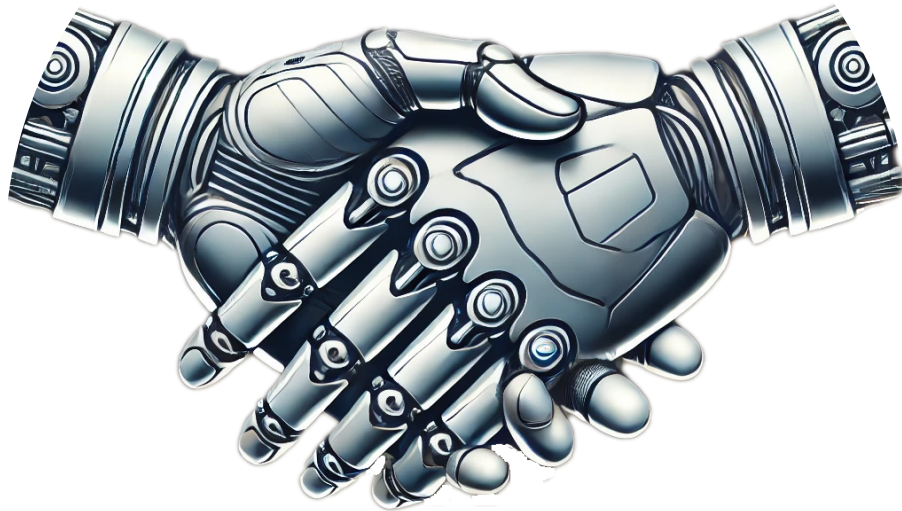


# SSH Handshake



Federico Williamson

Leonel Castinelli

Seguridad Informática 2024

# Índice

I. Introducción .....	3
I.1. ¿Qué es SSH? .....	3
I.2. Problema que soluciona .....	3
I.3. Principales aplicaciones .....	3
I.4. ¿Qué es el Handshake SSH? .....	3
II. El protocolo .....	3
II.1. Intercambio de Versiones ([1] Sección 4.2) .....	3
II.2. Intercambio de algoritmos (KEXInit) ([1] Sección 7.1) .....	4
II.3. Intercambio de claves Diffie Hellman([2] Sección 8) .....	5
II.4. Solicitud de servicio[3] .....	6
II.5. Solicitud de Conexion[3] .....	6
II.6. Ejecucion[2] .....	7
III. Apéndice I .....	8
Bibliografía .....	8

# I. Introducción

## I.1. ¿Qué es SSH?

**SSH** (o **Secure SHell**) es el nombre de un protocolo y del programa que lo implementa cuya principal función es el **acceso remoto** a un servidor por medio de un **canal seguro** en el que toda la información está cifrada. El puerto TCP asignado de forma predeterminada es el puerto **22**, asignado por la IANA [4].

## I.2. Problema que soluciona

Ante la comunicación entre dos equipos informáticos, existe la posible amenaza de que un equipo malicioso esté escuchando la comunicación legítima y busque robar información que se esté transmitiendo o incluso intente robar la identidad de uno de los actores en la comunicación para inducir mensajes maliciosos en la misma.

## I.3. Principales aplicaciones

Ya que este protocolo prueba ser de extrema utilidad, se aplica en las siguientes ocasiones:

- **Acceso Remoto Seguro:** Permite controlar equipos informáticos de forma segura a través de internet.
- **Reenvío de puertos:** Permite el acceso a servicios restringidos en redes privadas por medio de reenvío de puertos.
- **Túneles SSH:** Redirección de tráfico en la red a través de un canal seguro, útil para evitar restricciones de red y proteger datos sensibles.
- **Transferencia segura de archivos:** Los protocolos SCP y SFTP utilizan el protocolo SSH para la transferencia de archivos de forma segura.

## I.4. ¿Qué es el Handshake SSH?

Es un proceso por el cual se establece una conexión segura entre un cliente y un servidor. Se produce tras la primera comunicación con el servidor. Consta de **5 pasos**:

1. Intercambio de Versiones.
2. Intercambio de Claves (KEX).

3. Inicialización de intercambio Diffie-Hellman (DH).
4. Respuesta de DH.
5. Nuevas Claves (NewKeys).

# II. El protocolo

## II.1. Intercambio de Versiones ([1] Sección 4.2)

### II.1.1. Cadena de Version ([1] Sección 4.2)

La cadena de versión de un participante en la comunicación se encuentra en el siguiente formato:

```
<SSH-protocolversion>-<softwareversion>  
<comments>CRLF
```

Con esta cadena en este formato ambos participantes podrán intercambiar la versión del protocolo y la versión de software que están utilizando. Siempre deben terminar con CRLF<sup>1</sup> [2].

Ambas partes, **cliente** y **servidor** deben enviar sus cadenas de versión. Las cuales llamaremos  $V_C$  a la cadena de version del **cliente** y  $V_S$  a la del **servidor**.

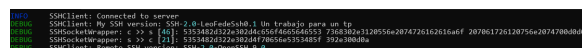


Figura 1: Intercambio de versiones

En la Figura 1 podemos observar como enviamos nuestra versión y recibimos la versión del servidor.

### II.1.2. Protocolo binario de paquetes[2]

Cada paquete se encuentra en el siguiente formato:

```
uint32    packet_length  
byte      padding_length  
byte[n1]  payload; n1 = packet_length -  
padding_length - 1  
byte[n2]   random    padding; n2 =  
padding_length  
byte[m]    mac (Message Authentication Code  
- MAC); m = mac_length
```

Donde cada parte representa:

<sup>1</sup>CRLF es un \r seguido de \n

- **packet\_length**: Es la longitud del paquete en bytes, sin contar “mac” o el mismo campo “packet\_length”.
- **padding\_length**: longitud del “random padding” en bytes.
- **payload**: La parte útil del paquete.
- **random\_padding**: padding de longitud arbitraria, para que la longitud total (packet\_length + padding\_length + payload + random\_padding) sea un múltiplo de  $8^2$ , el padding tiene que ser como mínimo de 4 y máximo de 255. Además sirve para introducir ruido al mensaje, confundiendo a receptores ilegítimos de la comunicación.
- **MAC**: Código de Autorización de Mensajes, si se negoció la autenticación de mensajes, este campo contiene los bytes de MAC. Inicialmente el algoritmo de MAC es **none**.

El como se encodean varios de los tipos de datos como uint32 string mpint esta detallado en [5].

Los numeros asignados a los tipo de mensaje estan detallados en [6].

### II.1.2.1. MAC

Una funcionalidad de protección, de extrema utilidad en lo que concierne a la seguridad de la comunicación, que ofrece SSH en su modelo de paquetes, es MAC: Message Authorization Code (Código de Autorización de Mensajes).

Este código es un algoritmo de hash del contenido del mensaje no encriptado, concatenado con el número de secuencia del paquete.

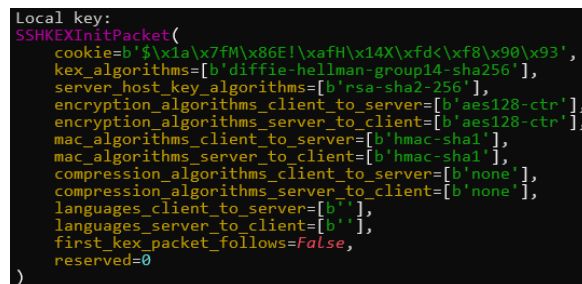
Esto es extremadamente útil en evitar replay attacks, entre otros.

## II.2. Intercambio de algoritmos (KEXInit) ([1] Sección 7.1)

SSH es un protocolo muy versátil, permite utilizar múltiples algoritmos para cada parte de la comunicación.

A efectos de determinar el algoritmo que se va a utilizar para esta conexión, es necesario

que el servidor y el cliente intercambien los algoritmos que soportan.

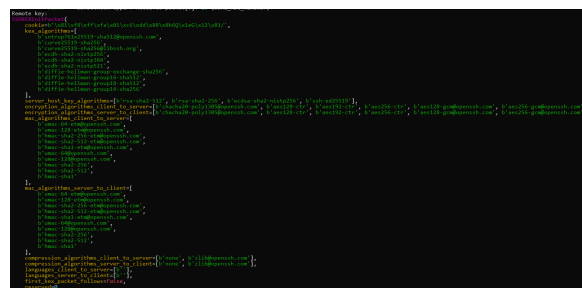


```
Local key:
SSHKEXInitPacket(
  cookie=b'$\x1a\x7fM\x86E!\xafH\x14X\xfd<\xf8\x90\x93',
  kex_algorithms=[b'diffie-hellman-group14-sha256'],
  server_host_key_algorithms=[b'rsa-sha2-256'],
  encryption_algorithms_client_to_server=[b'aes128-ctr'],
  encryption_algorithms_server_to_client=[b'aes128-ctr'],
  mac_algorithms_client_to_server=[b'hmac-sha1'],
  mac_algorithms_server_to_client=[b'hmac-sha1'],
  compression_algorithms_client_to_server=[b'none'],
  compression_algorithms_server_to_client=[b'none'],
  languages_client_to_server=[b''],
  languages_server_to_client=[b''],
  first_kex_packet_follows=False,
  reserved=0
)
```

Figura 2: Kex Init del cliente enviado a remoto

Nosotros mandamos el paquete de tipo KEX init mostrado en la Figura 2 al servidor.

Aca manifestamos los algoritmos que soportamos para cada categoria, en orden descendente de preferencia. Para forzar el uso de ciertos algoritmos y simplificar la implementación, enviamos listas de un único elemento.



```
Local key:
SSHKEXInitPacket(
  cookie=b'$\x1a\x7fM\x86E!\xafH\x14X\xfd<\xf8\x90\x93',
  kex_algorithms=[b'diffie-hellman-group14-sha256'],
  server_host_key_algorithms=[b'rsa-sha2-256'],
  encryption_algorithms_client_to_server=[b'aes128-ctr'],
  encryption_algorithms_server_to_client=[b'aes128-ctr'],
  mac_algorithms_client_to_server=[b'hmac-sha1'],
  mac_algorithms_server_to_client=[b'hmac-sha1'],
  compression_algorithms_client_to_server=[b'none'],
  compression_algorithms_server_to_client=[b'none'],
  languages_client_to_server=[b''],
  languages_server_to_client=[b''],
  first_kex_packet_follows=True,
  reserved=0
)
```

Figura 3: Kex Init recibido del servidor

En la Figura 3 se puede ver la respuesta del servidor, mostrando todos los algoritmos que soporta para cada categoria.

El protocolo SSH ([2]) define que se usa el primero en comun, llenando en orden de izquierda a derecha.

Es posible que en cada direccion de la comunicación<sup>3</sup> se usen algirtmos distintos.

En este momento, hambos cliente y servidor saben que algoritmo van a usar para cada caso (lo pueden inferir sin necesidad de transmitirlo).

A los **payload** de KEX Init que envian el **cliente** y el **servidor** seran utilizados luego y los llamaremos  $I_C$  y  $I_S$  respectivamente.

<sup>2</sup>O multiplo del tamaño de bloque del algoritmo de cifrado si este esta en efecto

<sup>3</sup>cliente a servidor, servidor a cliente

## II.3. Intercambio de claves Diffie Hellman([2] Sección 8)

Nosotros, elegimos usar el algoritmo de diffie-hellman-group14-sha256 como algoritmo de intercambio.

Esto significa que:

- Vamos a hacer un intercambio de tipo DH.
- Vamos a usar primos del grupo 14.
- Vamos a usar sha256 como algoritmo de hash.

El objetivo de KEX DH es que ambos participantes puedan mutuamente acordar en una clave, sin que esta sea derivable por un tercero que esta observando la comunicación.

Como funciona esto?

1. Se acuerdan  $p$  y  $g$ .  $p$  es un primo publico muy grande, y  $g$  su generador.

Como estamos usando group14, usamos el primos del grupo 14, que esta definido en [7]-3.

C (cliente) genera un número aleatorio  $x$  entre 1 y  $q^4$  y computa el valor:

$$e = g^x \mod p \quad (1)$$

En la Ecuación 1,  $e$  es la **clave pública del cliente** y a  $x$  lo llamaremos **clave privada del cliente**.

Figura 4: Generacion de clave privada y clave publica por parte del cliente.

S (servidor) genera su propio numero aleatorio  $y$  entre 0 y  $q$  y computa:

$$f = g^y \mod p \quad (2)$$

En la Ecuación 2 se puede observar la **clave pública del servidor**  $f$  y la **clave privada del servidor**  $y$ .

Figura 5: Generacion de clave privada y clave publica por parte del servidor.

S al recibir  $e$ , computa el **secreto compartido**  $K$  de la siguiente forma:

$$K = e^y \mod p \quad (3)$$

Una vez que S recibe  $x$  del C, tiene toda la informacion para derivar la clave y manda **SSH\_NEW\_KEYS**. Cuando C recibe  $f$  de S, puede calcular el secreto compartido y emitir **SSH\_NEW\_KEYS**.

Figura 6: El secreto compartido

En este momento ambos tienen suficiente información para terminar de derivar la clave compartida  $K$ , pero con solo  $K$ , resulta insuficiente para la comunicación.

Es necesario seguir los siguientes pasos: **Computar el hash H:**

$$H = \text{hash}(V_C \parallel V_S \parallel I_C \parallel I_S \parallel K_S \parallel e \parallel f \parallel K)$$

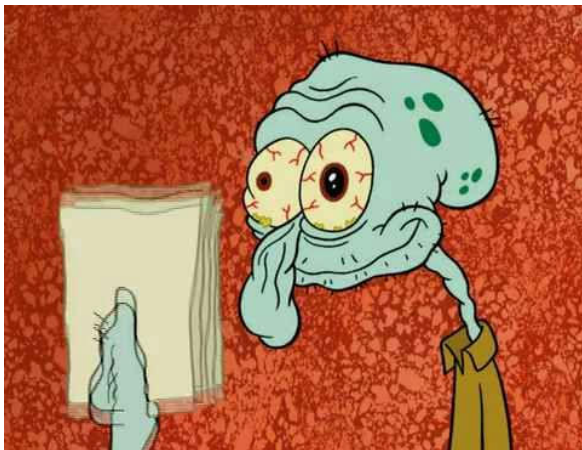
Donde:

- hash es sha256, por el algoritmo de intercambio que elegimos
- $V_C$  Es la version de ssh cliente (sin CRLF)
- $V_S$  Version de ssh del seervidor (sin CRLF)
- $I_C$  El Payload del KEX\_INIT enviado por el cliente
- $I_S$  El Payload del KEX\_INIT enviado por el servidor
- $K_S$  La Host-Key del servidor<sup>5</sup>

<sup>4</sup> $q$  hace referencia al orden del subgrupo, que en la práctica, no se necesita calcular explícitamente, es aproximadamente  $\lfloor \frac{p}{2} \rfloor$

- Un dato interesante de esto, es que al incluir el payload de  $I_C$  e  $I_S$ , que como anteriormente cubrimos en Sección II.2, contienen un cookie que contiene un valor random. Esto hace que sea completamente imposible para un atacante determinar  $H$  en sesiones repetidas.

- Primer IV C  $\rightarrow$  S:  
HASH( $K \parallel H \parallel 'A' \parallel \text{session\_id}$ )<sup>7</sup>
- Primer IV C  $\leftarrow$  S:  
HASH( $K \parallel H \parallel 'B' \parallel \text{session\_id}$ )
- Clave de Encriptación C  $\rightarrow$  S:  
HASH( $K \parallel H \parallel 'C' \parallel \text{session\_id}$ )
- Clave de Encriptación C  $\leftarrow$  S:  
HASH( $K \parallel H \parallel 'D' \parallel \text{session\_id}$ )
- Clave de MAC C  $\rightarrow$  S:  
HASH( $K \parallel H \parallel 'E' \parallel \text{session\_id}$ )
- Clave de MAC C  $\leftarrow$  S:  
HASH( $K \parallel H \parallel 'F' \parallel \text{session\_id}$ )



<sup>5</sup>Este es un valor adicional que envia el servidor cuando envia  $f$ , con el proposito de autenticarse.

<sup>6</sup>La session id es el primer  $H$  que se acuerda entre  $C$ , y  $S$ , y no cambia cuando hay un reset de keys.

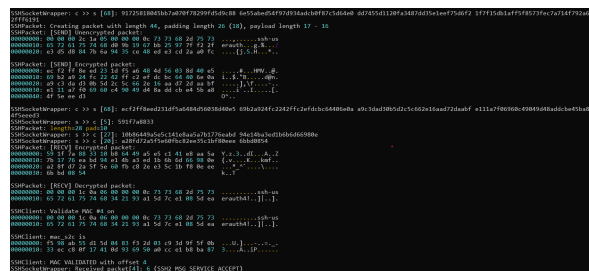
<sup>7</sup> $K$  esta encodeado como `mpint`,  $A$  es el byte encodeado en ASCII y `session id` no esta encodeado

Para probar el cliente, con el servidor de OpenSSH, es posible ejecutarlo en modo de debug. Una de las funcionalidades del modo debug, es una vez que se inicia la comunicación encriptada, se envía un paquete de tipo 02 (SSH\_IGNORE) con texto: markus. Este mensaje conocido es extremadamente útil para debuggear problemas.

[illegible]

En la Figura 8 se puede observar como tambien enviamos un paquete ignorable con markus al servidor para verificar que no produce errores.

Antes de proceder, requerimos que el servidor nos permita autenticar usuarios.



Una vez que tenemos permisos para autenticar, podemos solicitar una conexión



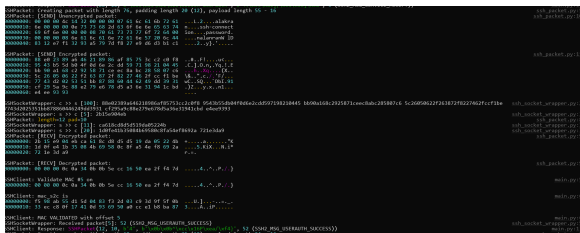


Figura 10: Solicitud de conexion con usuario y contraseña.

Es importante señalar que, aunque aquí se muestra la contraseña en texto claro, en realidad viaja cifrada.

Sin embargo, resulta algo controvertido enviar la contraseña directamente; sería preferible transmitir solo el hash correspondiente, lo que reduciría el riesgo en caso de que un servidor malicioso almacenara las contraseñas para luego utilizarlas en un ataque de fuerza bruta.

Una vez hecho esto, si la autenticación es exitosa, recibimos varios mensajes del servidor:

1. HostKeys: Autenticación del servidor
2. Confirmación de la apertura del canal

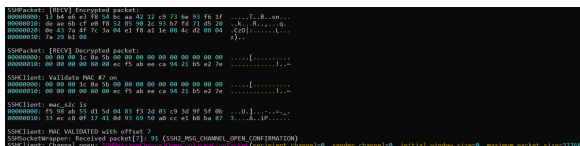


Figura 11: Confirmacion de apertura del canal.

Y estamos en posición de enviar un comando

## II.6. Ejecucion[2]

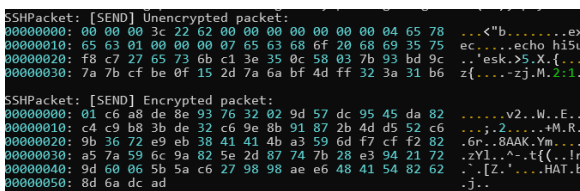


Figura 12: Ejecucion del comando

Enviamos un comando de EXEC con payload echo hi

Y recibimos varios mensajes del servidor:

- SSH2\_MSG\_CHANNEL\_WINDOW\_ADJUST: Incrementando el tamaño maximo de un paquete.

- SSH2\_MSG\_CHANNEL\_EXTENDED\_DATA: Con informacion del path, y otras variables de entorno
- SSH2\_MSG\_CHANNEL\_REQUEST: Pide abrir un canal de comunicacion hacia nosotros
- SSH2\_MSG\_CHANNEL\_DATA: Optimisticamente envia la respuesta a nuestra peticion antes que aceptemos abrir el canal.

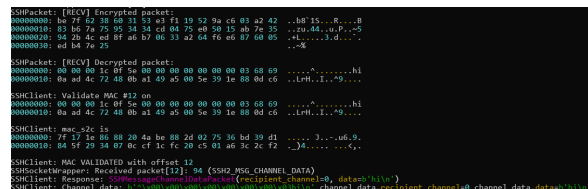


Figura 13: Respuesta de nuestra ejecucion

### III. Apéndice I

En Figura 14 se puede observar los pasos y mensajes que intercambian el cliente y servidor durante el handshake SSH para el «happy path», es decir, asumiendo que todo funciona bien y es aceptado.

## Bibliografía

- [1] «ietf: RFC 4253. The Secure Shell (SSH) Transport Layer Protocol». 2006.
- [2] «ietf: RFC 4254. The Secure Shell (SSH) Connection Protocol». 2006.
- [3] «ietf: RFC 4252. The Secure Shell (SSH) Authentication Protocol». 2006.
- [4] «The story of the SSH port is 22. — ssh.com». 2017.
- [5] «ietf: RFC 4251. The Secure Shell (SSH) Protocol Architecture». 2006.
- [6] «ietf: RFC 4250. The Secure Shell (SSH) Protocol Assigned Numbers ». 2006.
- [7] «ietf: RFC 3526. More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)». 2003.

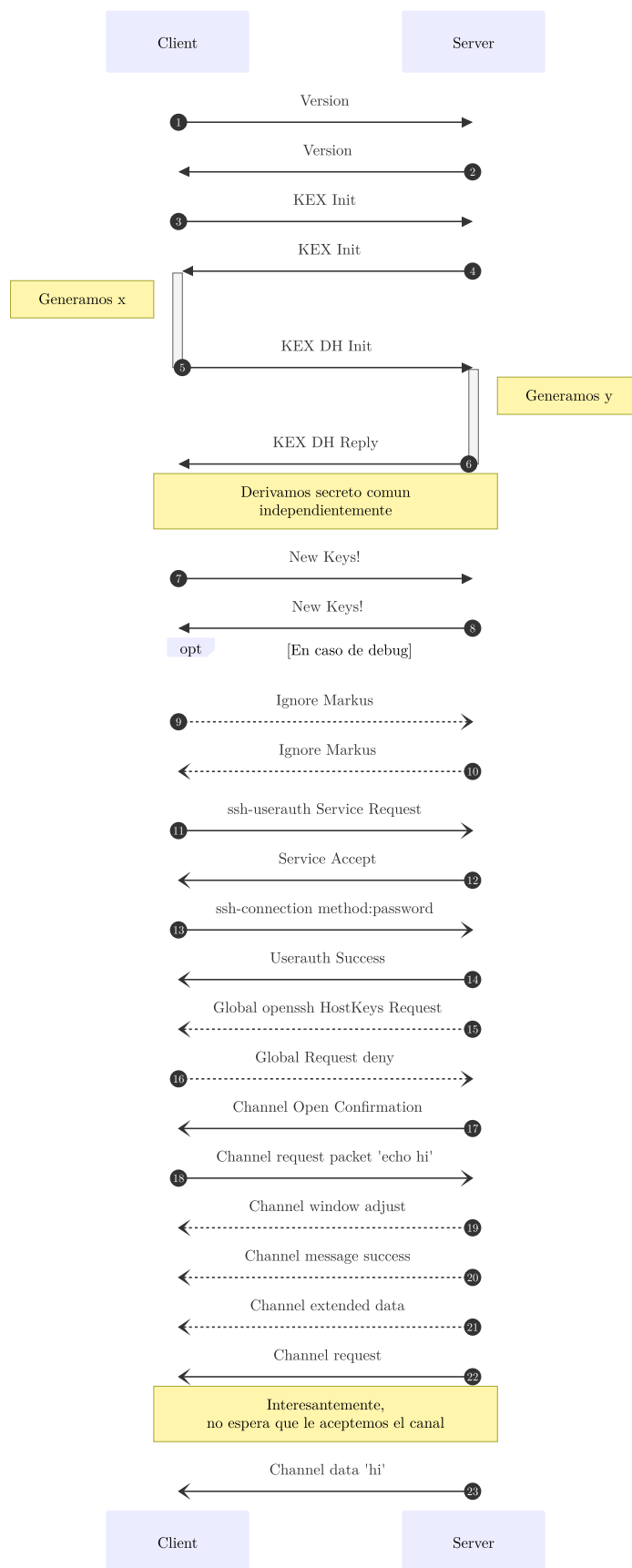


Figura 14: Pasos del intercambio SSH y ejecución de un comando.