Universidad Carlos III

Criptografía

Curso 2021-22

FECHA-ENTREGA: **09/09/2021**

GRUPO: **80**

Eduardo De Andrés Tabernero - 100363553

ÍNDICE

ÍNDICE 2

¿Cuál es el propósito de su aplicación? 3

¿Para qué utiliza cifrado simétrico o asimétrico? ¿Qué algoritmos ha utilizado y por qué? ¿Cómo gestiona las claves? 4

¿Para qué utiliza las funciones hash o HMAC? ¿Qué algoritmos ha utilizado y por qué? En caso de HMAC, ¿cómo gestiona la/s clave/s? 6

1. ¿Cuál es el propósito de su aplicación?

CriptoChat es una aplicación que simula un servicio de chat online cifrado de “extremo a extremo”. A esta sala de chat se pueden conectar tantos usuarios como lo deseen, pero deberán hacerlo mediante una clave de acceso al chat. Si los usuarios cuentan con la clave adecuada, podrán entrar en la sala de chat y comenzar a participar en ella.

Cada mensaje que envían los usuarios es cifrado y descifrado, de forma que el receptor consigue leer el mensaje original que introduce el emisor como texto plano, pero viaja de forma segura por el chat, ya que es el receptor el que descifrará dicho mensaje.

Este sistema funciona mediante un intermediario de confianza (en nuestro caso el servidor) que se encarga de todas las funciones de distribución de mensajes y control de acceso dentro de la sala de chat. Como tenemos varios usuarios dentro de nuestra sala el servidor se encarga de realizar el anteriormente cifrado de “extremo a extremo” uno por uno con el usuario que envía el mensaje y cada uno del resto de usuarios que están en la sala.

2. ¿Para qué utiliza cifrado simétrico o asimétrico? ¿Qué algoritmos ha utilizado y por qué? ¿Cómo gestiona las claves?

Como se ha mencionado anteriormente, la aplicación utiliza el cifrado de extremo a extremo, es decir cifrado asimétrico. Esto es porque los cifrados se producen a nivel dispositivo, es decir, los mensajes del emisor se cifran antes de ser enviados y los mensajes recibidos por el receptor no se descifran hasta que llegan a su destino. De esta forma, es muy complicado que se pueda interceptar información del chat, ya que un atacante necesitaría las claves privadas de cada usuario para poder descifrar los mensajes, además, tendría que haber conseguido previamente la clave del chat para poder acceder al mismo.
En cuanto al cifrado simétrico, este presenta un gran problema ya que se debe intercambiar la clave a través del chat, lo cual puede comprometer sustancialmente la seguridad de la aplicación en caso de que un atacante intercepte dicha clave.
En cuanto al algoritmo de cifrado asimétrico empleado en esta aplicación, se ha utilizado RSA, y se va a explicar cómo se produce el intercambio de mensajes entre el servidor y los clientes conectado al chat en este caso:

- En primer lugar, se crean los pares de claves pública y privada del usuario y del servidor mediante la función *RSA_generator()* de la librería *Cryptodome*:

```
from Cryptodome.PublicKey import RSA
```

```
#CREAMOS PARES DE CLAVES PUBLICAS
global private_key
global public_key
keys = criptool.RSA_generator()
private_key = keys[0]
public_key = keys[1]
print('\n**************************************************')
print(f"private key: {private_key}\npublic key: {public_key}")
print('\n**************************************************')
```

- Una vez se conecta el usuario al servidor, tras haber sido validada la clave de acceso al chat por el servidor (ver en el apartado de Hash), el servidor envía un mensaje de bienvenida al usuario. Para esto, y para el envío de cualquier mensaje se crea la función *broadcast_message()*, que envía dicho mensaje a todos los usuarios conectados excepto al que envía el mensaje:

```
#ENVIA EL MENSAJE A TODOS LOS CLIENTES QUE ESTEN CONECTADOS
def broadcast_message(message, sender, username):
    for client in clients: #enviamos el mensaje a todos los conectados
        if client != sender: #evitamos enviarnos el mensaje a nosotros mismos
            msg = criptool.RSA_encrypt(f"{username}: {message}", criptool.RSA_key_format(public_keys[client]))
            client.send(msg.encode('utf-8'))
```

- Como se puede observar, se utiliza la función *RSA_encrypt()*, que recibe como parámetros el mensaje a enviar y la clave pública del cliente, con la que se cifrará el mensaje:

```
def RSA_encrypt(text, key):
    msg = text.encode()
    cipherRSA = PKCS1_OAEP.new(key)
    encrypt_msg = cipherRSA.encrypt(msg)
    return binascii.hexlify(encrypt_msg).decode('utf-8')
```

- En cuanto a la recepción de mensajes, el servidor recibe los mensajes a través de la función *handle_messages()*:

```
#MANEJA LA RECEPCION DE PETICIONES QUE SE HACEN AL CLIENTE
def handle_messages(client):
    index = clients.index(client)
    username = usernames[index]
    while True:
        try:
            message = client.recv(1024).decode('utf-8')
            decrypt = criptool.RSA_decrypt(message, private_key)
            broadcast_message(decrypt, client, username)
        except:
            broadcast(f"CriptoBot: el usuario {username} se ha desconectado.".encode('utf-8'), client)
            del public_keys[client]
            clients.remove(client)
            usernames.remove(username)
            client.close()
            break
```

- Estos mensajes llegan cifrados, por lo que el servidor deberá descifrar dicho mensaje con su clave privada. La función *RSA_decrypt()* será la encargada de ello, y recibe como parámetros el mensaje a descifrar y la clave privada del servidor:

```
def RSA_decrypt(crypt, key):
    crypt = binascii.unhexlify(crypt.encode('utf-8'))
    cipherRSA = PKCS1_OAEP.new(key)
    msg = cipherRSA.decrypt(crypt)
    return msg.decode('utf-8')
```

3. ¿Para qué utiliza las funciones hash o HMAC? ¿Qué algoritmos ha utilizado y por qué? En caso de HMAC, ¿cómo gestiona la/s clave/s?

En esta aplicación, se utilizan las funciones Hash para verificar la entrada de los usuarios al chat. Tanto el usuario como el servidor comparten una clave de acceso al chat, de forma que el usuario enviará al servidor dicha clave hasheada. Una vez recibida por el servidor, este obtendrá el hash equivalente de su clave, para posteriormente comprobar dicho hash con el que le había enviado el usuario previamente. En caso de que ambos hashes coincidan (ya que si la clave es la misma, el resultado del hash será siempre el mismo), significa que el usuario puede acceder el chat, por lo que el servidor añadirá al usuario al chat.
De esta forma, la clave viajará entre el cliente y el servidor siempre hasheada, por lo que no se podrá "filtrar" la clave original en caso de producirse un ataque al servidor, ya que a partir de un hash, es imposible obtener la entrada que genera dicho hash (el atacante debería probar posibles contraseñas hasta que se generase ese mismo hash para poder obtener la clave, lo cual es de una complejidad y duración muy elevadas).
En este caso, se ha optado por crear dichos hashes mediante SHA-512 (librería hashlib) ya que es un algoritmo con un gran equilibrio entre seguridad y velocidad.

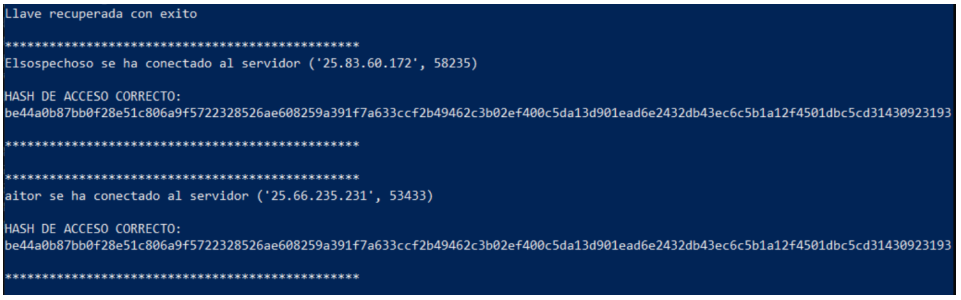
```python
def sha_512(data_to_hash):
    byte_hash = hashlib.new('sha512',data_to_hash)
    hex_hash = byte_hash.hexdigest()
    return hex_hash
```

- La función recibe por parámetro la clave que se ha de cifrar. Para ello, se crea el elemento *byte_hash*, indicando mediante el método *new()* el tipo de hash que se va a generar y la clave.
- El resultado de esta función tiene una serie de propiedades, como la *hexdigest()*, que convierte de bits a hexadecimal lo que permite enviar las claves en un formato acorde a un string..

A continuación se muestra cómo se llevan a cabo las comprobaciones previamente mencionadas. Se observa cómo comprueba el servidor la validación de la contraseña de acceso al chat:

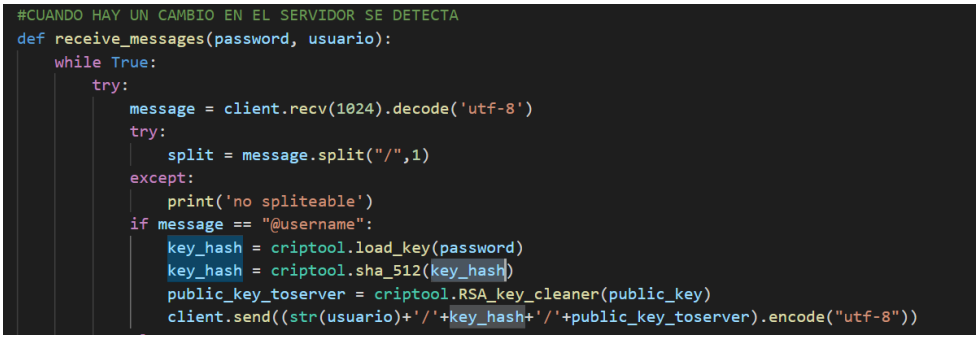
```python
while True:
    client, address = server.accept()
    client.send("@username".encode("utf-8"))
    pass_conf = criptool.load_server_key()
    pass_conf = criptool.sha_512(pass_conf)
    combined_msg = client.recv(1024).decode('utf-8')
    username, password, node_public_key = combined_msg.split('/')
    if password == pass_conf:
        client.send(("***/"+criptool.RSA_key_cleaner(public_key)).encode("utf-8"))
        clients.append(client)
        usernames.append(username)
        public_keys[client] = node_public_key
        print('\n**********************************************')
        print(f"{username} se ha conectado al servidor {str(address)}\n\nHASH DE ACCESO CORRECTO:\n{password}")
        print('\n**********************************************')
```

- En primer lugar, el servidor pide al usuario que introduzca sus credenciales (username, la clave de acceso al servidor hasheada y su clave pública), mediante el comando @username.
- Posteriormente, obtiene la clave de acceso al chat (que se encuentra almacenada en los ficheros "*server_key.key*" en este caso y "*criptochat.key*" para el caso de los clientes) y procede a hashearla.
- Por último, compara esta clave hasheada con la que ha obtenido del cliente tras solicitarla, también hasheada. Si ambas claves coinciden, implica que el cliente puede acceder al chat, por lo que el servidor le da acceso. La siguiente imagen muestra un ejemplo del proceso llevado a cabo en el servidor:



- Se observa en primer lugar que el servidor obtiene la clave de acceso al chat y comprueba posteriormente que los hashes de las claves de acceso al chat de cada uno de los usuarios es válido, por lo que ambos se pueden conectar sin problemas al chat.

En el caso del cliente, el funcionamiento es muy similar:



- El cliente, si recibe el mensaje @username por parte del servidor, es decir, el servidor requiere sus credenciales, este se las envía.
- Para ello, obtiene la clave del chat del archivo key correspondiente ("*criptochat.key*") y procede a hashearla para enviársela al servidor, junto al username y su clave pública.