys

Generate a 1024-bit key.

Chilkat RSA supports key sizes ranging from 512 bits to 4096 bits.

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

/\*

void generateKey(){

}

\*/

//RSA Public Key of B

string RSApublicb()

{

bool success;

success = rsa.UnlockComponent("T06152014RSA\_XFn8tkbGkQQd");

/\*if (success != true) {

printf("RSA component unlock failed\n");

return;

}\*/

bool success;

success = rsa.GenerateKey(1024);

/\*if (success != true) {

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

return;

}\*/

publicKey = rsa.exportPublicKey();

privateKey = rsa.exportPrivateKey();

/\*

// Start with a new RSA object to demonstrate that all we

// need are the keys previously exported:

CkRsa rsaEncryptor;

// Encrypted output is always binary to encode the encrypted bytes in a printable string

rsaEncryptor.put\_EncodingMode("hex");

// We'll encrypt with the public key and decrypt with the private

// key. It's also possible to do the reverse.

rsaEncryptor.ImportPublicKey(publicKey);

bool usePrivateKey;

usePrivateKey = false;

const char \* encryptedStr;

encryptedStr = rsaEncryptor.encryptStringENC(Hna.c\_str(),usePrivateKey);

\*/

/\*

string result(publicKey);

return result;

\*/

return publicKey;

printf("%s", publicKey);

}

//RSA Private Key

std::string RSAprivate(string Hnb)

{

bool success;

success = rsa.UnlockComponent("T06152014RSA\_XFn8tkbGkQQd");

/\*if (success != true) {

printf("RSA component unlock failed\n");

return;

}\*/

// Start with a new RSA object to demonstrate that all we

// need are the keys previously exported:

CkRsa rsaEncryptor;

// Encrypted output is always binary. In this case, we want

// to encode the encrypted bytes in a printable string.

// Our choices are "hex", "base64", "url", "quoted-printable".

rsaEncryptor.put\_EncodingMode("hex");

// We'll encrypt with the public key and decrypt with the private

// key. It's also possible to do the reverse.

rsaEncryptor.ImportPrivateKey(privateKey);

bool usePrivateKey;

usePrivateKey = true;

const char \* encryptedStr;

encryptedStr = rsaEncryptor.encryptStringENC(Hnb.c\_str(),usePrivateKey);

return publicKey;

}

//RSA Public Key of A

string RSApublica(string encrypt, const char \* pub)

{

bool success;

success = rsa.UnlockComponent("T06152014RSA\_XFn8tkbGkQQd");

/\*if (success != true) {

printf("RSA component unlock failed\n");

return;

}\*/

// Start with a new RSA object to demonstrate that all we

// need are the keys previously exported:

CkRsa rsaEncryptor;

// Encrypted output is always binary. In this case, we want

// to encode the encrypted bytes in a printable string.

// Our choices are "hex", "base64", "url", "quoted-printable".

rsaEncryptor.put\_EncodingMode("hex");

// We'll encrypt with the public key and decrypt with the private

// key. It's also possible to do the reverse.

rsaEncryptor.ImportPublicKey(publicKey);

bool usePrivateKey;

usePrivateKey = false;

const char \* encryptedStr;

encryptedStr = rsaEncryptor.encryptStringENC(encrypt.c\_str(),usePrivateKey);

string result(encryptedStr);

return result;

}

string RSAdecryptb(string decr)

{

bool success;

success = rsa.UnlockComponent("T06152014RSA\_XFn8tkbGkQQd");

/\*if (success != true) {

printf("RSA component unlock failed\n");

return;

}\*/

/\*if (success != true) {

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

return;

}\*/

// Keys are exported in XML format:

//const char \* publicKey;

// Now decrypt:

CkRsa rsaDecryptor;

rsaDecryptor.put\_EncodingMode("hex");

bool usePrivateKey;

usePrivateKey = false;

const char \* decryptedStr;

decryptedStr = rsaDecryptor.decryptStringENC(decr.c\_str(), usePrivateKey);

string result(decryptedStr);

return result;

}