ARC4

Ejercicio N° 1

Objetivos	● Buenas prácticas en programación de Tipos de Datos Abstractos (TDAs) ● Modularización de sistemas ● Correcto uso de recursos (memoria dinámica y archivos) ● Encapsulación y manejo de Sockets
Instancias de Entrega	**Entrega 1:** clase 4 (03/04/2018). **Entrega 2:** clase 6 (17/04/2018).
Temas de Repaso	● Uso de structs y typedef ● Uso de macros y archivos de cabecera ● Funciones para el manejo de Strings en C ● Funciones para el manejo de Sockets
Criterios de Evaluación	● Criterios de ejercicios anteriores ● Cumplimiento de la totalidad del enunciado del ejercicio ● Ausencia de variables globales ● Ausencia de funciones globales salvo los puntos de entrada al sistema (*main*) ● Correcta encapsulación en TDAs y separación en archivos ● Uso de interfaces para acceder a datos contenidos en TDAs ● Empleo de memoria dinámica de forma ordenada y moderada ● Acceso a información de archivos de forma ordenada y moderada

Índice

Introducción

La *criptografía de clave secreta* o *criptografía simétrica* consiste en métodos de codificación de la información que usan la misma clave para cifrar y descifrar los mensajes.

Uno de estos métodos lleva el nombre de Rivest Cipher 4, en honor a su creador Ron Rivest, y fue el método por default en varios protocolos de encriptación, como WEP, y WPA, hasta que se descubrieron varias vulnerabilidades. Este método es una marca registrada, pero existen algoritmos cuya salida coincide con RC4. Para evitar problemas legales, a esos algoritmos los podemos encontrar como ARC4 (Alleged RC4).

El presente ejercicio consistirá en construir un sistema distribuido que tendrá un cliente y un servidor, y en el cual el cliente enviará un archivo al servidor cifrando los mensajes con los algoritmos de ARC4.

Descripción

Cliente

La aplicación cliente consiste en un proceso simple que lee un archivo cuyo nombre recibe por parámetro (ver formato de línea de comandos), encripta los datos usando ARC4, y envía los datos cifrados a un servidor. Se debe tener en cuenta que el archivo de entrada podría ser un stream infinito, y que por ende no podemos leerlo en una sola lectura, sino que debemos procesarlo por *chunks,* y enviar la información cifrada (también por *chunks*) al servidor.
Para forzar que el procesamiento se haga como se describe anteriormente, se agregan algunas restricciones respecto de los tamaños de buffers:

- Se deberá leer el archivo de entrada en *chunks* de 64 Bytes (exceptuando la última lectura, si es que los datos no alcanzan para llenar un *chunk*).
- Se deberá encriptar y enviar dicho *chunk* completo al servidor antes de realizar una nueva lectura desde el archivo. **No se puede cargar más de un *chunk* a la vez en memoria.**

Luego del envío de los datos al servidor, el cliente debe terminar la comunicación y finalizar la ejecución ordenadamente.

Servidor

El servidor recibirá los datos cifrados que se envíen desde el cliente, los desencriptará, y guardará un archivo llamado "**out"** con el resultado de la desencripción. El archivo "**out"** debe ser escrito en modo binario, ser creado si no existe, y ser sobrescrito si existe.
Al igual que en el cliente, se establecen algunas restricciones sobre tamaños:

- El servidor deberá leer datos en *chunks* de 50 Bytes (exceptuando la última lectura, si es que los datos no alcanzan para llenar un *chunk*).
- Cada *chunk* se debe desencriptar y volcar al archivo **"out"** por completo antes de seguir leyendo del socket. **No se puede cargar más de un *chunk* a la vez en memoria.**

Luego de atender un cliente, el servidor debe terminar su ejecución ordenadamente.

Alleged Rivest Cipher 4

A continuación se describe el algoritmo ARC4. En dicho algoritmo, tanto la encriptación como la desencriptación usan el mismo procedimiento, que consiste en tres fases:

1. Fase de key-scheduling (KSA)

El algoritmo de key-scheduling consiste en generar un array de estado **S** a partir de la clave secreta. Un posible pseudo-código del KSA es:

```
for i from 0 to 255:
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

Notar que, como el key-scheduling-algorithm sólo depende de la clave secreta, y al ser ARC4 un método simétrico, el arreglo S será el mismo en la encriptación y en la desencriptación.

2. Fase de generación pseudoaleatoria (PRGA)

El algoritmo de generación pseudoaleatoria consiste en generar un flujo de bytes a partir del arreglo de estado **S** obtenido en la fase anterior.
Antes de empezar a generar números aleatorios, debemos inicializar dos variables i y j a cero:

```
i := 0
j := 0
```

Luego, se pueden generar los números pseudoaleatorios K con el algoritmo de PRGA. Un posible pseudo-código del PRGA es:

```
i := (i + 1) mod 256
j := (j + S[i]) mod 256
swap values of S[i] and S[j]
K := S[(S[i] + S[j]) mod 256]
output K
```

Al conjunto de todos los valores generados por el PRGA se le llama *key-stream*, y se usa en la última fase para cifrar los datos.

3. **Fase de cifrado (Cipher)**

El cifrado consiste en realizar una operación de OR exclusiva entre el *key-stream* y los datos a cifrar. Para toda posición i del stream de entrada input, el stream de salida output vale:

```
output[i] := key_stream[i] xor input[i]
```

Si el arreglo **input** contiene los datos sin cifrar, el arreglo **output** tendrá los datos cifrados, y viceversa.

Formato de Línea de Comandos

El binario que se debe generar es solamente uno (debe existir un solo **main** en todo el código). El primer parámetro que recibirá dicho binario es el tipo de aplicación que se quiere correr: si se recibe como primer parámetro **'client'** el binario se debe comportar como un cliente, y si se recibe '**server**' se debe comportar como servidor. A continuación se describen los parámetros extra para cada tipo de aplicación.

Cliente

Al proceso cliente lo invocaremos con la siguiente línea:

```
./tp client <ip-del-servidor> <puerto-del-servidor> <clave> <nombre-del-archivo>
```

Donde la clave se usará para encriptar el stream de datos, el nombre del archivo indica en qué archivo se encuentran los datos a transferir.

Servidor

El proceso servidor será invocado de la siguiente manera:

```
./tp server <puerto-del-servidor> <clave>
```

Donde la clave se usará para desencriptar el stream de datos.

Códigos de Retorno

Ambos procesos, cliente y servidor, deberán retornar 0 si se ejecutaron normalmente, o bien 1 si hubo algún error de parámetros.

No hay otros códigos de retorno estipulados en el alcance del ejercicio.

Entrada y Salida Estándar

Cliente

Entrada estándar

La aplicación cliente no recibe nada por entrada estándar.

Salida estándar

Se deberán imprimir por salida estándar los datos cifrados, en formato hexadecimal (en base 16) y en minúsculas (ver ejemplos de ejecución).

Salida estándar de errores

Si hay errores en los parámetros de entrada, se deben imprimir el mensaje:

```
Parámetros incorrectos.
```

Además se debe imprimir por salida estándar de errores el *key-stream* generado, en formato hexadecimal y en mayúsculas, ocupando dos caracteres de texto por cada byte a imprimir (ver ejemplos de ejecución).

Servidor

Entrada estándar

La aplicación servidor no recibe nada por entrada estándar.

Salida estándar

Se deberán imprimir por salida estándar los datos desencriptados, en formato hexadecimal y en minúsculas (ver ejemplos de ejecución).

Salida estándar de errores

Si hay errores en los parámetros de entrada, se deben imprimir el mensaje:

```
Parámetros incorrectos.
```

Además se debe imprimir por salida estándar de errores el *key-stream* generado, en formato hexadecimal y en mayúsculas, ocupando dos caracteres de texto por cada byte a imprimir (ver ejemplos de ejecución).

Hint: Investigar los distintos formatos que se pueden utilizar en la función **printf**.

Ejemplos de Ejecución

A continuación veremos algunos ejemplos de ejecución mostrando entradas y salidas esperadas.

Ejemplo 1: Primer ejemplo de Wikipedia

Archivo de entrada **input.txt**:

```
Plaintext
```

Si ejecutamos el servidor:

```
./tp server 7777 Key
```

Y el cliente:

```
./tp client 127.0.0.1 7777 Key input.txt
```

Debemos obtener las siguientes salidas:

Salida estándar del cliente (es el archivo cifrado representado en base 16, notar las minúsculas):

```
bbf316e8d940af0ad3
```

Salida estándar de errores del cliente (es el *key-stream* representado en base 16, notar las mayúsculas):

```
EB9F7781B734CA72A7
```

Salida estándar del servidor (es la representación en base 16 de los datos originales):

```
506c61696e74657874
```

Salida estándar de errores del servidor (misma salida de errores que el cliente):

```
EB9F7781B734CA72A7
```

Archivo **out** generado por el servidor:

```
Plaintext
```

Notar que el archivo **input.txt** y el archivo **out** deben coincidir, así como también ambas salidas estándar de errores. También debe suceder que la salida estándar del servidor sea la representación hexadecimal del archivo **out**. Estas dos particularidades pueden servirle al desarrollador para una detección temprana de errores.

Ejemplo 2: Pan con queso

Archivo de entrada **input.txt**:

```
Pan
```

Si ejecutamos el servidor:

```
./tp server 7777 queso
```

Y el cliente:

```
./tp client 127.0.0.1 7777 queso input.txt
```

Debemos obtener las siguientes salidas:

Salida estándar del cliente:

```
69cae6
```

Salida estándar de errores del cliente:

```
39AB88
```

Salida estándar del servidor:

```
50616e
```

Salida estándar de errores del servidor:

```
39AB88
```

Archivo **out** generado por el servidor:

```
Pan
```

Notar que todas las representaciones hexadecimales de cada ejecución tienen el mismo largo

Aclaración importante: Los datos deben ser impresos en ambos **stdout** y **stderr** en una representación base 16, pero la transferencia de datos se debe hacer en binario. En este ejemplo de ejecución se deben transferir por un socket exactamente **tres** bytes: el **01101001 (0x69)**, el **11001010 (0xCA)**, y el **11100110 (0xE6)**.

Ejemplo 3: Ejemplo con hexas chicos

Archivo de entrada **lorem_ipsum.txt**:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut
labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

Si ejecutamos el servidor:

```
./tp server 7777 Secret
```

Y el cliente:

```
./tp client 127.0.0.1 7777 Secret input.txt
```

Debemos obtener las siguientes salidas:

Salida estándar del cliente:

```
48bb1960518812293207 5d0a 88f4d5fd4031a75c2d6f611a78bd357f2b199ef4961630196258400e a489c361ffc53c
d3a9b85e86b3a2bf73ba3f158aca8040f287a2abb132d15108 e83eeaa830e512bd000f 82eac44fc25e4b4de688c6ad
422ca18f8ca41160cff6595aaa1722ec8ee978b9b2d470a0310c 36a27e967e4192cd6dc430b4162d88359e5e2623d5
d3c923c5de88f8a2eb2ab8614d2c25c95a2dd79e68a3f1c5f783a32bd7a2be9876e4a5d9adaaccc4ada775d581172c
ef4216c6897140 0a 5c58f7522b04 1830b3e73d69d2f9b85f4c4285 0c adf4f165a2cb29ad4b6798c0d017487951518d
939e63af33c7dd9b435a2140ceabc79caf8f29bd97689c4c3c46c29dfe5113ca 0e 3baf678bb807 f0d81e52dc15c508
e108 66687259de88fbfb561b3ca02664d5ce5223c39c7523de20fc6b34c2c334172dfd7b3715cedf09 ee870c 3f537f
87ff2b5e795e22d0e1f5aa06 ac3504 afde1bd02825cbab114bd46794f7d919c1faf9c08333ca8446552fb7bbc65997
db8103 4c1bcc5e0e08 46abaf741a78fcf6653db7c436362998909e32392ee002 a13e58be99800f 2cebcc9978ae422f
a55fea192a0f 9793c99ea5fed2d27808 17391494b762
```

Salida estándar de errores del cliente:

```
04D46B053CA87B594172302AEC9BB9923211D435594F00771DC9195F4876F087F375447C162D322EC5EDAA1196B65F
BAC7DF7EE3DFCBCB5F9A4C70EEEAE42FD2E2CBDEC25FBE35289C5B87D85F9732D46E6CEB8EAD2BB7303F6D93FCE6C1
234ECEFDE9847414EF923636C56547CCE3881FD7D3F411CC587D43C350B62B35B2A8
03AD5D947749A858F7304F4EF5A5AC4DACBFE5D4829A5FD1126D424ABA2E5FA2FA48C689A085E0CA5FB6D6D7F718C4
D0B5C1CBA1A7C28719B4E3785E863136A8E002292A292CD733476D6945DA971D0CAAD9DD3E6C21EA61C09B950A82A8
46C33802E9B5B16366591524E4E0BE02DA47A2FDF2312F5325EECFA8F0C0FD09D4F948EE294C34A7F59B3F77AF7C52
DB47E2D62786B77227AC61A47C8428100D1E30AAA89E88257E1CC34F08B9BB3F03A7F3194CAC45DC0E41E2A5417044
9C0F177BBBB3658FA77C5E2116E68B5E2C577E67A88290DA72C940768FAD72BE5C05A4C8722AB104F583F97AB48A90
A4E247ABF0663B40D99BB62BF8B2E566226FE07E7D7D28DF8F1D74589F8309 4DD6E4474340B8FFF854504D8963815A
3DCDFCF27A429FECF417C22E46D17F8B774362B7FAADBEC08DA6F21469755666E1DA4C
```

Salida estándar del servidor:

```
4c6f72656d20697073756d20646f6c6f722073697420616d65742c20636f6e73656374657475722061646970697363
696e6720656c69742c2073656420646f20656975736d6f642074656d706f7220696e6369646964756e74207574206c
61626f726520657420646f6c6f7265206d61676e6120616c697175612e20557420656e696d206164206d696e696d20
```

```
76656e69616d2c2071756973206e6f737472756420657865726369746174696f6e20756c6c616d636f206c61626f72
6973206e69736920757420616c697175697020657820656120636f6d6d6f646f20636f6e7365717561742e20447569
732061757465206972757265726520646f6c6f7220696e2072657072656568656e646572697420696e20766f6c7570746174
652076656c697420657373652063696c6c756d20646f6c6f726520657520667567696174206e756c6c612070617269
617475722e204578636570746575722073696e74206f6361656361742063757069646174617420e6f6e2070726f
6964656e742c2073756e7420696e2063756c706120717569206f6666696369612064657365727 56e74206d6f6c6c69
7420616e696d206964206573742061626f72756d2e
```

Salida estándar de errores del servidor:

```
04D46B053CA87B594172302AEC9BB9923211D435594F00771DC9195F4876F087F375447C162D322EC5EDAA1196B65F
BAC7DF7EE3DFCBCB5F9A4C70EEEAE42FD2E2CBDEC25FBE35289C5B87D85F9732D46E6CEB8EAD2BB7303F6D93FCE6C1
234ECEFDE9847414EF923636C56547CCE3881FD7D3F411CC587D43C350B62B35B2A803AD5D947749A858F7304F4EF5
A5AC4DACBFE5D4829A5FD1126D424ABA2E5FA2FA48C689A085E0CA5FB6D6D7F718C4D0B5C1CBA1A7C28719B4E3785E
863136A8E002292A292CD733476D6945DA971D0CAAD9DD3E6C21EA61C09B950A82A846C33802E9B5B16366591524E4
E0BE02DA47A2FDF2312F5325EECFA8F0C0FD09D4F948EE294C34A7F59B3F77AF7C52DB47E2D62786B77227AC61A47C
8428100D1E30AAA89E88257E1CC34F08B9BB3F03A7F3194CAC45DC0E41E2A54170449C0F177BBBB3658FA77C5E2116
E68B5E2C577E67A88290DA72C940768FAD72BE5C05A4C8722AB104F583F97AB48A90A4E247ABF0663B40D99BB62BF8
B2E566226FE07E7D7D28DF8F1D74589F8309 4DD6E4474340B8FFF854504D8963815A3DCDFCF27A429FECF417C22E46
D17F8B774362B7FAADBEC08DA6F21469755666E1DA4C
```

Archivo **out** generado por el servidor:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut
labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

Ver texto en negrita y **notar** que las representaciones hexadecimales muestran cada byte con **dos** caracteres.

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en ISO C (C99).
2. Está prohibido el uso de variables globales.
3. Se deben cumplir las restricciones sobre tamaños especificadas en la sección **Descripción.**

Referencias

[1] RC4: https://en.wikipedia.org/wiki/RC4
[2] Página de manual de printf: man 3 printf