

# Informe Laboratorio 5

## Sección 3

Diego Tapia  
e-mail: diego.tapia6@mail\_udp.cl

Noviembre de 2025

# Índice

|   |           |
|---|-----------|
| <b>Descripción de actividades</b>   | <b>3</b>  |
| <b>1. Desarrollo (Parte 1)</b>  | <b>6</b>  |
| 1.1. Códigos de cada Dockerfile . . . . .   | 6         |
| 1.1.1. C1 . . . . .   | 6         |
| 1.1.2. C2 . . . . .   | 6         |
| 1.1.3. C3 . . . . .   | 6         |
| 1.1.4. C4/S1 . . . . .  | 7         |
| 1.2. Creación de las credenciales para S1 . . . . .   | 7         |
| 1.3. Tráfico generado por C1, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado) . . . . .            | 8         |
| 1.4. Tráfico generado por C2, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado) . . . . .            | 9         |
| 1.5. Tráfico generado por C3, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado) . . . . .            | 10        |
| 1.6. Tráfico generado por C4 (iface lo), detallando tamaño paquetes del flujo y el HASSH respectivo (detallado) . . . . . | 11        |
| 1.7. Compara la versión de HASSH obtenida con la base de datos para validar si el cliente corresponde al mismo . . . . .  | 12        |
| 1.8. Tipo de información contenida en cada uno de los paquetes generados en texto plano . . . . .                         | 13        |
| 1.8.1. C1 . . . . .   | 13        |
| 1.8.2. C2 . . . . .   | 13        |
| 1.8.3. C3 . . . . .   | 13        |
| 1.8.4. C4/S1 . . . . .  | 14        |
| 1.9. Diferencia entre C1 y C2 . . . . .   | 14        |
| 1.10. Diferencia entre C2 y C3 . . . . .  | 14        |
| 1.11. Diferencia entre C3 y C4 . . . . .  | 15        |
| <b>2. Desarrollo (Parte 2)</b>  | <b>16</b> |
| 2.1. Identificación del cliente SSH con versión “?” . . . . .   | 16        |
| 2.2. Replicación de tráfico al servidor (paso por paso) . . . . .   | 17        |
| <b>3. Desarrollo (Parte 3)</b>  | <b>18</b> |
| 3.1. Replicación del KEI con tamaño menor a 300 bytes (paso por paso) . . . . .   | 18        |
| <b>4. Desarrollo (Parte 4)</b>  | <b>21</b> |
| 4.1. Explicación OpenSSH en general . . . . .   | 21        |
| 4.2. Capas de seguridad en OpenSSH . . . . .  | 21        |
| 4.3. Identificación de principios que no se cumplen completamente . . . . .   | 23        |
| 4.4. Conclusiones y comentarios . . . . .   | 23        |

## Descripción de actividades

Para este último laboratorio, se solicita trabajar con Docker y el protocolo SSH, a fin de poder entender el concepto de criptografía asimétrica y firmas digitales.

Para lo anterior deberá:

- Crear 4 contenedores en Docker por medio de un DockerFile, donde cada uno tendrá el siguiente SO: Ubuntu 16.10, Ubuntu 18.10, Ubuntu 20.10 y Ubuntu 22.10 a los cuales se llamarán C1, C2, C3 y C4 respectivamente.  
El equipo con Ubuntu 22.10 también será utilizado como S1.
- Para cada uno de ellos, deberá instalar el cliente openSSH disponible en los repositorios de apt, y para el equipo S1 deberá también instalar el servidor openSSH.
- En S1 deberá crear el usuario “**prueba**” con contraseña “**prueba**”, para acceder a él desde los clientes por el protocolo SSH.
- En total serán 4 escenarios, donde cada uno corresponderá a los siguientes equipos:
  - C1 → S1
  - C2 → S1
  - C3 → S1
  - C4 → S1

Pasos:

1. Para cada uno de los 4 escenarios, solo deberá establecer la conexión y no realizar ningún otro comando que pueda generar tráfico (como muestra la Figura). Deberá capturar el tráfico de red generado y analizar el patrón de tráfico generado por cada cliente. De esta forma podrá obtener una huella digital para cada cliente a partir de su tráfico.

Indique el tamaño de los paquetes del flujo generados por el cliente y el contenido asociado a cada uno de ellos. Indique qué información distinta contiene el escenario siguiente (diff incremental). El objetivo de este paso es identificar claramente los cambios entre las distintas versiones de ssh.

2. Para poder identificar que el usuario efectivamente es el informante, éste utilizará una versión única de cliente. ¿Con qué cliente SSH se habrá generado el siguiente tráfico?

| Protocol | Length | Info  |
|----------|--------|---|
| TCP      | 74     | 34328 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=14 |
| TCP      | 66     | 34328 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0  |
| SSHv2    | 85     | Client: Protocol (SSH-2.0-OpenSSH_?)          |
| TCP      | 66     | 34328 → 22 [ACK] Seq=20 Ack=42 Win=64256 Len= |
| SSHv2    | 1578   | Client: Key Exchange Init                     |
| TCP      | 66     | 34328 → 22 [ACK] Seq=1532 Ack=1122 Win=64128  |
| SSHv2    | 114    | Client: Elliptic Curve Diffie-Hellman Key Exc |
| TCP      | 66     | 34328 → 22 [ACK] Seq=1580 Ack=1574 Win=64128  |
| SSHv2    | 82     | Client: New Keys                              |
| SSHv2    | 110    | Client: Encrypted packet (len=44)             |
| TCP      | 66     | 34328 → 22 [ACK] Seq=1640 Ack=1618 Win=64128  |
| SSHv2    | 126    | Client: Encrypted packet (len=60)             |
| TCP      | 66     | 34328 → 22 [ACK] Seq=1700 Ack=1670 Win=64128  |
| SSHv2    | 150    | Client: Encrypted packet (len=84)             |
| TCP      | 66     | 34328 → 22 [ACK] Seq=1784 Ack=1698 Win=64128  |
| SSHv2    | 178    | Client: Encrypted packet (len=112)            |
| TCP      | 66     | 34328 → 22 [ACK] Seq=1896 Ack=2198 Win=64128  |

Figura 1: Tráfico generado del informante

Replique este tráfico generado en la imagen. Debe generar el tráfico con la misma versión resaltada en azul. Recuerde que toda la información generada es parte del sw, por lo tanto usted puede modificar toda la información.

3. Para que el informante esté seguro de nuestra identidad, nos pide que el patrón del tráfico de nuestro server también sea modificado, hasta que el Key Exchange Init del server sea menor a 300 bytes. Indique qué pasos realizó para lograr esto.

---

|       |                                   |
|-------|-----------------------------------|
| TCP   | 66 42350 → 22 [ACK] Seq=2 Ack=    |
| TCP   | 74 42398 → 22 [SYN] Seq=0 Win=    |
| TCP   | 74 22 → 42398 [SYN, ACK] Seq=0    |
| TCP   | 66 42398 → 22 [ACK] Seq=1 Ack=    |
| SSHv2 | 87 Client: Protocol (SSH-2.0-C)   |
| TCP   | 66 22 → 42398 [ACK] Seq=1 Ack=    |
| SSHv2 | 107 Server: Protocol (SSH-2.0-C)  |
| TCP   | 66 42398 → 22 [ACK] Seq=22 Ack=   |
| SSHv2 | 1570 Client: Key Exchange Init    |
| TCP   | 66 22 → 42398 [ACK] Seq=42 Ack=   |
| SSHv2 | 298 Server: Key Exchange Init     |
| TCP   | 66 42398 → 22 [ACK] Seq=1526 Ack= |

Figura 2: Captura del Key Exchange

4. Tomando en cuenta lo aprendido en este laboratorio, así como en los anteriores, explique el protocolo OpenSSH y las diferentes capas de seguridad que son parte del protocolo para garantizar los principios de seguridad de la información, integridad, confidencialidad, disponibilidad, autenticidad y no repudio. Es importante que sea muy específico en el objetivo del principio en el protocolo. En caso de considerar que alguno de los principios no se cumple, justifique su razonamiento. Es fundamental que su análisis se base en el tráfico SSH interceptado.

## 1. Desarrollo (Parte 1)

### 1.1. Códigos de cada Dockerfile

#### 1.1.1. C1

```
↳ dockerfile.C1 > ...
1  FROM ubuntu:16.04
2
3  RUN apt-get update && \
4      apt-get install -y openssh-client iproute2 tcpdump net-tools && \
5      apt-get clean
6
7  CMD ["/bin/bash"]
8
```

Figura 3: Código C1: Ubuntu 16.04

#### 1.1.2. C2

```
↳ dockerfile.C2 > ...
1  FROM ubuntu:18.04
2
3  RUN apt-get update && \
4      apt-get install -y openssh-client iproute2 tcpdump net-tools && \
5      apt-get clean
6
7  CMD ["/bin/bash"]
8
```

Figura 4: Código C2: Ubuntu 18.04

#### 1.1.3. C3

```
↳ dockerfile.C3 > ...
1  FROM ubuntu:20.04
2
3  RUN apt-get update && \
4      apt-get install -y openssh-client iproute2 tcpdump net-tools && \
5      apt-get clean
6
7  CMD ["/bin/bash"]
8
```

Figura 5: Código C3: Ubuntu 20.04

### 1.1.4. C4/S1

```
↳ dockerfile.C4 > ...
1  FROM ubuntu:22.04
2
3  RUN apt-get update && \
4      apt-get install -y openssh-client openssh-server iproute2 tcpdump net-tools && \
5      mkdir /var/run/sshd && \
6      apt-get clean
7
8  CMD ["/bin/bash"]
9
```

Figura 6: Código C4/S1: Ubuntu 24.04

## 1.2. Creación de las credenciales para S1

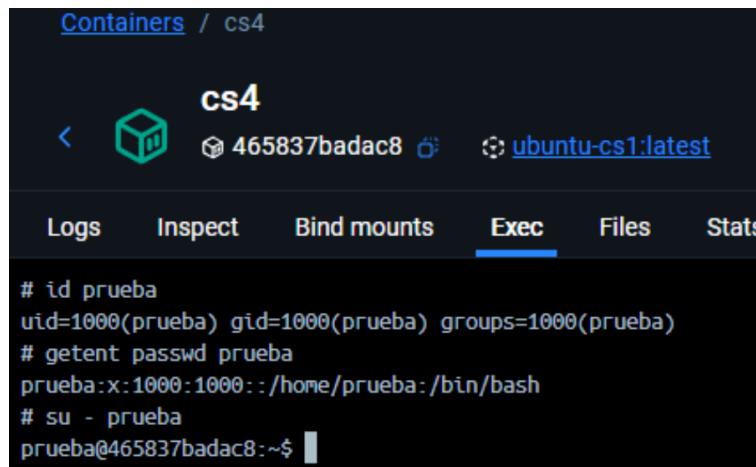


Figura 7: Creación de Credenciales

### 1.3. Tráfico generado por C1, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

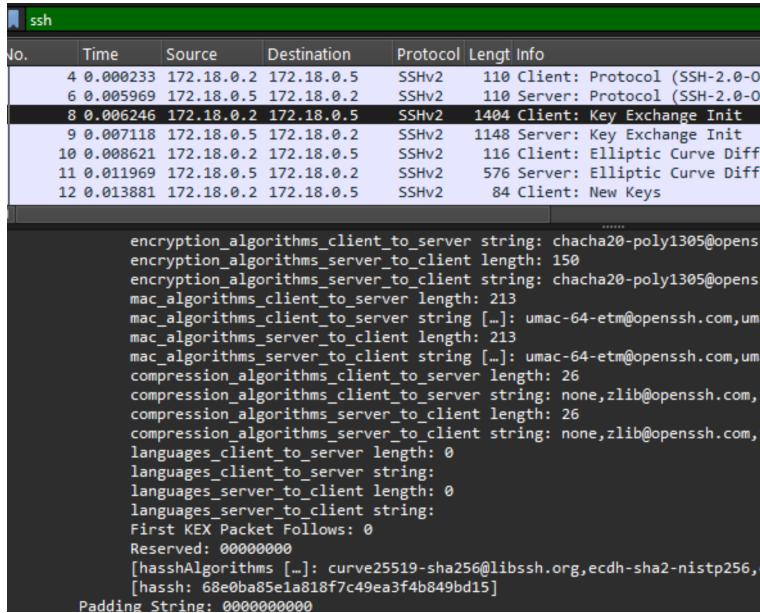


Figura 8: Tráfico C1

En la Figura 8 se observa la captura del tráfico SSH generado por el contenedor C1 hacia el servidor S1. El flujo corresponde al *handshake* inicial del protocolo SSH-2.0, donde el cliente con dirección IP 172.18.0.2 (OpenSSH 7.2p2 sobre Ubuntu 16.04) se comunica con el servidor 172.18.0.5.

Los primeros paquetes, de aproximadamente 110 bytes, contienen los *banners* de cliente y servidor en texto plano, es decir, las cadenas de identificación del protocolo (SSH-2.0) junto con la versión específica de OpenSSH utilizada en cada extremo. Posteriormente, los paquetes de 1404 bytes (cliente) y 1148 bytes (servidor) corresponden a los mensajes `SSH_MSG_KEXINIT`, en los cuales ambas partes anuncian sus listas de algoritmos soportados para intercambio de claves (*Key Exchange*), cifrado simétrico (*Ciphers*), integridad (*MACs*) y compresión (*Compression*). Estas listas aún se transmiten en texto plano, por lo que pueden ser inspeccionadas directamente en la captura.

A partir de la información contenida en el mensaje `KEXINIT` del cliente es posible calcular la huella digital o *HASSH* asociada a C1, obteniéndose el valor `68e0ba85e1a818f7c49ea3f4b849bd15`. Este identificador permite relacionar el tráfico observado con una versión concreta de OpenSSH (en este caso, 7.2p2). Finalmente, los paquetes de 116, 576 y 84 bytes corresponden respectivamente a `KEX ECDH INIT`, `KEX ECDH REPLY + New Keys` y `New Keys` del cliente. Tras este intercambio, Wireshark clasifica el resto del flujo como *Encrypted packet*, indicando que el

canal ya se encuentra completamente cifrado y que el contenido de los mensajes de aplicación no es visible en texto plano.

#### 1.4. Tráfico generado por C2, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

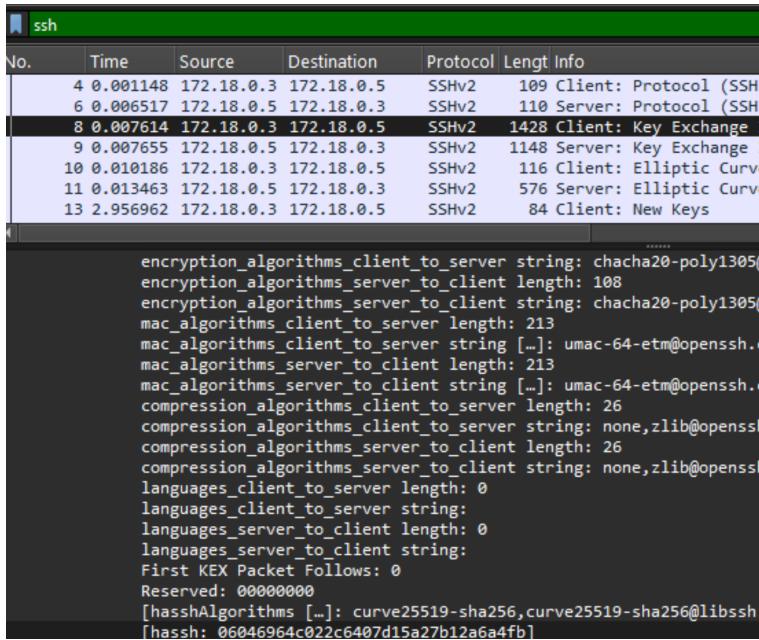


Figura 9: Tráfico C2

En la Figura 9 se observa la captura del tráfico SSH generado por el contenedor C2 hacia el servidor S1. En este caso, el cliente corresponde a la dirección IP 172.18.0.3 y utiliza la versión SSH-2.0-OpenSSH 7.6p1 sobre Ubuntu 18.04, mientras que el servidor mantiene la versión OpenSSH 8.2p1 con IP 172.18.0.5. El flujo mostrado corresponde nuevamente al *handshake* inicial del protocolo SSH-2.0.

Los primeros paquetes del flujo, de aproximadamente 109 bytes (banner del cliente) y 110 bytes (banner del servidor), contienen en texto plano las cadenas de identificación del protocolo y las versiones de OpenSSH de ambos extremos. A continuación, los paquetes de 1428 bytes (cliente) y 1148 bytes (servidor) corresponden a los mensajes `SSH_MSG_KEXINIT`, donde se anuncian las listas de algoritmos soportados para intercambio de claves (*KexAlgorithms*), cifrado simétrico (*Ciphers*), integridad (*MACs*) y compresión (*Compression*). Al igual que en C1, estas listas se observan en texto plano en el panel inferior de Wireshark, permitiendo inspeccionar directamente los algoritmos propuestos por C2.

A partir de la información contenida en el `KEXINIT` del cliente se calcula la huella digital *HASSH* asociada al flujo, obteniéndose para C2 el valor 06046964c022c6407d15a27b12a6a4fb.

Este identificador se asocia a la versión OpenSSH 7.6p1, lo que permite vincular el tráfico capturado con un cliente específico. Finalmente, los paquetes de 116, 576 y 84 bytes representan KEX ECDH INIT, KEX ECDH REPLY + New Keys y New Keys del cliente, respectivamente. Tras este intercambio, el resto del tráfico pasa a mostrarse como *Encrypted packet*, indicando que el canal ya se encuentra cifrado y que los datos de aplicación no son visibles en texto plano.

### 1.5. Tráfico generado por C3, detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

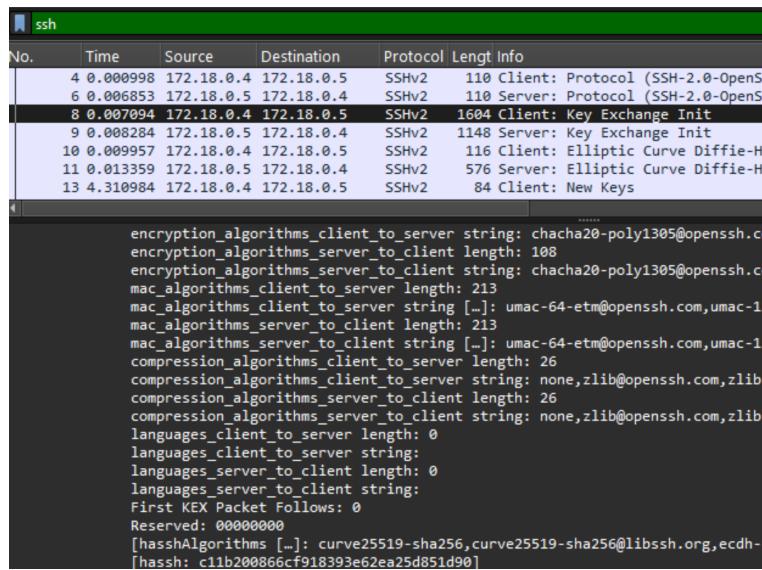


Figura 10: Tráfico C3

En la Figura 10 se muestra la captura del tráfico SSH generado por el contenedor C3 hacia el servidor S1. En esta sesión, el cliente tiene dirección IP 172.18.0.4 y utiliza la versión SSH-2.0-OpenSSH 8.2p1 (Ubuntu 20.04), mientras que el servidor también ejecuta OpenSSH 8.2p1 con IP 172.18.0.5. Nuevamente se observa el *handshake* inicial del protocolo SSH-2.0.

Los primeros paquetes del flujo, de 110 bytes tanto para el cliente como para el servidor, corresponden a los *banners* de protocolo enviados en texto plano, donde se indica la versión de SSH y de OpenSSH en cada extremo. Posteriormente, los paquetes de 1604 bytes (cliente) y 1148 bytes (servidor) representan los mensajes **SSH\_MSG\_KEXINIT**. En ellos se anuncian, en texto plano, las listas de algoritmos disponibles para intercambio de claves (*KexAlgorithms*), cifrado simétrico (*Ciphers*), integridad (*MACs*) y compresión (*Compression*). En el panel inferior de Wireshark se aprecian estas listas completas, lo que permite comparar directamente la propuesta de algoritmos de C3 con la de los clientes anteriores.

A partir del contenido del mensaje KEXINIT del cliente se calcula la huella digital *HASSH* asociada al flujo, obteniéndose para C3 el valor `c11b200866cf918393e62ea25d851d90`. Este fingerprint identifica la combinación específica de algoritmos ofrecida por OpenSSH 8.2p1 y coincide posteriormente con la observada en C4. Finalmente, los paquetes de 116, 576 y 84 bytes corresponden a KEX ECDH INIT, KEX ECDH REPLY + New Keys y New Keys del cliente, respectivamente. A partir de ese punto, Wireshark muestra el resto de los paquetes como *Encrypted packet*, indicando que el canal SSH ya se encuentra cifrado y que el contenido de los datos de usuario deja de ser visible en texto plano.

## 1.6. Tráfico generado por C4 (iface lo), detallando tamaño paquetes del flujo y el HASSH respectivo (detallado)

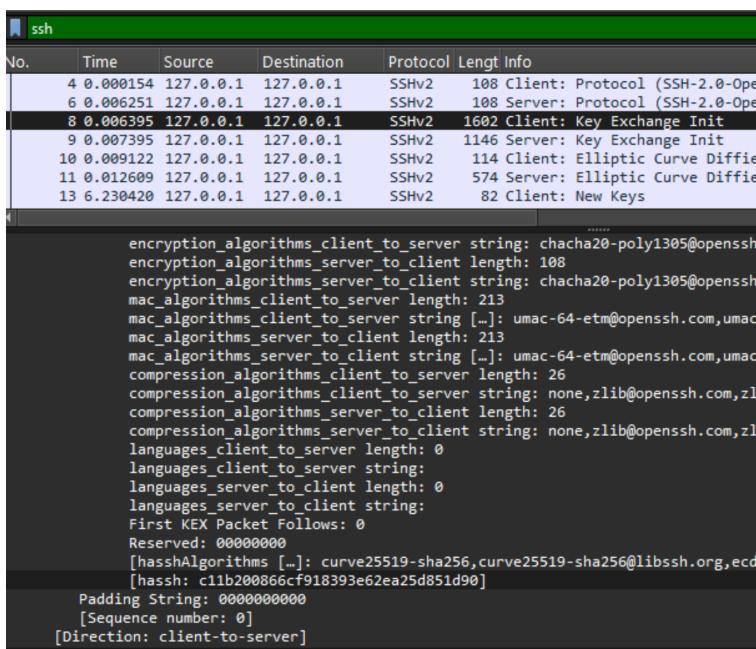


Figura 11: Tráfico C4

En la Figura 11 se presenta la captura del tráfico SSH correspondiente al contenedor C4, el cual actúa simultáneamente como cliente y servidor (C4/S1). En este escenario la comunicación se realiza sobre la interfaz de *loopback*, por lo que tanto el origen como el destino utilizan la dirección IP 127.0.0.1. Tanto el cliente como el servidor ejecutan la versión SSH-2.0-OpenSSH 8.2p1 sobre Ubuntu 22.04, por lo que se trata de una conexión SSH establecida dentro de la misma máquina.

Los primeros paquetes del flujo, de 108 bytes, corresponden a los *banners* de cliente y servidor en texto plano, donde se indica la versión del protocolo SSH y de OpenSSH utilizada en ambos extremos. A continuación, los paquetes de 1602 bytes (cliente) y 1146 bytes (servidor)

representan los mensajes `SSH_MSG_KEXINIT`, en los cuales C4 anuncia las listas de algoritmos disponibles para intercambio de claves (*KexAlgorithms*), cifrado simétrico (*Ciphers*), integridad (*MACs*) y compresión (*Compression*). En el panel inferior de Wireshark se aprecia que estas listas son prácticamente idénticas a las observadas en C3, lo que refleja que ambos utilizan la misma versión de OpenSSH y la misma configuración por defecto.

A partir del contenido del mensaje `KEXINIT` del cliente se calcula nuevamente la huella digital *HASSH*, obteniéndose para C4 el mismo valor que en C3: `c11b200866cf918393e62ea25d851d90`. Esto confirma que, a nivel de fingerprint criptográfico, C3 y C4 son indistinguibles y sólo difieren en la interfaz utilizada (red puente Docker versus *loopback*). Finalmente, los paquetes de 114, 574 y 82 bytes corresponden a `KEX ECDH INIT`, `KEX ECDH REPLY + New Keys` y `New Keys` del cliente, respectivamente. A partir de ese momento, Wireshark clasifica el resto del flujo como *Encrypted packet*, indicando que el canal SSH ya se encuentra cifrado y que el contenido de la sesión no es observable en texto plano.

## 1.7. Compara la versión de HASSH obtenida con la base de datos para validar si el cliente corresponde al mismo

Para cada uno de los clientes se extrajo el valor *HASSH* a partir del mensaje `SSH_MSG_KEXINIT` enviado por el cliente. Este identificador se construye a partir de las listas de algoritmos de intercambio de claves, cifrado, integridad y compresión que el cliente propone durante la negociación inicial, por lo que actúa como una huella digital de la implementación SSH utilizada.

Con el *HASSH* calculado para cada captura se realizó una búsqueda en la base de datos pública de huellas *HASSH* (repositorio `salesforce/hassh`) con el fin de validar si el fingerprint obtenido coincide con alguna entrada registrada para clientes OpenSSH.

- **C1 – HASSH:** `68e0ba85e1a818f7c49ea3f4b849bd15`.

El valor coincide con una entrada asociada a clientes basados en OpenSSH 7.2p2, lo que confirma la versión observada en el banner del cliente.

- **C2 – HASSH:** `06046964c022c6407d15a27b12a6a4fb`.

Este fingerprint se corresponde con implementaciones OpenSSH 7.6p1, coherente con la versión indicada en la cadena de identificación del cliente.

- **C3 – HASSH:** `c11b200866cf918393e62ea25d851d90`.

No se encontró coincidencia exacta en la base de datos. Esto sugiere que la combinación de algoritmos ofrecida por OpenSSH 8.2p1 aún no ha sido incorporada en el repositorio consultado.

- **C4 – HASSH:** `c11b200866cf918393e62ea25d851d90`.

Presenta el mismo fingerprint que C3, lo que era esperable dado que C4 utiliza la misma versión de cliente (OpenSSH 8.2p1) y la misma configuración por defecto; la única diferencia entre ambos escenarios es la interfaz de red utilizada.

## **1.8 Tipo de información contenida en cada uno de los paquetes generados en texto plano**

---

En resumen, el uso de *HASSH* permite verificar correctamente las versiones de los clientes más antiguos (C1 y C2), mientras que para versiones más recientes, como *OpenSSH 8.2p1*, la ausencia de coincidencias evidencia la necesidad de mantener actualizada la base de datos de fingerprints para cubrir implementaciones modernas.

### **1.8.1. C1**

En el escenario C1, el primer mensaje relevante en texto plano es el **banner** del cliente, observado en el paquete 4. Allí se indica la versión del protocolo (*SSH-2.0*) y la versión exacta del cliente *OpenSSH* que ejecuta C1. Luego, en los paquetes 8 y 9 se encuentran los mensajes *SSH\_MSG\_KEXINIT* del cliente y del servidor, respectivamente. Estos contienen, de forma legible, las listas de algoritmos propuestos para:

- intercambio de claves (*Key Exchange*),
- cifrado simétrico (*encryption algorithms*),
- integridad mediante MAC (*mac algorithms*),
- compresión (*compression algorithms*).

A partir del mensaje *New Keys* del cliente, el resto de los paquetes se marcan como *Encrypted packet*, por lo que el contenido de la sesión ya no es visible.

### **1.8.2. C2**

Para C2 la estructura es similar. El banner del cliente aparece en el paquete 4 e incluye la versión *SSH-2.0-OpenSSH 7.6p1* junto con la identificación del sistema operativo. En los paquetes 8 y 9 se observan nuevamente los mensajes *SSH\_MSG\_KEXINIT*, donde se listan en texto plano los algoritmos de intercambio de claves, cifrado, MAC y compresión que ofrece esta versión de *OpenSSH*. Estas listas permiten comparar directamente la configuración criptográfica de C2 con la de C1. Después del intercambio de claves ECDH y del mensaje *New Keys*, Wireshark deja de mostrar información legible y todo el tráfico pasa a estar cifrado.

### **1.8.3. C3**

En el caso de C3, el banner del cliente se visualiza también en el paquete 4 e indica que se está utilizando *OpenSSH 8.2p1*. Los paquetes 8 (cliente) y 9 (servidor) contienen las

propuestas de algoritmos en los mensajes `SSH_MSG_KEXINIT`, donde se aprecia que esta versión introduce una lista más extensa de algoritmos, por ejemplo, nuevas opciones de claves de servidor y curvas para el intercambio de claves. Toda esta información se encuentra sin cifrar y puede ser inspeccionada directamente. Una vez que se completa el intercambio de claves y se envía `New Keys`, la información de usuario deja de ser visible y los paquetes posteriores se muestran como cifrados.

#### 1.8.4. C4/S1

En el escenario C4/S1 la conexión se realiza sobre la interfaz `loopback` (127.0.0.1), por lo que tanto el cliente como el servidor se encuentran en la misma máquina. En los paquetes 4 y 6 se observan los banners de cliente y servidor, ambos con la misma versión `SSH-2.0-OpenSSH 8.2p1`. A continuación, los paquetes 8 y 9 contienen los mensajes `SSH_MSG_KEXINIT` de ambas partes, con las listas de algoritmos de intercambio de claves, cifrado, MAC y compresión, idénticas a las vistas en C3. Toda esta información se transmite en texto plano y es visible en el panel de detalle de Wireshark. Después de los mensajes de intercambio de claves ECDH y `New Keys`, el flujo pasa a ser completamente cifrado, de modo que los datos aplicativos ya no pueden interpretarse a partir de la captura.

### 1.9. Diferencia entre C1 y C2

La principal diferencia entre C1 y C2 se encuentra en la versión del cliente SSH y, por consiguiente, en el conjunto de algoritmos criptográficos que cada uno ofrece durante la ne-gociación. C1 utiliza `OpenSSH 7.2p2` sobre Ubuntu 16.04, mientras que C2 emplea `OpenSSH 7.6p1` sobre Ubuntu 18.04.

En el mensaje `SSH_MSG_KEXINIT` de C1 se observan algoritmos considerados más antiguos u obsoletos, por ejemplo algunos grupos Diffie–Hellman basados en `sha1` y cifrados en modo `cbc`. En cambio, C2 elimina parte de estos algoritmos débiles y prioriza opciones más modernas, como `curve25519-sha256` para el intercambio de claves y modos de cifrado en CTR o GCM. Esto se refleja también en un ligero aumento del tamaño del paquete de `KEXINIT` de C2, que incluye una lista más depurada y alineada con las recomendaciones actuales de seguridad.

### 1.10. Diferencia entre C2 y C3

Entre C2 y C3 las diferencias son más sutiles. Ambos clientes utilizan algoritmos moder-nos de intercambio de claves y cifrado, pero C3 ya corresponde a `OpenSSH 8.2p1` (Ubuntu 20.04), mientras que C2 se mantiene en `OpenSSH 7.6p1`. En la captura de C3 se aprecia que el paquete `SSH_MSG_KEXINIT` del cliente es más grande que en C2, lo que indica la inclusión de nuevas opciones, especialmente en la lista de claves de servidor (*server host key algorithms*), como soporte para autenticación basada en hardware (`sk-ecdsa` y variantes afines).

A nivel de algoritmos de cifrado simétrico y MAC, C2 y C3 son muy similares; la diferencia principal radica en la ampliación de compatibilidad y en la incorporación de mecanismos más recientes en la versión 8.2p1, lo que explica que el *HASSH* de C3 no coincida con entradas antiguas de la base de datos.

### 1.11. Diferencia entre C3 y C4

Entre C3 y C4 prácticamente no se observan diferencias en cuanto al protocolo SSH. Ambos escenarios utilizan OpenSSH 8.2p1 y comparten exactamente las mismas listas de algoritmos de intercambio de claves, cifrado, integridad y compresión en sus mensajes `SSH_MSG_KEXINIT`. De hecho, el *HASSH* calculado para C3 y C4 es idéntico, lo que indica que la huella criptográfica del cliente es la misma.

La única diferencia relevante entre ambos casos es de tipo topológico: C3 se comunica con el servidor a través de la red puente de Docker, mientras que C4 establece la sesión sobre la interfaz de *loopback* (127.0.0.1), actuando como cliente y servidor en la misma máquina. Esta diferencia afecta a las direcciones IP observadas en la captura, pero no modifica el comportamiento del *handshake* ni los algoritmos negociados.

## 2. Desarrollo (Parte 2)

### 2.1. Identificación del cliente SSH con versión “?”

En el enunciado se entrega una captura de tráfico SSH en la cual se oculta la versión exacta del cliente, y se solicita determinar con qué cliente SSH se generó dicho flujo. En la Figura 12 se muestra un extracto del *handshake* analizado.

| Time        | Source     | Destination | Protocol | Lengt | Info   |
|-------------|------------|-------------|----------|-------|--|
| 1 0.000000  | 172.18.0.3 | 172.18.0.2  | TCP      | 74    | 47870 + 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TStamp=1231665855 TSectr=0 WS=128                      |
| 2 0.000039  | 172.18.0.3 | 172.18.0.2  | TCP      | 74    | [TCP Retransmission] 47870 + 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TStamp=1231665855 TSectr=0 WS=128 |
| 3 0.000067  | 172.18.0.2 | 172.18.0.3  | TCP      | 74    | 22 + 47870 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TStamp=1072172398 TSectr=0 WS=128           |
| 4 0.000078  | 172.18.0.3 | 172.18.0.2  | TCP      | 66    | 47870 + 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TStamp=1231665855 TSectr=1072172398 WS=128                          |
| 5 0.000289  | 172.18.0.3 | 172.18.0.2  | SSHv2    | 107   | Client: Protocol (SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.7)   |
| 6 0.000303  | 172.18.0.2 | 172.18.0.3  | TCP      | 66    | 22 + 47870 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TStamp=1072172398 TSectr=1231665855 WS=128                         |
| 7 0.007315  | 172.18.0.3 | 172.18.0.3  | SSHv2    | 108   | Server: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.13)  |
| 8 0.007326  | 172.18.0.3 | 172.18.0.2  | TCP      | 66    | 47870 + 22 [ACK] Seq=42 Ack=43 Win=64256 Len=0 TStamp=1231665862 TSectr=1072172405 WS=128                        |
| 9 0.007679  | 172.18.0.3 | 172.18.0.2  | SSHv2    | 1426  | Client: Key Exchange Init  |
| 10 0.008851 | 172.18.0.2 | 172.18.0.3  | SSHv2    | 1146  | Server: Key Exchange Init  |
| 11 0.010695 | 172.18.0.3 | 172.18.0.2  | SSHv2    | 114   | Client: Elliptic Curve Diffie-Hellman Key Exchange Init  |
| 12 0.014511 | 172.18.0.2 | 172.18.0.3  | SSHv2    | 574   | Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=228)                   |
| 13 0.016570 | 172.18.0.3 | 172.18.0.2  | SSHv2    | 82    | Client: New Keys   |
| 14 0.074328 | 172.18.0.2 | 172.18.0.3  | TCP      | 66    | 22 + 47870 [ACK] Seq=1631 Ack=1466 Win=64128 Len=0 TStamp=1072172472 TSectr=1231665872 WS=128                    |
| 15 0.074338 | 172.18.0.3 | 172.18.0.2  | SSHv2    | 110   | Client: Encrypted packet (len=44)  |
| 16 0.074375 | 172.18.0.2 | 172.18.0.3  | TCP      | 66    | 22 + 47870 [ACK] Seq=1631 Ack=1510 Win=64128 Len=0 TStamp=1072172473 TSectr=1231665929 WS=128                    |
| 17 0.074450 | 172.18.0.2 | 172.18.0.3  | SSHv2    | 110   | Server: Encrypted packet (len=44)  |
| 18 0.074519 | 172.18.0.3 | 172.18.0.2  | SSHv2    | 134   | Client: Encrypted packet (len=68)  |
| 19 0.080367 | 172.18.0.2 | 172.18.0.3  | SSHv2    | 118   | Server: Encrypted packet (len=52)  |
| 20 0.126280 | 172.18.0.3 | 172.18.0.2  | TCP      | 66    | 47870 + 22 [ACK] Seq=1578 Ack=1727 Win=64128 Len=0 TStamp=1231665981 TSectr=1072172478 WS=128                    |
| 21 2.315749 | 172.18.0.3 | 172.18.0.2  | SSHv2    | 214   | Client: Encrypted packet (len=148)   |

Figura 12: Tráfico SSH capturado entre C2 (172.18.0.3) y el servidor S1 (172.18.0.2).

Al inspeccionar los primeros paquetes se observa, en el banner del cliente, la cadena: SSH-2.0-OpenSSH\_7.6p1 Ubuntu-4ubuntu0.7. Esta línea aparece en el paquete marcado como *Client: Protocol*, enviado desde la dirección IP 172.18.0.3 hacia 172.18.0.2. Por otro lado, el banner del servidor corresponde a SSH-2.0-OpenSSH\_8.2p1 Ubuntu-4ubuntu0.13, coherente con la configuración utilizada en el contenedor que actúa como servidor.

Además de la coincidencia exacta en el banner, el patrón de tráfico del *handshake* coincide con el observado previamente para el cliente C2:

- Intercambio inicial TCP (SYN, SYN-ACK, ACK).
- Envío de banners de cliente y servidor (paquetes de alrededor de 107–108 bytes).
- Mensajes SSH\_MSG\_KEXINIT de cliente y servidor con tamaños muy similares a los medidos en C2 (del orden de 1420–1150 bytes).
- Secuencia de KEX ECDH INIT, KEX ECDH REPLY + New Keys y finalmente New Keys del cliente, tras lo cual el resto de los paquetes se marcan como *Encrypted packet*.

Tanto la cadena de identificación como la estructura del *handshake* permiten concluir que el tráfico del informante fue generado con el cliente C2, es decir, con la versión OpenSSH 7.6p1 sobre Ubuntu 18.04.

## 2.2. Replicación de tráfico al servidor (paso por paso)

Con el objetivo de validar la hipótesis anterior, se procedió a replicar el tráfico SSH utilizando explícitamente el cliente C2. A continuación se describen los pasos realizados:

### 1. Configuración del servidor S1.

Primero se accedió al contenedor que actúa como servidor y se realizaron las siguientes acciones:

- Creación del directorio de ejecución de sshd:  
`mkdir -p /var/run/sshd`
- Reinicio del servicio SSH:  
`service ssh restart`
- Verificación de la existencia del usuario prueba en /etc/passwd:  
`cat /etc/passwd | grep prueba`

```
root@465837badac8:/# mkdir -p /var/run/sshd
root@465837badac8:/# service ssh restart
  * Restarting OpenBSD Secure Shell server sshd
root@465837badac8:/# cat /etc/passwd | grep prueba
prueba:x:1000:1000::/home/prueba:/bin/bash
root@465837badac8:/# |
```

Figura 13: Configuración del servidor S1 y verificación del usuario prueba.

### 2. Inicio de la captura de tráfico con tcpdump.

Desde el mismo contenedor servidor se creó un directorio para almacenar las capturas y se inició tcpdump filtrando el puerto 22:

```
mkdir -p /capturas
tcpdump -i eth0 port 22 -w /capturas/replicacion_c2.pcap
```

```
root@ca45af78d87e:/# mkdir -p /capturas
root@ca45af78d87e:/# tcpdump -i eth0 port 22 -w /capturas/replicacion_c2.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C94 packets captured
94 packets received by filter
0 packets dropped by kernel
root@ca45af78d87e:/# |
```

Figura 14: Captura de tráfico SSH en el servidor S1 con tcpdump.

**3. Establecimiento de la sesión SSH desde C2.**

En otra terminal se accedió al contenedor C2 (IP 172.18.0.3) y se inició la conexión hacia el servidor S1:

```
ssh prueba@172.18.0.2
```

Al solicitarse la contraseña, se ingresó prueba. Una vez dentro de la sesión SSH se ejecutaron algunos comandos simples (whoami, ls, pwd) para generar tráfico adicional y luego se cerró la sesión con exit.

**4. Finalización de la captura y análisis en Wireshark.**

Tras cerrar la sesión SSH, se detuvo tcpdump con Ctrl+C, obteniendo el archivo replicacion\_c2.pcap. Este archivo se copió al equipo anfitrión y se abrió en Wireshark. Al analizarlo se verificó que:

- El cliente era la IP 172.18.0.3 (C2) y el servidor la IP 172.18.0.2 (S1).
- El banner del cliente coincidía exactamente con el observado en la traza del informante: SSH-2.0-OpenSSH\_7.6p1 Ubuntu-4ubuntu0.7.
- Los tamaños y tipos de paquetes del *handshake* (banners, SSH\_MSG\_KEXINIT, KEX ECDH INIT/REPLY, New Keys y primeros *Encrypted packet*) presentaban el mismo patrón que la captura entregada en el enunciado.

A partir de esta replicación se confirma que el tráfico original puede reproducirse utilizando el cliente C2 con OpenSSH 7.6p1, lo que refuerza la conclusión de que ésta es la versión empleada por el informante en el escenario planteado.

### **3. Desarrollo (Parte 3)**

#### **3.1. Replicación del KEI con tamaño menor a 300 bytes (paso por paso)**

El objetivo de esta parte del laboratorio es modificar la configuración del servidor SSH de manera que el mensaje Key Exchange Init enviado por el *server* (SSH\_MSG\_KEXINIT) tenga un tamaño inferior a 300 bytes. Para lograrlo, se redujeron al mínimo las listas de algoritmos anunciadas por el servidor, manteniendo sólo una opción por categoría.

**1. Modificación de la configuración de sshd.**

En el contenedor que actúa como servidor (S1) se editó el archivo /etc/ssh/sshd\_config utilizando nano, agregando o ajustando las siguientes directivas:

### 3.1 Replicación del KEI con tamaño menor a 300 bytes (paso DESARROLLO (PARTE 3))

```
KexAlgorithms curve25519-sha256
Ciphers chacha20-poly1305@openssh.com
MACs hmac-sha2-256
HostKey /etc/ssh/ssh_host_ed25519_key
Compression no
```

Con esto se fuerza al servidor a anunciar un único algoritmo de intercambio de claves (`curve25519-sha256`), un único cifrado simétrico (`chacha20-poly1305@openssh.com`), un único MAC (`hmac-sha2-256`), una sola clave de servidor (`ssh_host_ed25519_key`) y se deshabilita la compresión. Al reducir la cantidad de opciones, el tamaño del paquete KEXINIT disminuye de manera significativa.

```
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none

# no default banner path
#Banner none

# Allow client to pass locale environment variables
AcceptEnv LANG LC_*

# override default of no subsystems
Subsystem      sftp      /usr/lib/openssh/sftp-server

# Example of overriding settings on a per-user basis
#Match User anoncvs
#    X11Forwarding no
#    AllowTcpForwarding no
#    PermitTTY no
#    ForceCommand cvs server
KexAlgorithms curve25519-sha256
Ciphers chacha20-poly1305@openssh.com
MACs hmac-sha2-256
HostKey /etc/ssh/ssh_host_ed25519_key
Compression no
```

Figura 15: Edición del archivo `/etc/ssh/sshd_config` para limitar los algoritmos anunciados por el servidor SSH.

#### 2. Reinicio del servicio SSH.

Una vez modificada la configuración, se reinició el servicio para aplicar los cambios:

```
service ssh restart
```

### 3.1 Replicación del KEI con tamaño menor a 300 bytes (paso DESARROLLO (PARTE 3))

Opcionalmente, en el cliente C2 se eliminó la entrada previa del servidor en `known_hosts` para evitar advertencias por cambio de clave de host.

#### 3. Captura de un nuevo *handshake* SSH.

Con el servidor ya configurado, se inició una nueva sesión SSH desde el cliente C2 hacia S1 y, en paralelo, se capturó el tráfico (por ejemplo, con `tcpdump` en el servidor o directamente con Wireshark en la interfaz de Docker). La sesión se limitó al *login* y ejecución de algunos comandos simples para generar el *handshake* completo.

#### 4. Verificación del tamaño del Key Exchange Init.

Finalmente, se abrió la captura en Wireshark y se filtró por protocolo SSH. Al seleccionar el paquete etiquetado como *Server: Key Exchange Init* se verificó en la columna *Length* que su tamaño efectivo era de 290 bytes, cumpliendo así la condición de ser menor a 300 bytes.

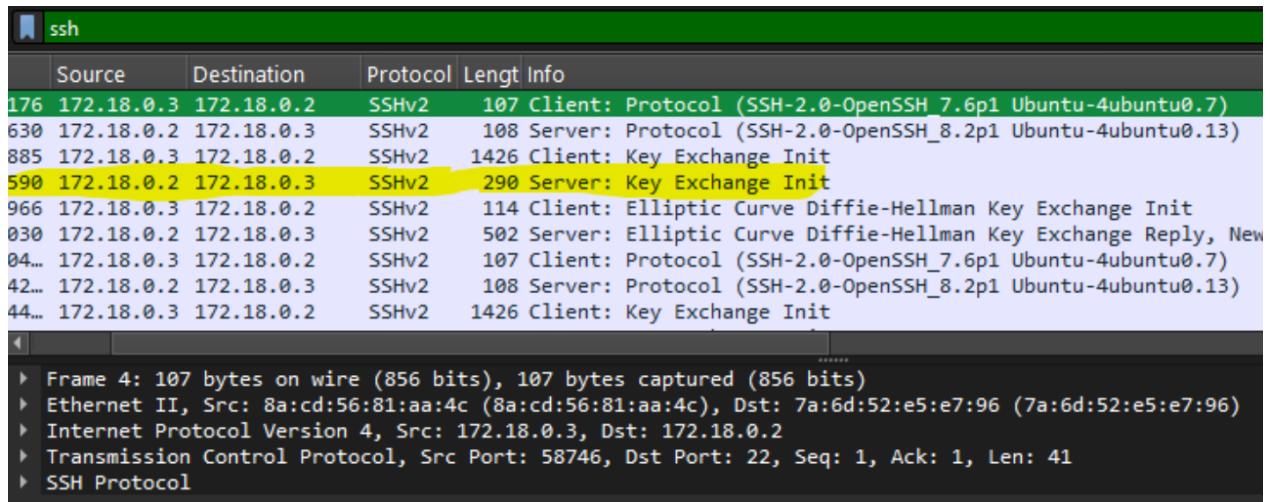


Figura 16: Captura en Wireshark donde se observa el mensaje *Server: Key Exchange Init* con longitud de 290 bytes.

Gracias a estas modificaciones se demostró que el tamaño del mensaje `SSH_MSG_KEXINIT` del servidor depende directamente de la cantidad de algoritmos incluidos en las listas de negociación. Al restringir dichas listas a una sola opción por categoría y desactivar la compresión, es posible controlar el patrón de tráfico del *handshake* SSH y obtener un *Key Exchange Init* con longitud inferior a 300 bytes.

## 4. Desarrollo (Parte 4)

### 4.1. Explicación OpenSSH en general

OpenSSH (*Open Secure Shell*) es una implementación libre del protocolo SSH que permite establecer conexiones remotas seguras entre un cliente y un servidor. Su objetivo principal es reemplazar protocolos inseguros como `telnet`, `rsh` o `ftp`, proporcionando confidencialidad, integridad y autenticación sobre redes no confiables.

OpenSSH funciona bajo un modelo *cliente–servidor*. El cliente inicia la conexión hacia el servidor, el cual escucha normalmente en el puerto TCP 22. El flujo básico de una sesión SSH puede resumirse en las siguientes etapas:

1. **Intercambio de banners:** ambas partes envían en texto plano su cadena de identificación (`SSH-2.0-OpenSSH...`), indicando la versión del protocolo y de la implementación. Esta información se observa en los primeros paquetes del flujo.
2. **Negociación de algoritmos:** cliente y servidor intercambian los mensajes `SSH_MSG_KEXINIT`, donde proponen sus listas de algoritmos soportados para intercambio de claves, cifrado, integridad y compresión. A partir de estas listas se elige un conjunto común de algoritmos.
3. **Intercambio de claves:** utilizando el algoritmo acordado (por ejemplo `curve25519-sha256` o algún grupo Diffie–Hellman), cliente y servidor calculan de forma colaborativa una clave secreta compartida sin transmitirla explícitamente por la red.
4. **Autenticación:** una vez creado el canal cifrado, el servidor se autentica ante el cliente mediante su clave de host, y luego el cliente se autentica frente al servidor mediante contraseña o par de claves pública/privada.
5. **Canal de datos cifrado:** a partir del mensaje `New Keys` todo el tráfico de datos de la sesión (comandos, salida de la terminal, *port forwarding*, etc.) viaja cifrado y protegido mediante códigos de integridad.

En el laboratorio, este flujo completo se pudo observar en los diferentes escenarios C1–C4, tanto en las capturas de Wireshark como en la configuración de los contenedores Docker.

### 4.2. Capas de seguridad en OpenSSH

El protocolo SSH, y en particular OpenSSH, implementa varias capas de seguridad que se combinan para proteger la comunicación. A partir de las trazas analizadas se pueden identificar claramente las siguientes:

1. **Capa de negociación de algoritmos.**

Corresponde a los mensajes `SSH_MSG_KEXINIT`. En ellos se anuncian listas de:

- algoritmos de intercambio de claves (`KexAlgorithms`),

- cifrado simétrico (**Ciphers**),
- integridad (**MACs**),
- algoritmos de compresión (**Compression**).

Esta capa permite seleccionar algoritmos comunes y suficientemente robustos, contribuyendo a la **confidencialidad**, **integridad** y **autenticidad** del canal.

## 2. Capa de intercambio de claves.

Una vez elegidos los algoritmos, se ejecuta el protocolo de intercambio de claves (ECDH o DH clásico). Los paquetes etiquetados como **KEX ECDH INIT** y **KEX ECDH REPLY** en Wireshark corresponden a esta etapa. Su función es derivar una clave secreta compartida sin exponerla directamente en la red, lo cual protege frente a atacantes pasivos y contribuye a la **confidencialidad**.

## 3. Capa de autenticación.

Con el canal cifrado establecido, el servidor se identifica mediante su clave de host y el cliente se autentica usando contraseña o clave pública. La verificación de la clave de host en el archivo **known\_hosts** evita ataques de *man-in-the-middle*, reforzando la **autenticidad** de los extremos e influyendo parcialmente en el **no repudio** cuando se utilizan claves públicas personales.

## 4. Capa de cifrado de datos.

Todo el tráfico posterior se cifra con el algoritmo acordado (por ejemplo **chacha20-poly1305@openssh.com** o **aes-gcm**). En las capturas, esta etapa se observa cuando los paquetes pasan a rotularse como *Encrypted packet*. Esta capa garantiza principalmente la **confidencialidad** de la información intercambiada.

## 5. Capa de integridad.

Cada paquete cifrado incluye un código de autenticación de mensaje (MAC), calculado con algoritmos como **hmac-sha2-256**. Esto permite detectar modificaciones no autorizadas en tránsito, aportando a la **integridad** de los datos.

## 6. Capa de compresión (opcional).

SSH puede comprimir el flujo de datos antes de cifrarlo. En el laboratorio se desactivó esta funcionalidad en algunos escenarios (**Compression no**) para reducir el tamaño del mensaje **KEXINIT**. Aunque la compresión no es una capa de seguridad en sí misma, puede afectar el patrón de tráfico y el rendimiento.

## 7. Capa de registro y control.

OpenSSH mantiene registros de accesos y cambios de clave de host, y el usuario puede inspeccionar estos eventos en los archivos de **logs**. Esto apoya la **disponibilidad** del servicio y facilita tareas de auditoría y detección de incidentes.

En conjunto, estas capas permiten que una sesión SSH proporcione confidencialidad e integridad de extremo a extremo, autenticación mutua y cierta capacidad de trazabilidad.

### 4.3. Identificación de principios que no se cumplen completamente

Si bien OpenSSH implementa varios mecanismos de seguridad, a partir del análisis de las capturas y de la configuración utilizada se identifican algunos puntos donde los principios clásicos de seguridad no se cumplen de manera absoluta:

- **No repudio limitado.**

El protocolo no incorpora, por defecto, un mecanismo fuerte de no repudio. Cuando se usa autenticación por contraseña, es imposible demostrar posteriormente que una acción específica fue realizada por un usuario en particular. Incluso utilizando claves públicas, la evidencia depende de los `logs` locales y de la correcta protección de las claves privadas, por lo que el no repudio no está totalmente garantizado.

- **Confidencialidad parcial en la fase inicial.**

Los mensajes de intercambio de banners y KEXINIT se envían en texto plano. Aunque no contienen datos confidenciales del usuario, sí exponen información sobre versiones de software y listas de algoritmos soportados. Estos metadatos permiten realizar *fingerprinting* del cliente/servidor e incluso seleccionar vulnerabilidades específicas si alguna versión está desactualizada.

- **Dependencia de la configuración.**

La seguridad efectiva de la sesión depende fuertemente de la configuración elegida. En C1, por ejemplo, se observaron algoritmos más antiguos y potencialmente débiles, mientras que en versiones más recientes dicho conjunto se encuentra depurado. Un administrador que no actualice o ajuste la configuración podría dejar activos algoritmos obsoletos, afectando la **confidencialidad** y la **integridad**.

Aun así, en los escenarios configurados en el laboratorio se utilizaron versiones y algoritmos considerados seguros en la actualidad, por lo que los principios de confidencialidad, integridad, autenticidad y disponibilidad se consideran razonablemente satisfechos.

### 4.4. Conclusiones y comentarios

La realización de este laboratorio permitió pasar de una visión teórica del protocolo SSH a un entendimiento práctico. Mediante la creación de contenedores Docker con distintas versiones de Ubuntu y OpenSSH se observó cómo varía el *handshake* entre C1, C2, C3 y C4, tanto en el tamaño de los paquetes como en los algoritmos negociados.

El uso de huellas *HASSH* fue útil para identificar y comparar clientes, mostrando que es posible asociar un patrón de tráfico a versiones concretas de OpenSSH. A su vez, la modificación del archivo `sshd_config` para obtener un *Key Exchange Init* menor a 300 bytes evidenció la relación directa entre la configuración del servidor y el tráfico observable en la red.

Finalmente, el trabajo reforzó el uso de herramientas como Docker, `tcpdump` y Wireshark en el análisis de seguridad, destacando que comprender los detalles del protocolo es clave para evaluar riesgos y ajustar la configuración de forma segura.