

# Proyecto final Programación de Sistemas 2019-1T

Preparado por: Federico Domínguez

Agosto 9, 2019

El proyecto final consiste en la creación de un grupo de chat usando el modelo cliente – servidor, serialización de datos con CBOR y mensajería encriptada.

## Programa servidor

El programa servidor debe poder mantener varias conexiones de clientes simultáneamente y tener el siguiente uso, el cuál debe mostrarse al usar la opción `-h` (mostrar ayuda en inglés):

```
./chat_server -h
chat_server distributes encrypted chat messages between connected clients.
```

Usage:

```
chat_server [-d] <port>
chat_server -h
```

Options:

```
-h          Help, show this screen.
-d          Daemon mode.
```

Por defecto, *chat\_server* se ejecuta en modo consola y usando la opción `-d` como *daemon* en el *background* y escuchando conexiones de clientes en el puerto especificado como argumento.

**Opcional:** Eventos como una conexión nueva o algún mensaje de error pueden ser registrados en la bitácora general de Linux usando la interface de la librería [syslog](#).

## Programa cliente

El programa cliente tiene el siguiente uso:

```
./chat_client -h
chat_client connects to a remote chat_server service, allows the user to send messages to the chat group and displays chat messages from other clients.
```

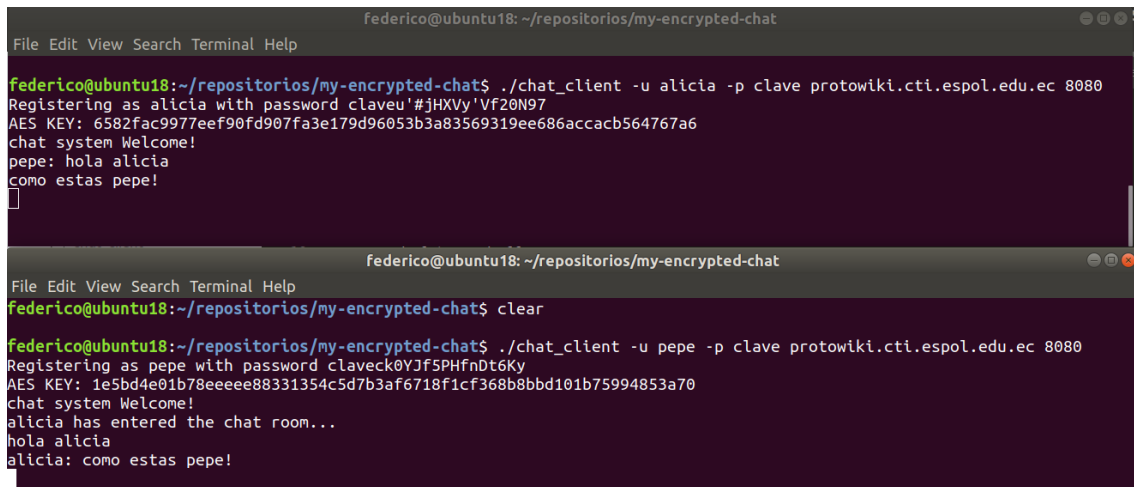
Usage:

```
chat_client -u <user> -p <password> <ip> <port>
chat_client -h
```

Options:

```
-h          Help, show this screen.
-u          Specify the username.
-p          Specify the password.
```

`chat_client` permite ingresar texto desde consola el cual es enviado como mensaje a `chat_server`. El servicio en `chat_server` debe reenviar el mensaje a cada cliente conectado, exceptuando el cliente remitente (Ilustración 1).



```
federico@ubuntu18: ~/repositorios/my-encrypted-chat
File Edit View Search Terminal Help

federico@ubuntu18:~/repositorios/my-encrypted-chat$ ./chat_client -u alicia -p clave protowiki.cti.espol.edu.ec 8080
Registering as alicia with password claveu'#jHXVy'VF20N97
AES KEY: 6582fac9977eef90fd907fa3e179d96053b3a83569319ee686accacb564767a6
chat system Welcome!
pepe: hola alicia
como estas pepe!
_

federico@ubuntu18: ~/repositorios/my-encrypted-chat
File Edit View Search Terminal Help

federico@ubuntu18:~/repositorios/my-encrypted-chat$ clear

federico@ubuntu18:~/repositorios/my-encrypted-chat$ ./chat_client -u pepe -p clave protowiki.cti.espol.edu.ec 8080
Registering as pepe with password claveck0YJf5PHfnDt6Ky
AES KEY: 1e5bd4e01b78eeee88331354c5d7b3af6718f1cf368b8bbd101b75994853a70
chat system Welcome!
alicia has entered the chat room...
hola alicia
alicia: como estas pepe!
```

Ilustración 1: Ejemplo de dos clientes en una sesión de chat

## Comunicación cliente - servidor

El programa servidor debe aceptar varios clientes simultáneamente usando hilos. La comunicación entre cliente y servidor debe ser codificada usando [CBOR](#), específicamente la librería [libcbor](#) (aunque cualquier implementación de CBOR en C es permitida).

La comunicación entre cliente y servidor debe ser siempre en bloques de 256 bytes, si la cantidad de datos a enviar es menor entonces los bytes restantes deberán ser ceros. Los mensajes de chat deben ser enviados a todos los clientes conectados, similar a un grupo de chat. Cuando un cliente se conecta o desconecta, se debe enviar un mensaje de “presencia” al resto de clientes. La comunicación debe seguir una lógica similar a lo mostrado en la Ilustración 2.

**Opcional:** La clave puede estar “quemada” en el código, pero el servidor podría mantener un registro de usuarios y claves para implementar una verdadera autenticación.

**Opcional:** Por defecto un mensaje debe ser enviado a todos los usuarios conectados, pero se podría usar el campo *DESTINATION* (ver sección Codificación en CBOR) para especificar un usuario destino lo cual funcionaría como un mensaje privado.

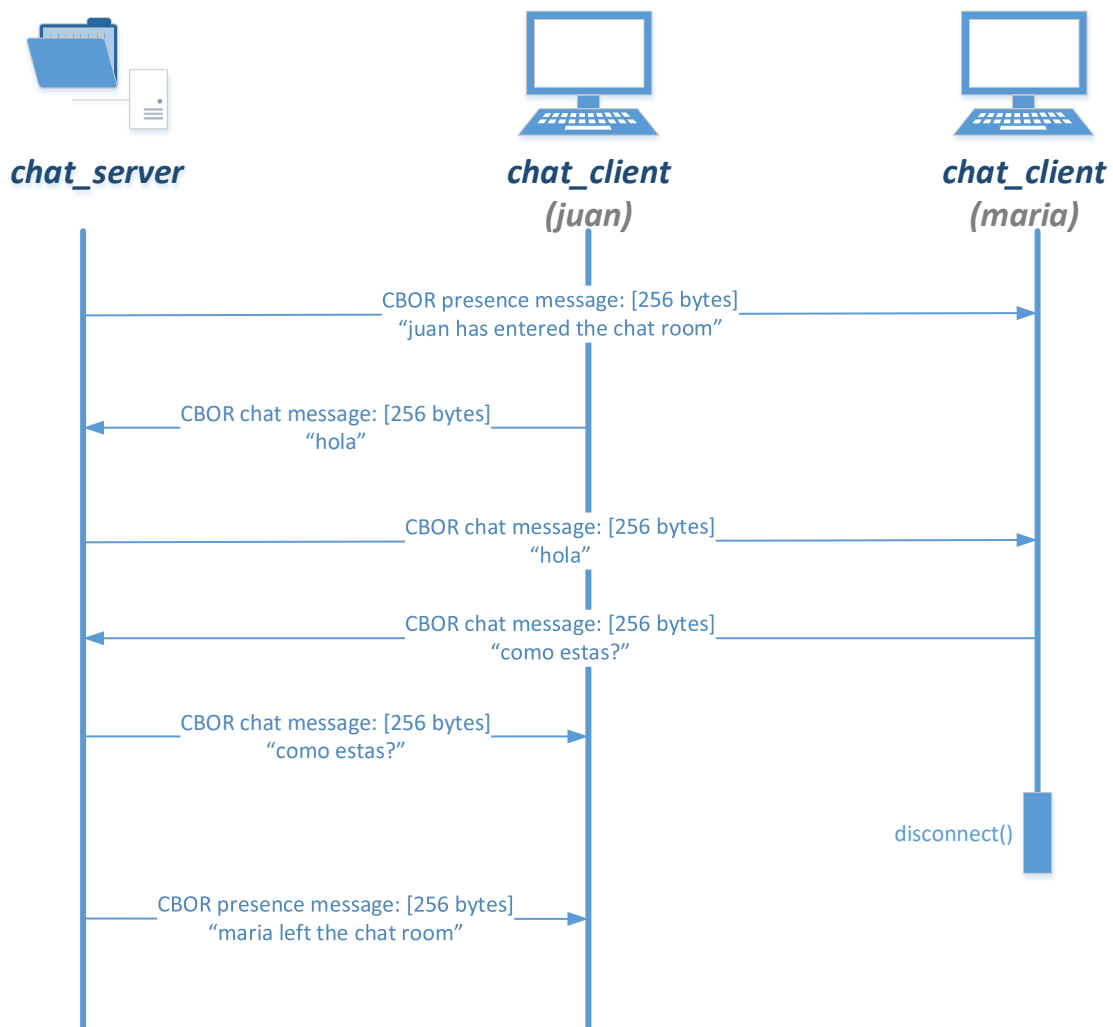


Ilustración 2: Todos los mensajes enviados entre cliente y servidor deben ser en bloques de 256 bytes. Además de mensajes de chat, se deben enviar mensajes de presencia.

## Encriptación

La aplicación usa encriptación AES para la transmisión de mensajes a través de la red. Se debe usar el algoritmo de encriptación AES256 en modo CBC. Se recomienda usar el código en:

- <https://github.com/B-Con/crypto-algorithms>

Es mandatorio enviar todos los mensajes en bloques encriptados de 256 bytes, excepto por los primeros mensajes de inicio de sesión como se aprecia en la Ilustración 3. El uso del modo CBC en AES requiere definir un vector inicial (IV) de 16 bytes el cuál debe ser generado de manera aleatoria por el servidor y enviado al cliente sin encriptar.

Cada cliente debe tener un usuario y clave predefinido. La clave es conocida por ambos, el cliente y el servidor, y no debe ser enviada durante la conexión. Al iniciar la sesión, el cliente envía un mensaje sin encriptar con su usuario y un *token* aleatorio al servidor. La clave concatenada con el *token* será usada para generar la llave AES de 256 bits como se muestra en la Ilustración 3.

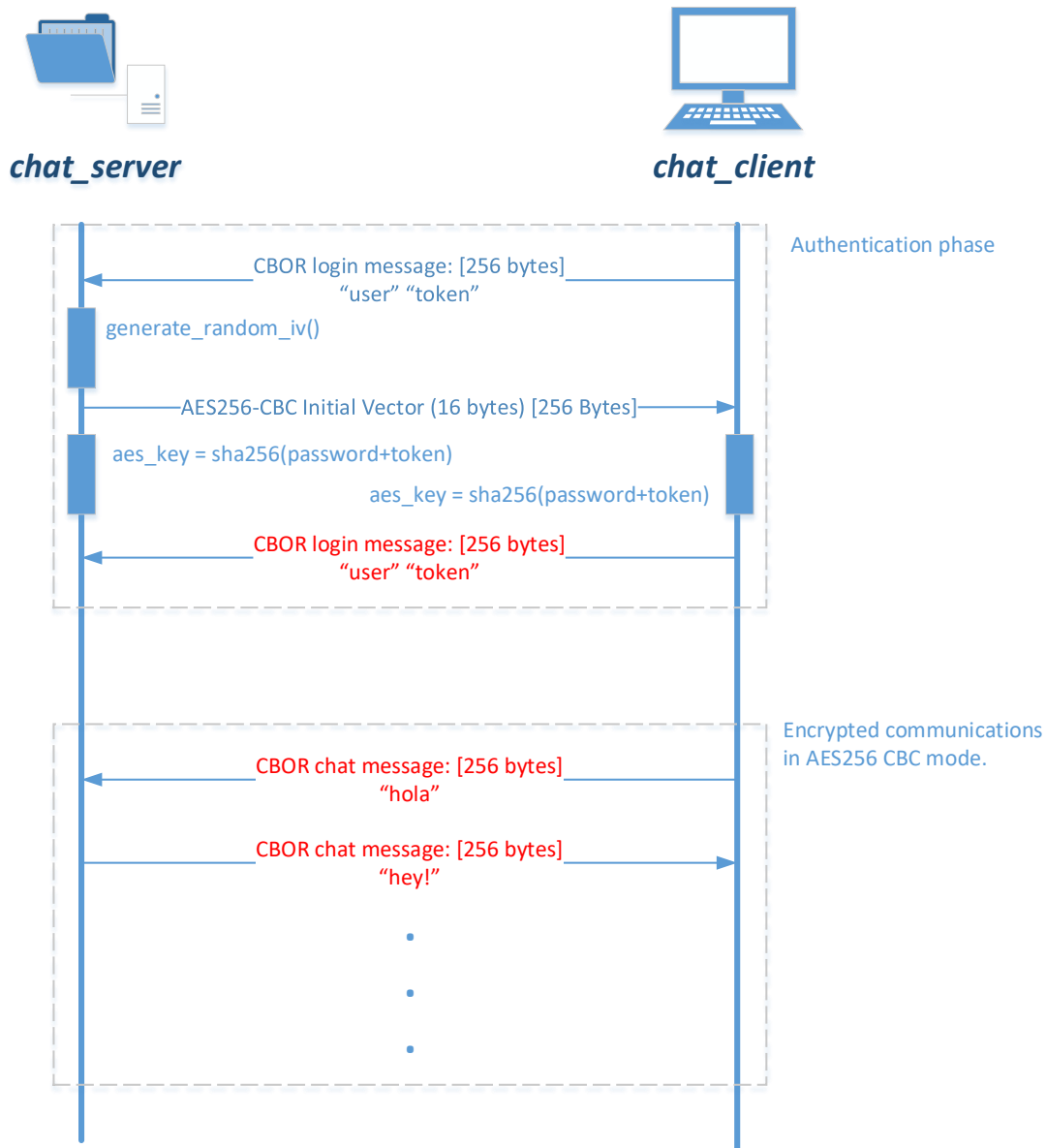


Ilustración 3: Inicio de sesión entre cliente y servidor en donde se establecen las llaves AES 256 bits. Los mensajes en rojo son encriptados.

## Codificación en CBOR

En esta aplicación existen dos tipos de mensajes: login y chat. La codificación en CBOR usará parejas *key – value* para enviar mensajes de chat o mensajes de login. En la Tabla 1 se aprecian los 5 tipos de parejas *key – value* que la aplicación debe soportar. Las llaves (*key*) se representan con un entero (*UINT8*) del 1 al 5 y los valores (*value*) con cadena de caracteres (excepto PRESENCE). El mensaje debe ser empaquetado usando CBOR MAP.

Tabla 1: Parejas key - value en codificación CBOR

Campo	Key	Value	Descripción	Uso
MESSAGE	1	STRING: 140 bytes	Texto del mensaje	Chat message
USER	2	SRING: 32 bytes	Usuario que envía el mensaje	Chat/login message
TOKEN	3	STRING: 32 bytes	Token aleatorio	Login message
DESTINATION	4	STRING: 32 bytes	Opcional: Usuario destino	Chat message
PRESENCE	5	BOOLEAN	"true" si es mensaje presencia	Chat message

Para enviar un mensaje es necesario primero empaquetarlo y luego encriptarlo como se muestra en la Ilustración 4. El proceso de recepción de mensajes es el inverso.

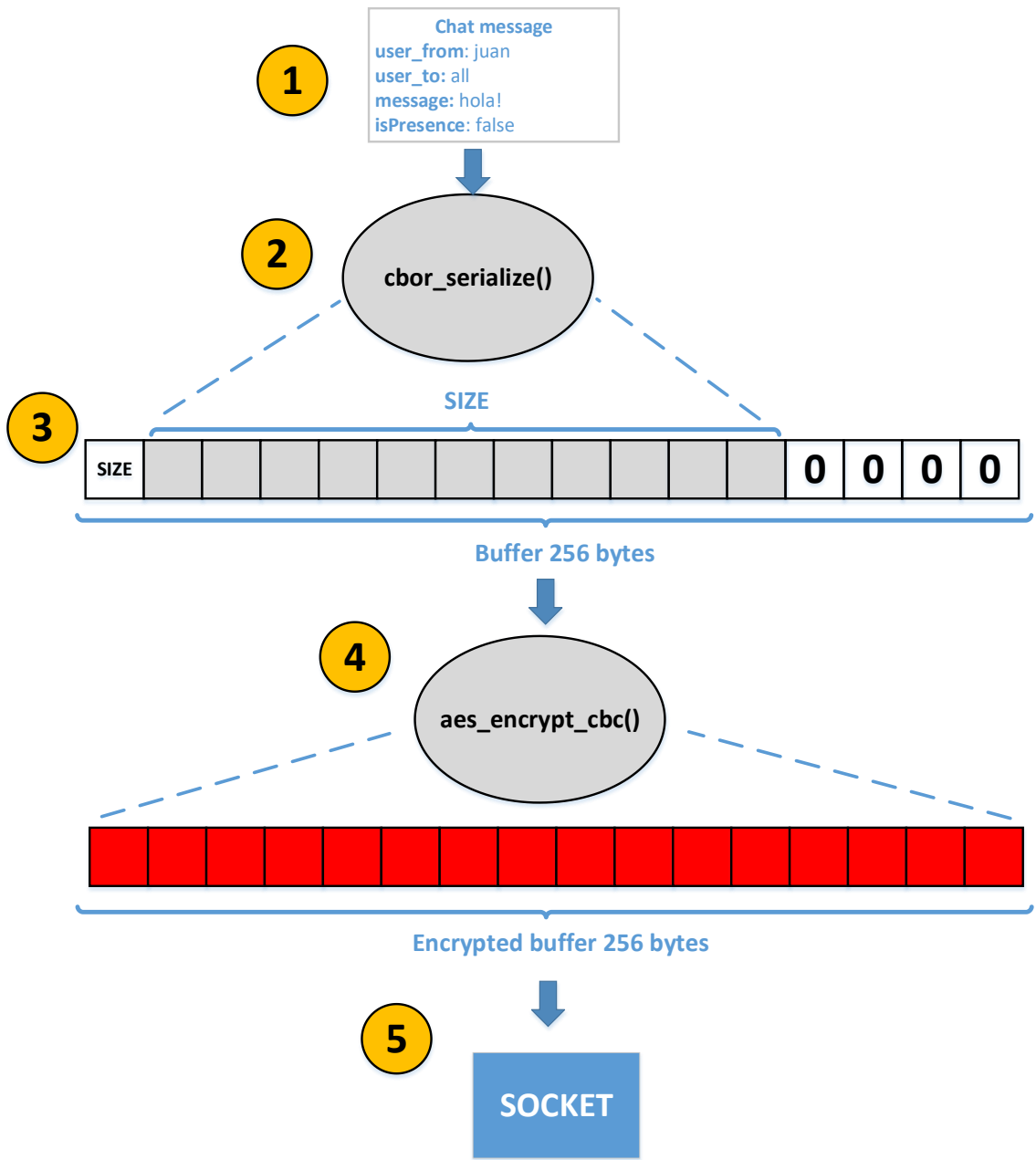


Ilustración 4: El proceso de envío de un mensaje es 1) armar el mensaje en una estructura en memoria; 2) serializar el mensaje usando CBOR; 3) copiar el mensaje serializado a un buffer de 256 bytes, el primer byte debe representar el tamaño del mensaje, los bytes sobrantes deben contener ceros; 4) encriptar el buffer completo usando AES256; 5) enviar el buffer encriptado a través del socket.