1 Práctica chat udp cifrado

El objetivo es crear un servidor de chat y un cliente mediante udp. Las comunicaciones serán cifradas bajo el esquema que se explicará en la práctica.

El funcionamiento del cliente y el servidor será el siguiente:

Nota: Todas las comunicaciones entre cliente y servidor se harán mediante mensajes JSON (encriptados o no).

Se necesitarán claves públicas y privadas para el servidor y el cliente: cliente.publica, cliente.privada, servidor.publica y servidor.privada.

1. Se inicia el servidor. El servidor debe leer de disco una clave pública y otra privada.

   ```
   $ node servidor.js
   ```
2. Se arranca el cliente indicando por línea de comandos el nombre del usuario en el chat:

   ```
   $ node cliente.js pepe
   ```
   El cliente leerá las claves públicas y privadas desde disco. También leerá la clave pública del servidor desde disco.
3. El cliente se debe conectar al servidor y enviar su clave pública.
4. El servidor generará una contraseña aleatoria para cifrar posteriormente los mensajes de forma simétrica y usará la clave pública del cliente cifrarla.
5. El servidor enviará al cliente la clave cifrada con la clave pública del cliente y una firma de la clave.
6. El cliente recibirá la clave cifrada y la firma. Descifrará la clave y comprobará la firma.
7. A partir de entonces los mensajes entre cliente y servidor se cifrarán usando la contraseña simétrica generada. El cifrado puede ser del mensaje JSON completo o de cada valor de los campos del mensaje.
8. El cliente mandará los mensajes al servidor en JSON de la forma: {“usuario”: “nombre”, “mensaje”: “texto del mensaje”}

   Donde el token en la clave simétrica cifrada con la clave pública del servidor que se envió en el paso 5.
9. El servidor cuando reciba un mensaje del cliente, lo descifrará y se lo enviará a cada uno de los clientes que estén conectados en ese momento.

Destacar que para cliente que se conecte al servidor deberá tener una contraseña diferente. El servidor sabrá qué contraseña tiene que usar para cada cliente pues llevará un registro de las mismas.

La sesión del cliente debe ser interactiva para ello se usará la biblioteca “readline”. Un ejemplo de uso de la biblioteca “readline”:

```
const readline = require('readline');

console.log("Pulsa ctl+c para parar.");
process.stdout.write('> ');
```

```
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
  terminal: false
});

rl.on('line', (line) => {
  console.log(line);
  process.stdout.write('> ');
});

rl.once('close', () => {
  // end of input
});
```

En el ejemplo aparece la línea:

```
process.stdout.write('> ');
```

Esta línea sirve para poner el símbolo > y que el usuario siga escribiendo en la línea actual y no en la siguiente.

Una sesión de ejemplo del cliente podría ser:

```
$ node cliente.js pepe
> hola
lolo: mundo
paco: hola
> adios
lolo: adios
paco: adios
```

Nota: Puede usar “\r” para borrar lo que había anteriormente en la línea y quitar el símbolo “>” anterior:

```
console.log(“\rlolo: mundo”)
```

Se puntuará:

1. (1 pto) Que se intercambie la clave pública del cliente y se use para cifrar la contraseña simétrica.
2. (2 ptos) Que se firme la contraseña simétrica generada desde el servidor y se compruebe desde el cliente.
3. (2 ptos) Que se cifren de forma simétrica los mensajes que se intercambien cliente y servidor.
4. (2 ptos) Que el servidor cumpla con sus funciones de envío y recepción de mensajes según lo estipulado en la práctica.
5. (2 ptos) Que el cliente cumpla con sus funciones de envío y recepción de mensajes según lo estipulado en la práctica.
6. (1 pto) Puede cambiar el color del texto escrito en la consola usando las cadenas de escape que se describen en el siguiente artículo:

https://cartaslinux.wordpress.com/2021/09/14/cambiando-el-color-de-la-salida-estandar-en-la-consola/

Se valorará que los usuarios y los mensajes salgan coloreados en colores y atributos distintos, por ejemplo:

```
$ node cliente.js pepe
> hola
lolo: mundo
paco: hola
> adios
lolo: adios
paco: adios
```