C

Cifrador-Descifrador

Creación de un cifrador descifrador en lenguaje C.

Autor:
Alejandro Fisac

4 de agosto de 2024

Índice general

Capítulo 1

Creación de un cifrador/descifrador:

1.1. Planteamiento:

Para poder desarrollar esta practica se utilizaran librerías capaces de trabajar con directorios y otras que sean capaces dada una clave cifrar un archivo.

1.1.1. Librerías de directorios y archivos:

Al realizarse en sistemas linux, para poder operar con directorios y ficheros, requerimos del paquete **dirent.h**. La documentación oficial se puede encontrar en MAN Docs.

Funcionamiento:

En primer lugar, debemos saber cómo ver que ficheros y directorios hay en una dirección.

```
1  #include <dirent.h>
2  #include <stdio.h>
3
4  int main() {
5    DIR *dir;
6    struct dirent *entry;
7    dir = opendir("/");
8    if (dir == NULL) {
9      perror("opendir"); /*Devuelve un error por la salida estandar.*/
10     return -1;
11   }
12   while ((entry = readdir(dir)) != NULL) {
13     printf("Nombre: %s\n", entry->d_name);
14     printf("Tipo de archivo: %d\n", entry->d_type);
15   }
16   closedir(dir);
17   return 0;
18 }
```

Listing 1.1: Sacar los ficheros y directorios que hay en una determinada carpeta

Para saber que tipo de archivo es cada cosa reconocida debemos fijarnos en la siguiente documentación GNU Docs.

6	DT_BLK	block device: Manejo de datos en bloques (p.e. CD-ROM, particiones de disco duro)
2	DT_CHR	*Character Device*: Dispositivo que maneja flujo de caracteres. Permite lectura y escritura de forma secuencial. (p.e. un terminal)
4	DT_DIR	*Directory*: Se trata de un directorio.
1	DT_FIFO	*FIFO/Pipe*: Archivo especial de comunicación entre procesos.
10	DT_LNK	*Symbolic Link*: Contiene la referencia de otro archivo.
8	DT_REG	*Regular File*: Archivo de índole regular. (p.e. documento de texto)
12	DT_SOCK	*Sockets*: Comunicación entre red de procesos. (p.e. TCP, UDP)
0	DT_UNKNOWN	*Unknown File*: Archivo que no puede ser identificado. (p.e. archivos corruptos o de otro estilo)

Para esta práctica y el sistema utilizado debemos partir del directorio / que es el directorio raiz. De manera recursiva debemos ir analizando si el fichero analizado es un directorio o no para saber que hacer con él.

```
(Salida omitida ...)
Nombre: lib64
Tipo de archivo: 10
Nombre: snap
Tipo de archivo: 4
Nombre: bin
Tipo de archivo: 10
Nombre: mnt
Tipo de archivo: 4
Nombre: dev
Tipo de archivo: 4
Nombre: usr
Tipo de archivo: 4
Nombre: proc
Tipo de archivo: 4
Nombre: var
Tipo de archivo: 4
Nombre: root
Tipo de archivo: 4
Nombre: home
Tipo de archivo: 4
Nombre: etc
Tipo de archivo: 4
Nombre: lib
Tipo de archivo: 10
Nombre: media
Tipo de archivo: 4
Nombre: srv
Tipo de archivo: 4
Nombre: run
Tipo de archivo: 4
Nombre: libx32
Tipo de archivo: 10
Nombre: .
Tipo de archivo: 4
Nombre: lib32
Tipo de archivo: 10
Nombre: boot
Tipo de archivo: 4
Nombre: sys
Tipo de archivo: 4
Nombre: lost+found
Tipo de archivo: 4
(Salida omitida ...)
```

Listing 1.2: Salida del programa C.

Capítulo 2

OpenSSL:

Para el cifrado vamos a utilizar un sistema de cifrado clave público-privado, para ello voy a generar una combinación de clave público-privado para este ejercicio.

```
1 openssl genpkey -algorithm RSA -out claveprivada.key -pkeyopt
     rsa_keygen_bits:4096
2 openssl rsa -in claveprivada.key -pubout -out clavepublica.pem # Generamos
     la clave publica en base a la privada.
```

Listing 2.1: Generación clave público-privada

Antes de investigar como realizarlo en C, voy a cifrar un archivo por línea de comandos con la clave pública generada.

```
1 openssl rsautl -encrypt -pubin -inkey clavepublica.pem -in prueba.txt -out
     archivo_cifrado.enc
```

Listing 2.2: "Generación clave público-privada"

```
1
2 openssl pkeyutl -encrypt -pubin -inkey clavepublica.pem -in prueba.txt -
     out archivo_cifrado.enc
3 openssl pkeyutl -decrypt -inkey claveprivada.pem -in archivo_cifrado.enc -
     out archivo_desencriptado.txt
4
5 # En base al padding:
6 openssl rsautl -encrypt -inkey public_key.pem -pubin -in plaintext.txt -
     out encrypted_data.bin -pkcs
7 openssl rsautl -decrypt -inkey private_key.pem -in encrypted_data.bin -out
     decrypted_text.txt -pkcs
```

Listing 2.3: 'Encriptación y desencriptación por comandos.'

Resultante queda:

```
Desencriptado:
    Hola Mundo
    Este es el contenido sin cifrar.
```

Encriptado:

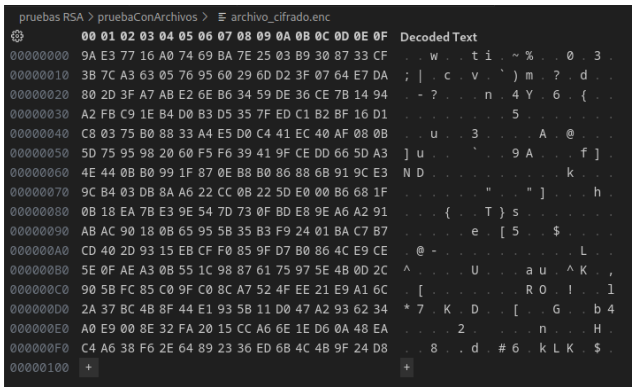

Figura 2.1: Texto Encriptado

2.0.1. Preparación del texto:

Función conversor de un byte a binario:

```c
void byteToBinary(unsigned char byte, char *binaryStr) {
  /*Debemos convertir cada byte del archivo a binario para posteriormente
     aplicarle la clave publica.*/
  int i;
  for (i = 7; i >= 0; i--) {
    binaryStr[7 - i] = (byte & (1 << i)) ? '1' : '0';
  }
  binaryStr[8] = '\0';
}
```

Listing 2.4: Conversión de byte a binario

```c
unsigned char binaryToByte(const char *binaryString) {
  unsigned char byteValue = 0;
  unsigned int i = 0;
  for (i = 0; i < 8; ++i) {
    if (binaryString[i] == '1') {
      byteValue |= (1 << (7 - i));
    } else if (binaryString[i] != '0') {
      fprintf(stderr,
              "Error: Cadena binaria contiene caracteres no validos.\n");
      exit(1);
    }
  }
  return byteValue;
}
```

Listing 2.5: Conversión de binario a Byte

Esta función en base a una cadena binaria lo transforma en los caracteres correspondientes. Ejemplo práctico de Conversor a binario:

```
Binario:
01001000011011110110110001100001001000000000101001001101011101010 1
1011100110010001101111001000000
Caracteres:
Hola
Mundo
```

Listing 2.6: Ejemplo de funcionamiento del conversor

2.0.2. Eliminación total de un fichero:

Para poder eliminar un archivo vamos a hacer uso de la librería *stdio.h*. La función que vamos a utilizar es:

```
1 int remove(const char *pathname);
```

Listing 2.7: Función para eliminar archivos

Esta función se puede utilizar tanto para directorios como para ficheros.

2.1. Cifrado y Descifrado:

Para cifrar y descifrar los documentos vamos a hacer uso de las librerias *openssl* y *crypto*. Antes de profundizar en las mismas vamos a hablar del planteamiento del cifrado y descifrado.

Para este ejercicio lo que se va a hacer es utilizar el cifrado **RSA** y el de **clave simétrica**. Para ello vamos a hablar de las funciones más importantes de *OpenSSL*.

2.1.1. Métodos y funciones importantes:

Inicialización y Configuración:

```
1 /*Cargamos funciones necesarias para el funcionamiento del ssl */
2 void inicializarEntorno(){
3     OPENSSL_init_crypto(0, NULL);
4     OPENSSL_init_ssl(0, NULL);
5 }
```

Listing 2.8: Init de la librería SSL

El primeroo inicializa el protocolo criptográfico mientras el segundo inicializa los protocolos SSL/TLS.

Generación de claves:

```
1 RSA_new(); /*Devuelve un puntero al RSA vacio*/
2 RSA_generate_key_ex(RSA *rsa, int bits, BIGNUM *e, BN_GENCB *cb);
3 EVP_PKEY_new(); /*Puntero a un EVP_Key vac a*/
4 EVP_PKEY_assign_RSA(EVP_PKEY *pkey, RSA *key); /*Devuelve 1 en caso de
    haberse realizado correctamente.*/
```

Listing 2.9: Init de la librería SSL

- RSA_new: crea un objeto RSA vacio.
- RSA_generate_key_ex: Genera un par de claves RSA. *rsa: es el lugar donde se almacenará la clave generada. Bits: es la cantidad de bits para la clave RSA. e: Exponente para la clave RSA. cb: Es un callback para ver el proceso de generación de la clave RSA.
- EVP_PKEY_new: Crea un *Envelope PKey* vacía.
- EVP_PKEY_assign_RSA: pkey: Estructura EVP-Pkey donde se asignará la clave RSA. key: La clave RSA a asignar. Retorna 1 si se realiza de manera correcta.

Cifrado:

```
1 EVP_CIPHER_CTX *EVP_CIPHER_CTX_new();
2 int EVP_EncryptInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type, ENGINE
      *impl, unsigned char *key, unsigned char *iv);
3 int EVP_EncryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl,
    unsigned char *in, int inl);
4 int EVP_EncryptFinal_ex(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl
    );
5
6 /*Mediante clave p blica:*/
7 int RSA_public_encrypt(int flen, const unsigned char *from, unsigned char
    *to, RSA *rsa, int padding);
8 /*Ejemplo de padding: RSA_PKCS1_OAEP_PADDING*/
```

Listing 2.10: Cifrado

- EVP_CIPHER_CTX_new: Crea un contexto de cifrado vacío.
- EVP_EncryptInit_ex: ctx: Contexto de cifrado, type: algoritmo de cifrado a utilizar, impl: opcional implementación del algoritmo de cifrado, key: Clave de cifrado, iv: Vector de inicialización. Devuelve 1 en caso de realizarse correctamente. Es el encargado de **inicializar el contexto de cifrado.**
- EVP_EncryptUpdate: ctx: Contexto de cifrado, out: buffer donde se almacena el cifrado, outl: longitud del out, in: los datos a cifrar, intl: longitud de los datos a cifrar de entrada. Es el encargado de cifrar. Devuelve 1 si se realiza correctamente, 0 en otro caso.
- EVP_EncryptFinal_ex: ctx: Contexto de cifrado, out:buffer donde se almacenará el texto cifrado final, outl: longitud del buffer out.

```
1
2  /* Crear un nuevo contexto de cifrado*/
3  ctx = EVP_CIPHER_CTX_new();
4
5  /* Inicializar el contexto de cifrado para cifrar con AES-256-CBC */
6  EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv);
7
8  /* Cifrar los datos*/
9  EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, strlen(plaintext));
10
11 /* Finalizar el cifrado*/
12 EVP_EncryptFinal_ex(ctx, ciphertext + len, &len);
13
14 /* Liberar el contexto de cifrado*/
15 EVP_CIPHER_CTX_free(ctx);
```

Listing 2.11: Sintaxis básica

Descifrado:

```
1  EVP_CIPHER_CTX_new();
2
3  int EVP_DecryptInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type, ENGINE
       *impl, unsigned char *key, unsigned char *iv);
4
5  int EVP_DecryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl,
      unsigned char *in, int inl);
6
7  int EVP_DecryptFinal_ex(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl
      );
```

Listing 2.12: Descifrado

- EVP_CIPHER_CTX_new: Crea el contexto de descifrado.
- EVP_DecryptInit_ex: ctx: Contexto de descifrado, type: algoritmo a utilizar para descifrar, impl: Implementación de algortimo a descifrar, es opcional. key: la clave de descifrado, iv: vector de inicialización a utilizar.
- EVP_DecryptUpdate: ctx: contexto de descifrado, out: el buffer donde se almacenará el texto plano descifrado, outl: longitud del texto descifrado, int:datos a descifrar, inl: longitud de los datos a descifrar.
- EVP_DecryptFinal_ex: ctx: cointexto de descifrado, out: buffer donde se almacenará el texto plano final. outl: longitud del buffer out.

```c
/* Crear un nuevo contexto de cifrado*/
ctx = EVP_CIPHER_CTX_new();

/* Inicializar el contexto de cifrado para descifrar con AES-256-CBC*/
EVP_DecryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv);

/* Descifrar los datos*/
EVP_DecryptUpdate(ctx, plaintext, &len, ciphertext, strlen(ciphertext));

/* Finalizar el descifrado*/
EVP_DecryptFinal_ex(ctx, plaintext + len, &len);

/* Liberar el contexto de cifrado*/
EVP_CIPHER_CTX_free(ctx);
```

Listing 2.13: Descifrado genérico C

Limpieza de OpenSSL:

```c
void EVP_CIPHER_CTX_cleanup(EVP_CIPHER_CTX *ctx);
void EVP_CIPHER_CTX_free(EVP_CIPHER_CTX *ctx);
void RSA_free(RSA *rsa);
void EVP_PKEY_free(EVP_PKEY *pkey);
```

Listing 2.14: Limpieza de OpenSSL

Cargar clave pública RSA desde un documento:

```c
RSA *rsa = PEM_read_RSA_PUBKEY(FILE *fp, RSA **x, pem_password_cb *cb,
    void *u);
```

Listing 2.15: Cargar clave desde un fichero.

- fp: fichero donde se encuentra la clave.
- x: puntero RSA que almacenara la clave cargada.
- cb: función callback que proporciona la contraseña para desencriptar la clave pública si es que se encuentra encriptada.
- u: Parámetro que pasa el usuario a la función de callback.

Se sigue el mismo criterio con la función *PEM_read_PrivateKey()*.

2.2. Cifrado:

2.2.1. Escritura de la clave simétrica y el IV:

Para posteriormente poder descifrar el archivo que generemos encriptado al cifrarlo con una clave simétrica necesitaremos esa misma clave con el IV generado. Para ello, en la primera línea del fichero situaremos de manera encriptada con la clave pública, la clave simétrica y el IV utilizado.

2.2.2. Escritura del resto del documento:

Para el resto del documento, lo que hacemos es: crear una función que irá leyendo las líneas del mismo y llamando a la función encargada de encriptar.

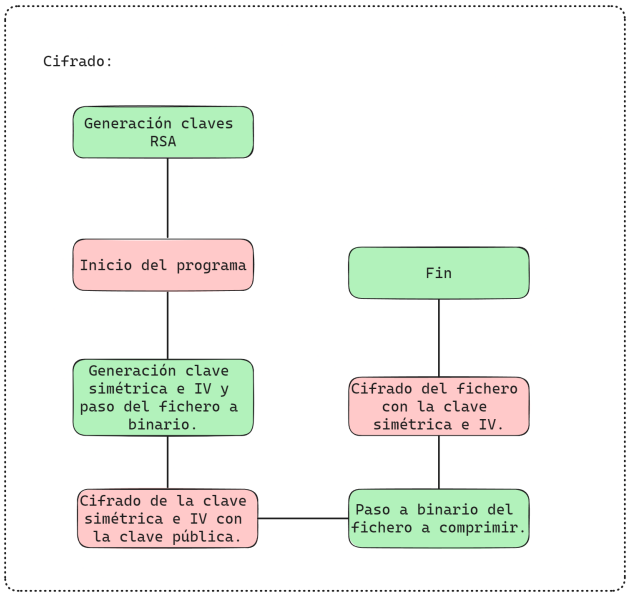

Figura 2.2: Diagrama de flujo del cifrado.

2.3. Descifrado:

En primer lugar para el descifrado, debemos de ser capaces de recuperar la *clave simétrica* y el *IV* utilizado. Para ello, sabemos que en el algoritmo utilizado para el cifrado, el tamaño que ocupa cifrado tanto el IV como la clave simétrica es de 512 bytes cada uno. Mediante *fread* cargaremos en un buffer dichas cadenas y nos encargaremos de descifrarlas. En cambio, el texto cifrado es de 32 bytes.

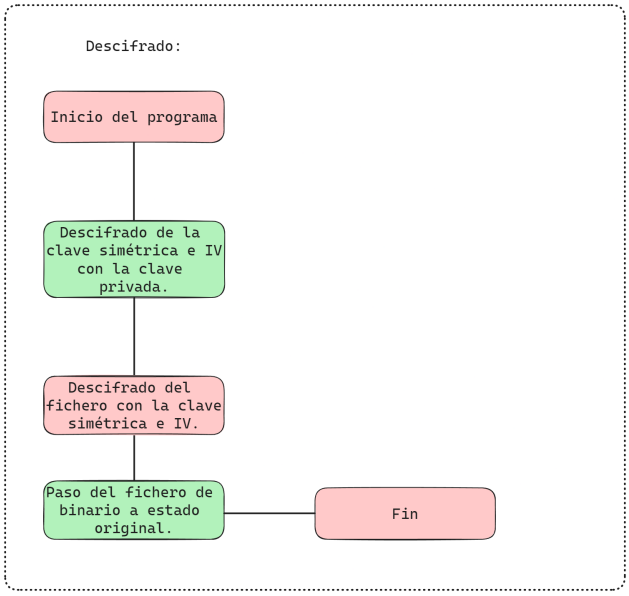

Figura 2.3: Diagrama de flujo del descifrado.

Capítulo 3

Puesta a punto del programa:

3.1. Generación del par de claves RSA:

Es muy importante que estas claves esten siempre localizadas ya que son las llaves maestras para cifrar y descifrar el archivo, en caso de perderlas o ser corruptas los archivos **no** se podrán recuperar.

```
1 openssl genpkey -algorithm RSA -out claveprivada.key -pkeyopt
     rsa_keygen_bits:4096
2 openssl rsa -in claveprivada.key -pubout -out clavepublica.pem
```

Listing 3.1: Generación del par de claves RSA.

3.2. Compilación:

Para la compilación es necesario tener instalado el compilador de C y las librerías de OpenSSL.

```
1 sudo apt install gcc
2 sudo apt install openssl
3 # Situate en la carpeta donde se encuentren los c digos .c y abre una
     terminal.
4 gcc *.c -o ppal -lssl -lcrypto
```

Listing 3.2: Instalación de librerías y compilación.

3.3. Ejecución:

```
1 ./ppal
2 # Sigue las instrucciones de la terminal en base a lo que quieras hacer.
```

Listing 3.3: Ejecución.

Capítulo 4

Códigos de error:

Cifrado:

- −1: Error generando espacio para *rsa_keypub*.
- −2: Error generando una clave simétrica aleatoria.
- −3: Error generando un IV aleatorio.
- −4: Error encriptando la clave simétrica.
- −5: Error encriptando el IV.
- −6: Error con los permisos de apertura de la clave pública.
- −7: La clave publica obtenida es NULL.

Descifrado:

- −1: Error al leer del fichero la clave simétrica cifrada.
- −2: Error al leer del fichero el IV cifrada.
- −3: Problema al intentar acceder a alguno de los ficheros.

Por seguridad los archivos que se generan de manera intermedia y el original únicamente se borraran si el proceso ha funcionado de manera correcta.

Capítulo 5

Fuentes de Documentación:

- MAN directorios
- GNU directorios
- Librería OpenSSL
- Github OpenSSL