|  |  |  |
| --- | --- | --- |
| Лабораторная работа #5  «Программирование SSL-клиента»  Вариант #5 | Выполнил | Ноздренков С.В. |
| Группа | ЭВМ-1.Н |
| Проверил | Жариков Д. Н. |
| Подпись |  |

# Цель работы

Цель работы: Изучение программного интерфейса OpenSSL. Изучение порядка действий, выполняемых клиентом SSL.

# Задание

Разработать две программы: клиент и сервер, позволяющие получать список файлов указанного каталога на компьютере, где функционирует сервер, переименовывать, копировать в другой каталог или удалять указанный файл, запускать программу на выполнение, выполнять перезагрузку компьютера.

Отличия от 1/2 лабораторной работы:

# Crypto.hpp

#include <iostream>

#include <string>

#include <openssl/rsa.h>

#include <openssl/pem.h>

#include <openssl/x509.h>

using namespace std;

void dbg(const string &s) { cout << s << endl; }

/\*\*

@brief Класс шифрования данных

\*/

class crypto\_t

{

string BIO\_to\_string(BIO \*bio)

{

int len = BIO\_pending(bio);

char \*buf = new char[len + 1];

BIO\_read(bio, buf, len);

string res = string(buf);

delete[] buf;

return res;

}

BIO \* string\_to\_BIO(const string &s)

{

BIO \*bio = NULL;

const int len = s.size();

char \*buf = new char[len + 1];

memset(buf, 0, len + 1);

memcpy(buf, s.c\_str(), len);

BIO\_write(bio, buf, len);

delete[] buf;

return bio;

}

string public\_key;

string private\_key;

public:

/\*\*

@brief Инициализация приватного и публичного ключей

@param name - имена ключей

\*/

crypto\_t()

{

// генерируем ключи

RSA \*rsa = RSA\_generate\_key(2048, RSA\_F4, 0, 0);

// сохраняем публичный ключ

BIO \*bio = BIO\_new(BIO\_s\_mem());

PEM\_write\_bio\_RSAPublicKey(bio, rsa);

public\_key = BIO\_to\_string(bio);

BIO\_free\_all(bio);

// сохраняем приватный ключ

bio = BIO\_new(BIO\_s\_mem());

PEM\_write\_bio\_RSAPrivateKey(bio, rsa, 0, 0, 0, 0, 0);

private\_key = BIO\_to\_string(bio);

BIO\_free\_all(bio);

// очищаем память

RSA\_free(rsa);

}

/\*\*

@brief Возвращает публичный свой ключ

\*/

string get\_public\_key()

{

return public\_key;

}

/\*\*

@brief Расшифровывает сообщение msg своим приватным ключем

@param msg - зашифрованное сообщение

\*/

string decrypt(const string &msg)

{

BIO \*bio = string\_to\_BIO(private\_key);

RSA \*priv\_key = PEM\_read\_bio\_RSAPrivateKey(bio, 0, 0, 0);

string ans;

int len = RSA\_size(priv\_key);

for (int i = 0; i < msg.size(); i += len)

{

string s = msg.substr(i, len);

unsigned char \*from = new unsigned char[len + 1];

memset(from, 0, len + 1);

memcpy(from, s.c\_str(), len);

unsigned char \*to = NULL;

int olen = RSA\_private\_decrypt(len, from, to, priv\_key, RSA\_PKCS1\_PADDING);

if (olen < 0)

{

delete[] from;

delete[] to;

return string();

}

ans += string(reinterpret\_cast<char \*>(to));

delete[] from;

delete[] to;

}

BIO\_free\_all(bio);

RSA\_free(priv\_key);

return ans;

}

/\*\*

@brief Шифрует сообщение msg при помощи публичного ключа public\_key

@param msg - сообщение

@param public\_key - публичный ключ

\*/

string encrypt(const string &msg, const string &public\_key)

{

BIO \*bio = string\_to\_BIO(public\_key);

RSA \*pub\_key = PEM\_read\_bio\_RSAPublicKey(bio, 0, 0, 0);

string ans;

int len = RSA\_size(pub\_key) - 11;

for (int i = 0; i < msg.size(); i += len - 1)

{

string s = msg.substr(i, len - 1);

unsigned char \*from = new unsigned char[len];

memset(from, 0, len);

memcpy(from, s.c\_str(), s.size());

unsigned char \*to = NULL;

int olen = RSA\_public\_encrypt(len, from, to, pub\_key, RSA\_PKCS1\_PADDING);

if (olen != RSA\_size(pub\_key))

{

delete[] from;

delete[] to;

return string();

}

ans += string(reinterpret\_cast<char \*>(to));

delete[] from;

delete[] to;

}

BIO\_free\_all(bio);

RSA\_free(pub\_key);

return ans;

}

};