Un dibujo de una cara feliz

Descripción generada automáticamente con confianza baja

**Proyecto final**

**Libsodium**

**Seguridad en redes**

**Profesor:** DOMINGUEZ PEREZ, LUIS JULIAN

**Autor:** Iñaki Sebastián Orozco García

**Matrícula**: IS719276

**Fecha**: /2021

# Explicación del código:

#include <stdio.h>

#include <sodium.h>

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

#define DEBUG 0

using namespace std;

constexpr auto CHUNK\_SIZE = 4096;

Hago estos imports para poder definir esa función, esa función sirve para poder fácilmente darle el path a un archivo que tiene texto dentro y regresa un string que contiene todo ese archivo.

using std::cout; using std::cerr;

using std::endl; using std::string;

string readFileIntoString(const string& path) {

    struct stat sb {};

    string res;

    FILE\* input\_file = fopen(path.c\_str(), "r");

    if (input\_file == nullptr) {

        perror("fopen");

    }

    stat(path.c\_str(), &sb);

    res.resize(sb.st\_size);

    fread(const\_cast<char\*>(res.data()), sb.st\_size, 1, input\_file);

    fclose(input\_file);

    return res;

}

Estas dos funciones encriptar y desencriptar son las mismas de la entrega pasada, va encriptando carácter por carácter hasta que acaba utilizando una llave que recibe la función.

static int

encriptar(const char\* archivo\_resultante, const char\* archivo\_original,

    const unsigned char key[crypto\_secretstream\_xchacha20poly1305\_KEYBYTES])

{

    unsigned char  buf\_in[CHUNK\_SIZE];

    unsigned char  buf\_out[CHUNK\_SIZE + crypto\_secretstream\_xchacha20poly1305\_ABYTES];

    unsigned char  header[crypto\_secretstream\_xchacha20poly1305\_HEADERBYTES];

    crypto\_secretstream\_xchacha20poly1305\_state st;

    FILE\* fp\_t, \* fp\_s;

    unsigned long long out\_len;

    size\_t         rlen;

    int            eof;

    unsigned char  tag;

    //lo abre con rb para poder leer archivos en general.

    fp\_s = fopen(archivo\_original, "rb");

    //lo abre con wb para poder escribir en el archivo.

    fp\_t = fopen(archivo\_resultante, "wb");

    //Esta línea inicializa la encripción basándose en una llave secreta y un header, la llave no se volverá a usar.

    crypto\_secretstream\_xchacha20poly1305\_init\_push(&st, header, key);

    fwrite(header, 1, sizeof header, fp\_t);

    //Mientras el archivo original no haya terminado se va a ejecutar este código que encripta cada caracter y lo guarda en el archivo resultante (fp\_t)

    do {

        //largo del archivo original

        rlen = fread(buf\_in, 1, sizeof buf\_in, fp\_s);

        eof = feof(fp\_s);

        tag = eof ? crypto\_secretstream\_xchacha20poly1305\_TAG\_FINAL : 0;

        //Encripta el mensaje con el estado &st que se inicializó arriba, de largo rlen y el tag FINAL

        crypto\_secretstream\_xchacha20poly1305\_push(&st, buf\_out, &out\_len, buf\_in, rlen,

            NULL, 0, tag);

        fwrite(buf\_out, 1, (size\_t)out\_len, fp\_t);

    } while (!eof);

    //Cierra los dos archivos al terminar la encripción y regresa 0

    fclose(fp\_t);

    fclose(fp\_s);

    return 0;

}

static int

desencriptar(const char\* archivo\_resultante, const char\* archivo\_original,

    const unsigned char key[crypto\_secretstream\_xchacha20poly1305\_KEYBYTES])

{

    unsigned char  buf\_in[CHUNK\_SIZE + crypto\_secretstream\_xchacha20poly1305\_ABYTES];

    unsigned char  buf\_out[CHUNK\_SIZE];

    unsigned char  header[crypto\_secretstream\_xchacha20poly1305\_HEADERBYTES];

    crypto\_secretstream\_xchacha20poly1305\_state st;

    FILE\* fp\_t, \* fp\_s;

    unsigned long long out\_len;

    size\_t         rlen;

    int            eof;

    int            ret = -1;

    unsigned char  tag;

    //Abre el archivo original a desencriptar y crea un archivo resultante donde estará el resultado.

    fp\_s = fopen(archivo\_original, "rb");

    fp\_t = fopen(archivo\_resultante, "wb");

    fread(header, 1, sizeof header, fp\_s);

    //Si el pull inicial del state no es 0 por falta de header o incompleto, va a ret que cierra los dos archivos.

    if (crypto\_secretstream\_xchacha20poly1305\_init\_pull(&st, header, key) != 0) {

        goto ret;

    }

    //Mientras no termine el documento de desencripción no termina el while.

    do {

        rlen = fread(buf\_in, 1, sizeof buf\_in, fp\_s);

        eof = feof(fp\_s);

        //Si no regresa 0 la desencripción (está corrupto el archivo) va a ret.

        if (crypto\_secretstream\_xchacha20poly1305\_pull(&st, buf\_out, &out\_len, &tag,

            buf\_in, rlen, NULL, 0) != 0) {

            goto ret;

        }

        //Si termina el archivo antes de que se espere salta a ret.

        if (tag == crypto\_secretstream\_xchacha20poly1305\_TAG\_FINAL && !eof) {

            goto ret;

        }

        fwrite(buf\_out, 1, (size\_t)out\_len, fp\_t);

    } while (!eof);

    ret = 0;

ret:

    fclose(fp\_t);

    fclose(fp\_s);

    return ret;

}

Esta función sirve para firmar un archivo y guardar el contenido de ese archivo además de la firma en un documento nuevo, se usa una llave privada y una publica para firmarlo.

static int

firmar(const char\* archivo\_resultante, const char\* archivo\_original,

    const unsigned char publicKey[crypto\_sign\_PUBLICKEYBYTES], const unsigned char privateKey[crypto\_sign\_SECRETKEYBYTES])

{

Aqui uso la función que defini arriba para convertir a string el contenido del archivo original y luego copio todo ese contenido a una variable llamada mensaje que es un arreglo de char sin signo, es necesario así para poder usarlo en la función crypto\_sign.

    string file\_contents;

    file\_contents = readFileIntoString(archivo\_original);

    int file\_length = file\_contents.length();

    //convertimos a unsigned char\* el string

    unsigned char\* mensaje = new unsigned char[file\_length+1]();

    std::copy(file\_contents.begin(), file\_contents.end(), mensaje);

    mensaje[file\_contents.length()] = 0;

    //reservamos la memoria del mensaje firmado

    unsigned char \*signed\_message = new unsigned char[crypto\_sign\_BYTES + file\_length];

    unsigned long long signed\_message\_len;

Reservo también memoria para la variable signed message que es en la que estará el contenido del archivo con la firma y nos lo regresa la función de libsodium.

    crypto\_sign(signed\_message, &signed\_message\_len, mensaje, file\_length, privateKey);

Reservo también memoria para la variable unsignedsigned message que se necesita para comprobar que el mensaje se firmó correctamente.

    unsigned char \*unsigned\_message = new unsigned char[file\_length];

    unsigned long long unsigned\_message\_len;

    if (crypto\_sign\_open(unsigned\_message, &unsigned\_message\_len,

        signed\_message, signed\_message\_len, publicKey) != 0) {

        //SE FIRMO MAL

        return 1;

    }

Creo un arreglo de resultados que mide lo mismo que el archivo firmado para pasarle todo el contenido del signed\_message.

    char\* result = new char[crypto\_sign\_BYTES + file\_length];

    for (unsigned int i= 0; i < crypto\_sign\_BYTES + file\_length; i++) {

        result[i] = signed\_message[i];

    }

Estas son impresiones para debugear, no son necesarias para que funcione el código y solo funcionan si la variable global constante vale 1.

    if (DEBUG) {

        cout << "file\_contents: " << file\_contents << endl;

        cout << endl;

        cout << "crypto\_sign\_BYTES: " << crypto\_sign\_BYTES << endl;

        cout << endl;

        cout << "file\_length: " << file\_length << endl;

        cout << endl;

        cout << "crypto\_sign\_BYTES + file\_length: " << crypto\_sign\_BYTES + file\_length << endl;

        cout << endl;

        for (int i = 0; i < file\_length; i++) cout << mensaje[i]; cout << endl;

        cout << "mensaje length: " << strlen((char\*)mensaje) << endl;

        cout << endl;

        cout << "unsigned\_Message length: " << strlen((char\*)unsigned\_message) << endl;

        cout << endl;

        for (int i = 0; i < crypto\_sign\_BYTES + file\_length; i++) cout << signed\_message[i]; cout << endl;

        cout << "signed\_message length: " << strlen((char\*)signed\_message) << endl;

        cout << endl;

        cout << "result length: " << strlen((char\*)result) << endl;

        cout << endl;

    }

Con fstream guardo en la dirección del archivo\_resultante el documento firmado, y le digo que solo guarde los bits que mide el archivo mas la firma por que tuve un problemón por que guardaba basura y al final encontré esta hermosa solución.

    ofstream ofs;

    ofs.open(archivo\_resultante, ios::app);

    ofs.write((char\*)result, crypto\_sign\_BYTES + file\_length);

    ofs.close();

    //borramos las variables que reservamos

    delete[] signed\_message;

    delete[] unsigned\_message;

    delete[] mensaje;

    delete[] result;

    return 0;

}

Esta es la función para verificar una firma, solo recibe el archivo a verificar y una llave pública que es par de la privada con la que se firmó el archivo.

static int

verificar(const char\* archivo, const unsigned char publicKey[crypto\_sign\_PUBLICKEYBYTES])

{

    string file\_contents;

    file\_contents = readFileIntoString(archivo);

    unsigned long long file\_length = file\_contents.length();

    //si está mal creado el archivo firmado ni tiene caso que se haga nada más

    if (file\_length < crypto\_sign\_BYTES) {

        return 1;

    }

Divido en dos variables la firma y el contenido del archivo para poder validarlo ya que libsodium solo tiene la función crypto sign verify detached que lo verifica en partes.

    string signature = file\_contents.substr(0, crypto\_sign\_BYTES);

    file\_contents = file\_contents.substr(crypto\_sign\_BYTES, file\_length);

    //ahora que separamos el mensaje del archivo reajustamos el largo

    file\_length = file\_contents.length();

    //convertimos a unsigned char\* el string

    unsigned char\* mensaje = new unsigned char[file\_length + 1];

    std::copy(file\_contents.begin(), file\_contents.end(), mensaje);

    mensaje[file\_contents.length()] = 0;

    //reservamos la memoria de la firma y la convertimos a char\*

    unsigned char\* sig = new unsigned char[crypto\_sign\_BYTES + 1];

    std::copy(signature.begin(), signature.end(), sig);

    sig[signature.length()] = 0;

Copio eso en variables unsigned char\* por que es el tipo de variable que necesita la función de libsodium.

    if (DEBUG) {

        cout << "file\_contents: " << file\_contents << endl;

        cout << endl;

        cout << "crypto\_sign\_BYTES: " << crypto\_sign\_BYTES << endl;

        cout << endl;

        cout << "file\_length: " << file\_length << endl;

        cout << endl;

        cout << "crypto\_sign\_BYTES + file\_length: " << crypto\_sign\_BYTES + file\_length << endl;

        cout << endl;

        cout << "mensaje: " << mensaje << endl;

        cout << endl;

    }

Verifica que la firma es correcta, si no es imprime no es válida y regresa 1.

    if (crypto\_sign\_verify\_detached(sig, mensaje, file\_length, publicKey) != 0) {

        cout << "La firma no es válida";

        return 1;

    }

    cout << "La firma es válida";

    //borramos las variables que reservamos

    delete[] mensaje;

    delete[] sig;

    return 0;

}

Esta función sirve para escribir en un archivo una llave secreta. Solo recibe el archivo y la llave secreta.

static int

escribirSK(const char\* archivo, const unsigned char key[crypto\_secretstream\_xchacha20poly1305\_KEYBYTES])

{

Guarda en ese archivo casteando a char la llave (que es unsigned char\*) y necesitamos que sea solo char\*.

    FILE\* keyFile;

    keyFile = fopen(archivo, "w");

    if (keyFile != NULL)

    {

        fputs(reinterpret\_cast<const char\*>(key), keyFile);

        fclose(keyFile);

    }

    else {

        return 1;

    }

    return 0;

}

Hago lo mismo con esta función pero esta guarda dos llaves, el par de pública y privada y sirve exactamente igual que la de arriba.

static int

escribirKeys(const char\* archivo\_public, const char\* archivo\_private, const unsigned char publicKey[crypto\_sign\_PUBLICKEYBYTES], const unsigned char privateKey[crypto\_sign\_SECRETKEYBYTES])

{

    FILE\* publicKeyFile;

    publicKeyFile = fopen(archivo\_public, "w");

    if (publicKeyFile != NULL)

    {

        fputs(reinterpret\_cast<const char\*>(publicKey), publicKeyFile);

        fclose(publicKeyFile);

    }

    else {

        return 1;

    }

    FILE\* privateKeyFile;

    privateKeyFile = fopen(archivo\_private, "w");

    if (privateKeyFile != NULL)

    {

        fputs(reinterpret\_cast<const char\*>(privateKey), privateKeyFile);

        fclose(privateKeyFile);

    }

    else {

        return 1;

    }

    return 0;

}

Esta función agrega al programa una llave secreta de un archivo usando la función que definí hasta arriba de read fileIntoString, luego copia a una variable unsigned char\* key y es la que se guarda globalmente en el programa.

static int

extraerSK(const char\* archivo, unsigned char key[crypto\_secretstream\_xchacha20poly1305\_KEYBYTES]){

    string file\_contents;

    file\_contents = readFileIntoString(archivo);

    std::copy(file\_contents.begin(), file\_contents.end(), key);

    key[file\_contents.length()] = 0;

    return 0;

}

Igual que la función de arriba y abajo, esta es para la llave pública.

static int

extraerPublicKey(const char\* archivo, unsigned char publicKey[crypto\_sign\_PUBLICKEYBYTES]) {

    string file\_contents;

    file\_contents = readFileIntoString(archivo);

    std::copy(file\_contents.begin(), file\_contents.end(), publicKey);

    publicKey[file\_contents.length()] = 0;

    return 0;

}

Igual que la función de arriba, esta es para la llave privada.

static int

extraerPrivateKey(const char\* archivo, unsigned char privateKey[crypto\_sign\_SECRETKEYBYTES]) {

    string file\_contents;

    file\_contents = readFileIntoString(archivo);

    std::copy(file\_contents.begin(), file\_contents.end(), privateKey);

    privateKey[file\_contents.length()] = 0;

    return 0;

}

Esta función es la que sirve para imprimir el menú, no hace nada más.

void showChoices()

{

    cout << "MENU" << endl;

    cout << "1: Generación de claves " << endl;

    cout << "2: Recuperación de claves" << endl;

    cout << "3: Cifrado de archivos " << endl;

    cout << "4: Descifrado de archivos " << endl;

    cout << "5: Firma de archivos " << endl;

    cout << "6: Verificación de firma de archivos" << endl;

    cout << "0: Salir" << endl;

    cout << "Ingresa tu elección :";

}

int

main(void)

{

    //Si no se puede inicializar la librería de libsodium regresa error

    if (sodium\_init() != 0) {

        return 1;

    }

Declaramos las llaves desde que se inicia el programa, si el usuario carga otras llaves se sobreescriben. Estas llaves funcionan de manera global en el programa y son las que se llaman en las funciones dentro del switchcase. También son las que se exportan si el usuario decide así a un archivo para posteriormente volver a cargarlas en otra “sesión”.

    //Se declara la llave secreta para cifrar/descifrar

    unsigned char secretKey[crypto\_secretstream\_xchacha20poly1305\_KEYBYTES];

    crypto\_secretstream\_xchacha20poly1305\_keygen(secretKey);

    //Se declara la llave pública

    unsigned char publicKey[crypto\_sign\_PUBLICKEYBYTES];

    //Se declara la llave privada

    unsigned char privateKey[crypto\_sign\_SECRETKEYBYTES];

    //Se guarda el par de llaves

    crypto\_sign\_keypair(publicKey, privateKey);

    string a, b, c;

    char origin[300], destiny[300];

    int choice, eleccion;

Inicia el ciclo while que se repite mientras el usuario no telcee 0.

    do

    {

        showChoices();

        cin >> choice;

        switch (choice)

        {

        case 0:

            break;

En este caso da opción de llave secreta o llave pública y llama la función de escribirSK y escribir Keys.

        case 1:

            do {

                cout << "1: Obtener una llave secreta " << endl;

                cout << "2: Obtener una llave pública y privada " << endl;

                cout << "0: Salir " << endl;

                cout << "Ingresa tu elección :";

                cin >> eleccion;

                switch (eleccion) {

                case 0:

                    break;

                case 1:

                    cout << "Escriba la dirección donde se guardará la llave: ";

                    cin >> a;

                    strcpy(origin, a.c\_str());

                    if (escribirSK(origin, secretKey) != 0) {

                        return 1;

                    }

                    break;

                case 2:

                    cout << "Escriba la dirección donde se guardará la llave pública: ";

                    cin >> a;

                    cout << "Escriba la dirección donde se guardará la llave privada: ";

                    cin >> b;

                    strcpy(origin, a.c\_str());

                    strcpy(destiny, b.c\_str());

                    if (escribirKeys(origin, destiny, publicKey, privateKey) != 0) {

                        return 1;

                    }

                    break;

                default:

                    cout << "Invalid input" << endl;

                }

            } while (eleccion != 0);

            break;

En este caso pregunta para extraer llaves al programa desde un archivo y llama a las funciones extraerSK y extreaer public key y private key.

case 2:

            do {

                cout << "1: Extraer una llave secreta " << endl;

                cout << "2: Extraer una llave pública y privada " << endl;

                cout << "0: Salir " << endl;

                cout << "Ingresa tu elección :";

                cin >> eleccion;

                switch (eleccion) {

                case 0:

                    break;

                case 1:

                    cout << "Escriba la dirección donde se guardará la llave: ";

                    cin >> a;

                    strcpy(origin, a.c\_str());

                    if (extraerSK(origin, secretKey) != 0) {

                        return 1;

                    }

                    break;

                case 2:

                    cout << "Escriba la dirección donde se guardará la llave pública: ";

                    cin >> a;

                    cout << "Escriba la dirección donde se guardará la llave privada: ";

                    cin >> b;

                    strcpy(origin, a.c\_str());

                    strcpy(destiny, b.c\_str());

                    if (extraerPublicKey(origin, publicKey) != 0) {

                        return 1;

                    }

                    if (extraerPrivateKey(destiny, privateKey) != 0) {

                        return 1;

                    }

                    break;

                default:

                    cout << "Invalid input" << endl;

                }

            } while (eleccion != 0);

            break;

La tercera opción es para encriptar, pide dos entradas que son casteadas a char\* para poder usarlas como direcciones, si la función encriptar no regresa 0 regresa 1 (error).

        case 3:

            cout << "Escriba la dirección del archivo a encriptar: ";

            cin >> a;

            cout << "Escriba la dirección del archivo resultante: ";

            cin >> b;

            strcpy(origin, a.c\_str());

            strcpy(destiny, b.c\_str());

            if (encriptar(destiny, origin, secretKey) != 0) {

                return 1;

            }

            break;

Ahora para desencriptar, es igual al case de arriba nada más que este llama a desencriptar.

        case 4:

            cout << "Escriba la dirección del archivo encriptado: ";

            cin >> a;

            cout << "Escriba la dirección del archivo desencriptado: ";

            cin >> b;

            strcpy(origin, a.c\_str());

            strcpy(destiny, b.c\_str());

            if (desencriptar(destiny, origin, secretKey) != 0) {

                return 1;

            }

            break;

Para firmar, pide los dos archivos y ejecuta la función firmar, igual a los cases de arriba pero diferente función.

        case 5:

            cout << "Escriba la dirección del archivo a firmar: ";

            cin >> a;

            cout << "Esciba la dirección del archivo firmado: ";

            cin >> b;

            strcpy(origin, a.c\_str());

            strcpy(destiny, b.c\_str());

            if (firmar(destiny, origin, publicKey, privateKey) != 0) {

                return 1;

            }

            break;

Este case es parecido a los de arriba pero solo pide un input ya que solo se necesita verificar el archivo.

        case 6:

            cout << "Escriba la dirección del archivo a verificar: ";

            cin >> a;

            strcpy(origin, a.c\_str());

            if (verificar(origin, publicKey) != 0) {

                return 1;

            }

            break;

        default:

            cout << "Invalid input" << endl;

        }

    } while (choice != 0);

    //Si todo funcionó, regresa código de éxito.

    return 0;

}

# Conclusión

Este proyecto me gustó mucho, es mi segunda vez trabajando con c++ ya que la primera fue en la tarea pasada, nunca lo había usado pero me gustó y aunque se parece mucho a c si tiene sus diferencias. Igual fue un reto ya que tuve muchos problemas con la alocación de memoria y con los punteros, al final me siento muy satisfecho ya que con ayuda del profesor pude lograr que funcionara el programa.