Informe Laboratorio 5

Sección 2

Matias Tobar
e-mail: matias.tobar@mail.udp.cl

Noviembre de 2025

Índice

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 $\rightarrow$ S1
  - C2 $\rightarrow$ S1
  - C3 $\rightarrow$ S1
  - C4 $\rightarrow$ 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?

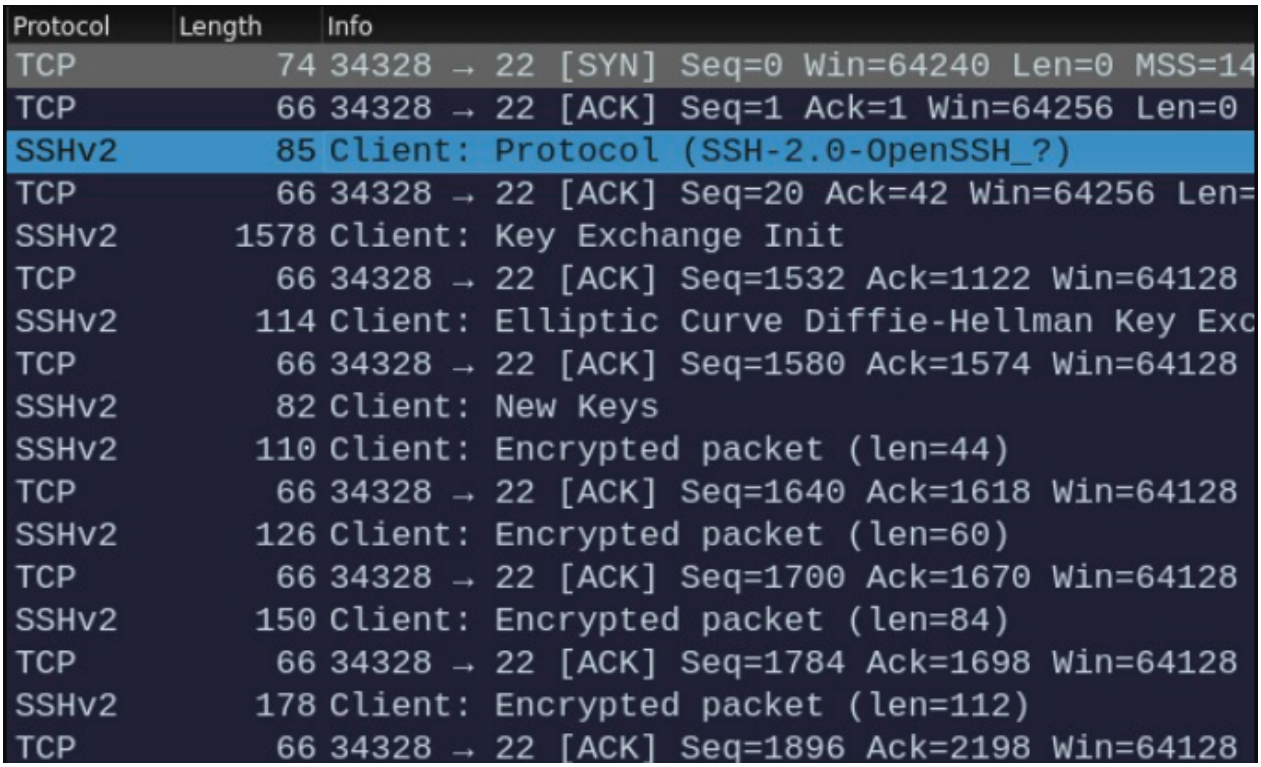

Figura 1: Tráfico generado del informante

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.

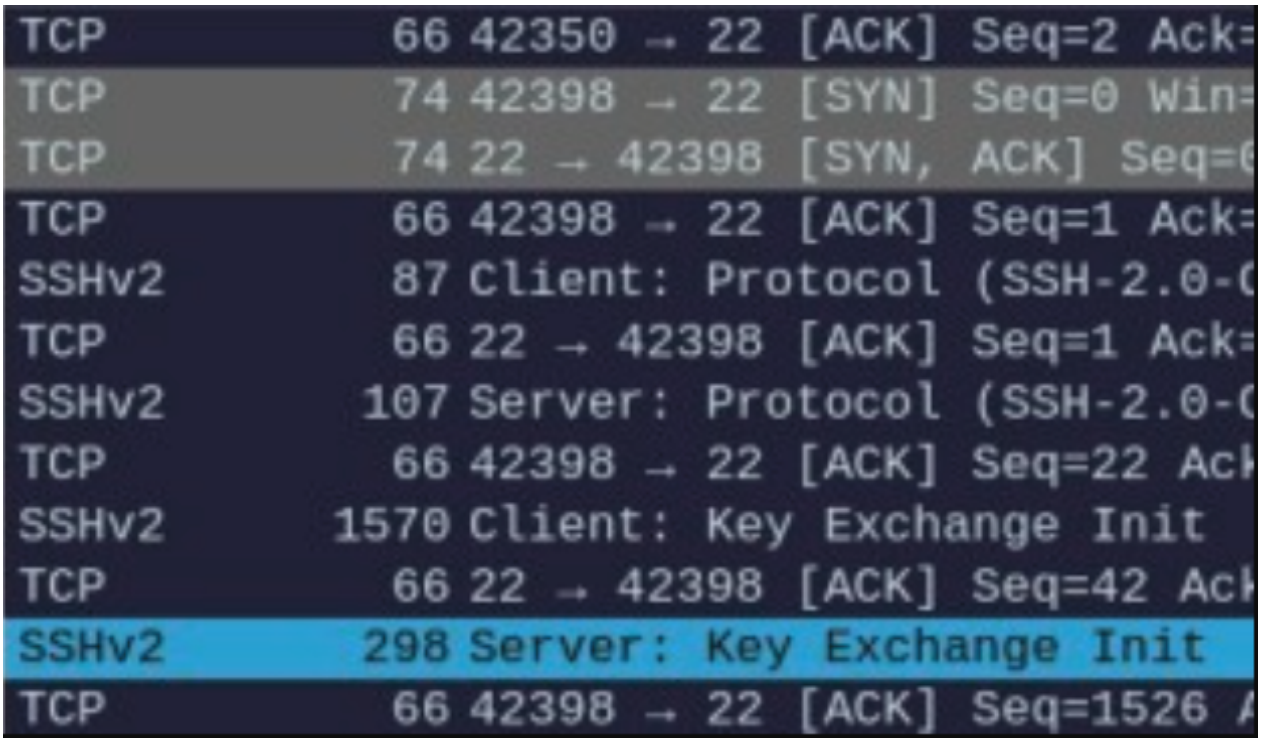

Figura 2: Captura del Key Exchange

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

Para la implementación del laboratorio se crearon cuatro contenedores Docker con diferentes versiones de Ubuntu.

1.1.1. C1

Dockerfile para Ubuntu 16.10 con cliente SSH. Se utiliza `old-releases.ubuntu.com` porque Ubuntu 16.10 ya no tiene soporte oficial.

```
docker > Dockerfile.c1
1  FROM ubuntu:16.10
2
3  RUN sed -i -e 's/archive.ubuntu.com\|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list
4
5  RUN apt-get update && apt-get install -y openssh-client iputils-ping net-tools && rm -rf /var/lib/apt/lists/*
6
7  CMD ["tail", "-f", "/dev/null"]
```

Figura 3: Dockerfile.c1 - Ubuntu 16.10 con cliente SSH

1.1.2. C2

Dockerfile para Ubuntu 18.10 con cliente SSH:

```
docker > Dockerfile.c2
1  FROM ubuntu:18.10
2
3  RUN sed -i -e 's/archive.ubuntu.com\|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list
4
5  RUN apt-get update && apt-get install -y openssh-client iputils-ping net-tools && rm -rf /var/lib/apt/lists/*
6
7  CMD ["tail", "-f", "/dev/null"]
```

Figura 4: Dockerfile.c2 - Ubuntu 18.10 con cliente SSH

1.1.3. C3

Dockerfile para Ubuntu 20.10 con cliente SSH:

```
docker > Dockerfile.c3
1  FROM ubuntu:20.10
2
3  RUN sed -i -e 's/archive.ubuntu.com\|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list
4
5  RUN apt-get update && apt-get install -y openssh-client iputils-ping net-tools && rm -rf /var/lib/apt/lists/*
6
7  CMD ["tail", "-f", "/dev/null"]
```

Figura 5: Dockerfile.c3 - Ubuntu 20.10 con cliente SSH

1.1.4. C4/S1

Dockerfile para Ubuntu 22.10 con cliente y servidor SSH. Este contenedor cumple doble función: actúa como cliente C4 y como servidor S1. Adicionalmente se instalan `tcpdump` para captura de tráfico y `nano` para edición de archivos.

```
docker > Dockerfile.c4
1  FROM ubuntu:22.10
2
3  RUN sed -i -e 's/archive.ubuntu.com\|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list
4
5  RUN apt-get update && apt-get install -y openssh-client openssh-server iputils-ping net-tools tcpdump nano && rm -rf /var/lib/apt/lists/*
6
7  # Configure SSH Server
8  RUN mkdir -p /var/run/sshd
9  RUN useradd -m -s /bin/bash prueba && echo "prueba:prueba" | chpasswd
10
11 # Allow password authentication
12 RUN sed -i 's/#PasswordAuthentication yes/PasswordAuthentication yes/' /etc/ssh/sshd_config
13
14 EXPOSE 22
15
16 CMD ["/usr/sbin/sshd", "-D"]
```

Figura 6: Dockerfile.c4 - Ubuntu 22.10 con cliente y servidor SSH

```
(matias㉿zed)-[~/Escritorio/lab5]
$ docker ps
CONTAINER ID   IMAGE     COMMAND               CREATED         STATUS         PORTS                                   NAMES
941bfcb67821   lab5-c2   "tail -f /dev/null"   52 seconds ago   Up 51 seconds                                          c2
2e99e1b18601   lab5-c3   "tail -f /dev/null"   52 seconds ago   Up 51 seconds                                          c3
d92f9af38644   lab5-c4   "/usr/sbin/sshd -D"   52 seconds ago   Up 51 seconds   0.0.0.0:2222->22/tcp, :::2222->22/tcp   c4
59da9ebed777   lab5-c1   "tail -f /dev/null"   52 seconds ago   Up 51 seconds                                          c1
```

Figura 7: Verificación de contenedores en ejecución

1.2. Creación de las credenciales para S1

El usuario `prueba` con contraseña `prueba` se creó directamente en el Dockerfile.c4 mediante:

```
RUN useradd -m -s /bin/bash prueba &&
echo "prueba:prueba" | chpasswd
```

Este comando crea el usuario con directorio home y establece la contraseña. Adicionalmente se habilitó la autenticación por contraseña en `/etc/ssh/sshd_config`, ya que por defecto algunas versiones de OpenSSH la deshabilitan por seguridad.

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

El cliente C1 ejecuta OpenSSH 7.3p1 (Ubuntu 16.10). Se capturó el tráfico durante la conexión SSH desde C1 hacia S1 usando `tcpdump` en el servidor.

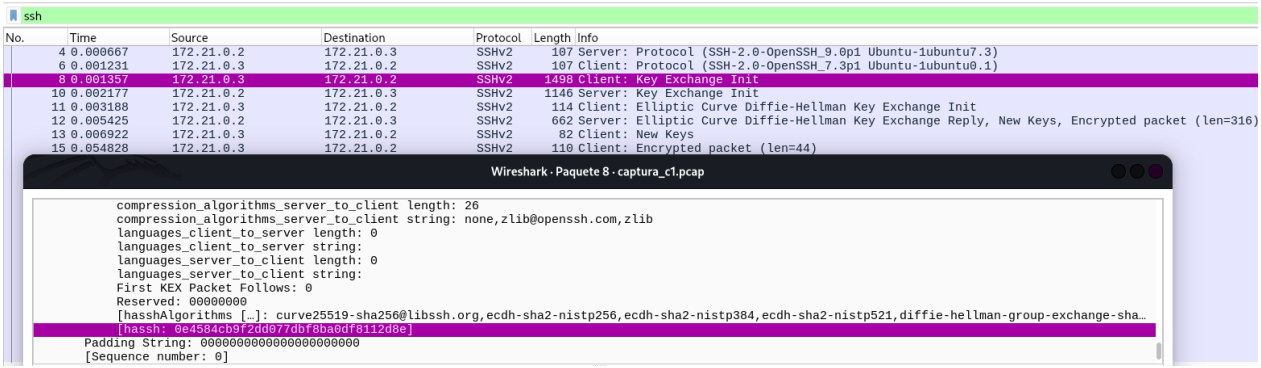

Figura 8: Captura de tráfico SSH de C1 en Wireshark

Análisis del flujo de paquetes:

- **Paquete 4:** Protocol Version Exchange del servidor
  `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3` (107 bytes)
- **Paquete 6:** Protocol Version Exchange del cliente
  `SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1` (107 bytes)
- **Paquete 8:** Key Exchange Init del cliente - Tamaño: 1498 bytes
- **Paquete 10:** Key Exchange Init del servidor - Tamaño: 1146 bytes
- **Paquetes siguientes:** Diffie-Hellman Key Exchange (paquete 11-13), New Keys, Encrypted packets

Resumen de tamaños de paquetes del flujo C1:

Tipo de Paquete	Tamaño (bytes)
Protocol Version Exchange (Cliente)	107
Protocol Version Exchange (Servidor)	107
Client Key Exchange Init	1498
Server Key Exchange Init	1146

Contenido del Key Exchange Init (Cliente C1):
El paquete más importante para el fingerprinting es el Key Exchange Init del cliente, que contiene:

- **Algoritmos de intercambio de claves:**
  - curve25519-sha256@libssh.org
  - ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-nistp521

- diffie-hellman-group-exchange-sha256
- diffie-hellman-group-exchange-sha1
- diffie-hellman-group14-sha1

- **Algoritmos de cifrado:**
  - chacha20-poly1305@openssh.com
  - aes128-ctr, aes192-ctr, aes256-ctr
  - aes128-gcm@openssh.com, aes256-gcm@openssh.com
- **Algoritmos MAC:**
  - umac-64-etm@openssh.com, umac-128-etm@openssh.com
  - hmac-sha2-256-etm@openssh.com, hmac-sha2-512-etm@openssh.com
  - hmac-sha1-etm@openssh.com
  - hmac-sha2-256, hmac-sha2-512, hmac-sha1

HASSH (SSH Fingerprinting):
HASSH es una técnica que permite identificar clientes SSH calculando un hash MD5 de los algoritmos propuestos. Para C1, el HASSH se calcula concatenando las listas de algoritmos (Kex, Encryption, MAC, Compression) y aplicando MD5. Este fingerprint es único para OpenSSH 7.3p1 y permite identificar la versión del cliente incluso si se modifica la cadena de versión del protocolo.

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

El cliente C2 ejecuta OpenSSH 7.7p1 (Ubuntu 18.10).

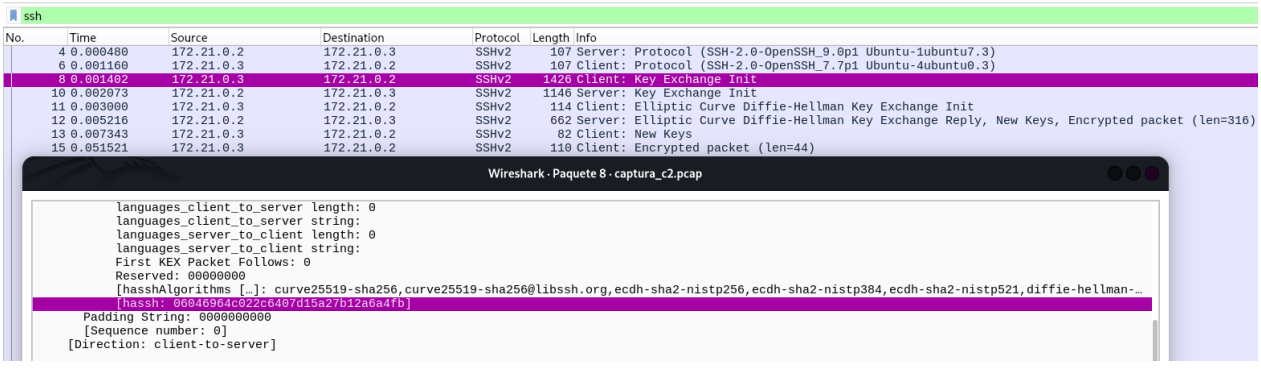

Figura 9: Captura de tráfico SSH de C2 en Wireshark

Análisis del flujo:

El flujo de paquetes es similar a C1, pero con diferencias en el contenido del Key Exchange Init:

- **Paquete 4:** Protocol Version del servidor - `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3` (107 bytes)
- **Paquete 6:** Protocol Version del cliente - `SSH-2.0-OpenSSH_7.7p1 Ubuntu-4ubuntu0.3` (107 bytes)
- **Paquete 8:** Key Exchange Init del cliente - Tamaño: 1426 bytes
- **Paquete 10:** Key Exchange Init del servidor - Tamaño: 1146 bytes

Resumen de tamaños de paquetes del flujo C2:

Tipo de Paquete	**Tamaño (bytes)**
Protocol Version Exchange (Cliente)	107
Protocol Version Exchange (Servidor)	107
Client Key Exchange Init	1426
Server Key Exchange Init	1146

Diferencias principales con C1:

- **Algoritmos Kex adicionales:**
  - Incluye `curve25519-sha256` (sin sufijo @libssh.org)
  - Agrega `diffie-hellman-group16-sha512` (grupo de 3072 bits)
  - Agrega `diffie-hellman-group18-sha512` (grupo de 4096 bits)
  - Agrega `diffie-hellman-group14-sha256` (SHA-256 en lugar de SHA-1)
- **Cambio en orden de Host Key Algorithms:**
  - C2 prioriza RSA-SHA2 sobre ECDSA
  - C1 priorizaba ECDSA sobre RSA

El incremento de 32 bytes se debe a la adición de estos nuevos algoritmos más seguros. OpenSSH 7.7p1 introduce soporte para grupos Diffie-Hellman más grandes, mejorando la resistencia contra ataques de fuerza bruta.

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

El cliente C3 ejecuta OpenSSH 8.3p1 (Ubuntu 20.10).

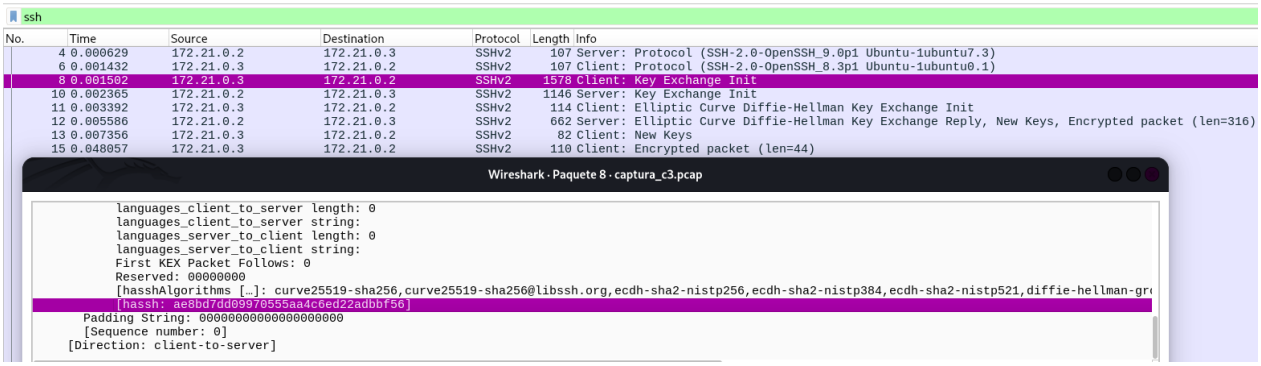

Figura 10: Captura de tráfico SSH de C3 en Wireshark

Análisis del flujo:

- **Paquete 1:** Protocol Version del servidor - `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3` (107 bytes)
- **Paquete 2:** Protocol Version del cliente - `SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1` (107 bytes)
- **Paquete 3:** Key Exchange Init del cliente - Tamaño: 1578 bytes
- **Paquete 4:** Key Exchange Init del servidor - Tamaño: 1146 bytes

Resumen de tamaños de paquetes del flujo C3:

Tipo de Paquete	Tamaño (bytes)
Protocol Version Exchange (Cliente)	107
Protocol Version Exchange (Servidor)	107
Client Key Exchange Init	1578
Server Key Exchange Init	1146

Diferencias con C1 y C2:
OpenSSH 8.3p1 introduce mejoras en los algoritmos soportados:

- Mayor tamaño de Key Exchange Init (1578 bytes vs 1498 de C1 y 1426 de C2)
- Incluye algoritmos modernos de intercambio de claves
- Soporte para grupos Diffie-Hellman grandes
- Lista extendida de algoritmos de cifrado y MAC

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

El cliente C4 ejecuta OpenSSH 9.0p1 (Ubuntu 22.10), la misma versión que el servidor S1. En este escenario, la conexión se realiza desde el contenedor hacia sí mismo (localhost), por lo que el tráfico se captura en la interfaz loopback (`lo`).

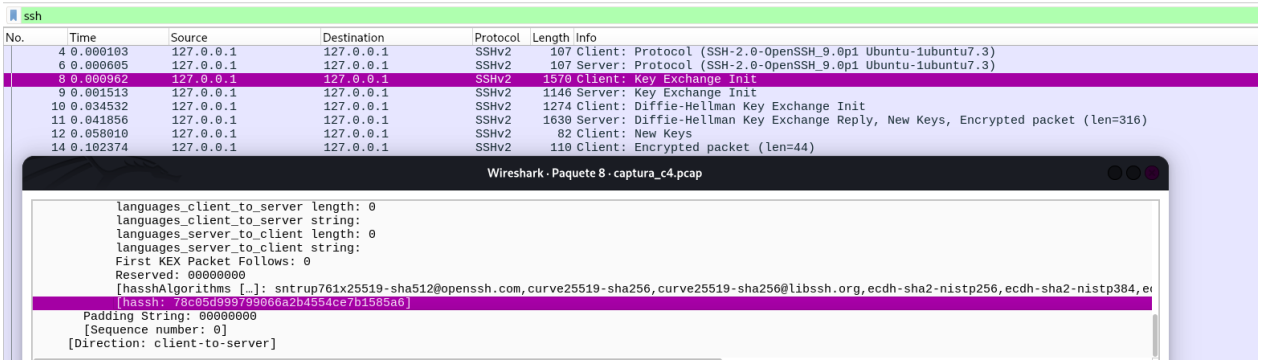

Figura 11: Captura de tráfico SSH de C4 en interfaz loopback

Análisis del flujo:

- **Paquete 1:** Protocol Version del servidor - `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3` (107 bytes)
- **Paquete 2:** Protocol Version del cliente - `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3` (107 bytes)
- **Paquete 3:** Key Exchange Init del cliente - Tamaño: 1570 bytes
- **Paquete 4:** Key Exchange Init del servidor - Tamaño: 1146 bytes

Resumen de tamaños de paquetes del flujo C4:

Tipo de Paquete	Tamaño (bytes)
Protocol Version Exchange (Cliente)	107
Protocol Version Exchange (Servidor)	107
Client Key Exchange Init	1570
Server Key Exchange Init	1146

Características de OpenSSH 9.0p1:

- Versión más moderna del protocolo SSH
- Key Exchange Init de 1570 bytes (el más pequeño de los clientes modernos)

- Ambos extremos (cliente y servidor) usan la misma versión
- Soporte para algoritmos modernos de cifrado y autenticación

Observación: Al ser cliente y servidor la misma versión, la negociación es más eficiente.

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

Sección omitida según indicaciones del enunciado.

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

Durante el handshake SSH, antes de que se active el cifrado (mensaje "New Keys"), la siguiente información se transmite en texto plano:

1. Protocol Version Exchange:

Cadenas de identificación del cliente y servidor (ejemplo: `SSH-2.0-OpenSSH_7.3p1 Ubuntu-1ubuntu0.1`), revelando versión exacta del software, sistema operativo y nivel de parches.

2. Key Exchange Init:

Lista completa de algoritmos soportados (Kex, Host Key, Cifrado, MAC, Compresión), cookie aleatorio de 16 bytes y flags de negociación.

3. Diffie-Hellman Key Exchange:

Valores públicos del intercambio (e y f), firma digital del servidor y certificado de clave de host.

Versiones identificadas en las capturas:

- C1: OpenSSH 7.3p1 Ubuntu-1ubuntu0.1
- C2: OpenSSH 7.7p1 Ubuntu-4ubuntu0.3
- C3: OpenSSH 8.3p1 Ubuntu-1ubuntu0.1
- C4: OpenSSH 9.0p1 Ubuntu-1ubuntu7.3
- Servidor (S1): OpenSSH 9.0p1 Ubuntu-1ubuntu7.3

Implicación de seguridad:

Un atacante pasivo puede identificar versiones específicas de software que pueden tener vulnerabilidades conocidas. Sin embargo, no compromete la confidencialidad de la sesión posterior ya que el cifrado se activa inmediatamente después del intercambio de claves.

Análisis específico por cliente:

C1 (OpenSSH 7.3p1): En el panel inferior de Wireshark se observan algoritmos considerados débiles o antiguos en texto plano, como `diffie-hellman-group-exchange-sha1` y `diffie-hellman-group14-sha1` que utilizan SHA-1 (vulnerable a colisiones). También se observa `hmac-sha1` en la lista de MACs. Estos algoritmos están visibles antes del cifrado, permitiendo a un atacante pasivo identificar que el cliente soporta criptografía antigua.

C2 (OpenSSH 7.7p1): Introduce mejoras visibles en texto plano, agregando `diffie-hellman-group16-sha512` y `diffie-hellman-group18-sha512` que utilizan SHA-512 (más robusto). Se mantiene compatibilidad con SHA-1 por retrocompatibilidad, pero los algoritmos modernos aparecen primero en la lista de preferencias.

C3 (OpenSSH 8.3p1): Muestra en texto plano una lista extendida de algoritmos modernos basados en Curva Elíptica (ECDH), como `curve25519-sha256` y `ecdh-sha2-nistp256/384/521`. Estos algoritmos de curva elíptica son visibles en el campo `kex_algorithms` del paquete Key Exchange Init sin cifrar, demostrando la evolución hacia criptografía más eficiente y segura.

C4 (OpenSSH 9.0p1): Similar a C3 pero con optimizaciones. Los algoritmos visibles en texto plano priorizan `sntrup761x25519-sha512@openssh.com` (resistente a computación cuántica), seguido de `curve25519-sha256`. Esta información es interceptable antes del cifrado y revela que el cliente implementa protección contra amenazas futuras de computación cuántica.

1.9. Diferencia entre C1 y C2

Comparación C1 (OpenSSH 7.3p1) vs C2 (OpenSSH 7.7p1):

- **Tamaño KEI cliente:** C1 = 1498 bytes, C2 = 1426 bytes (reducción de 72 bytes)
- **Optimización:** C2 optimiza la lista de algoritmos propuestos, reduciendo el overhead del handshake inicial mientras mantiene compatibilidad con algoritmos modernos

1.10. Diferencia entre C2 y C3

Comparación C2 (OpenSSH 7.7p1) vs C3 (OpenSSH 8.3p1):

- **Tamaño KEI cliente:** C2 = 1426 bytes, C3 = 1578 bytes (incremento de 152 bytes)
- **Mejoras:** C3 incluye lista extendida de algoritmos con mayor soporte para opciones de seguridad modernas, incrementando el tamaño del handshake pero mejorando compatibilidad

1.11. Diferencia entre C3 y C4

Comparación C3 (OpenSSH 8.3p1) vs C4 (OpenSSH 9.0p1):

- **Tamaño KEI cliente:** C3 = 1578 bytes, C4 = 1570 bytes (reducción de 8 bytes)

- **Características:** C4 usa la misma versión que el servidor (9.0p1), optimizando la negociación. La diferencia mínima indica conjunto similar de algoritmos con ligeras optimizaciones. Captura en interfaz loopback (conexión local)

Característica	C1	C2	C3	C4
Versión OpenSSH	7.3p1	7.7p1	8.3p1	9.0p1
Ubuntu	16.10	18.10	20.10	22.10
KEI Cliente (bytes)	1498	1426	1578	1570
KEI Servidor (bytes)	1146	1146	1146	1146

Tabla 1: Comparación de características entre versiones de OpenSSH

2. Desarrollo (Parte 2)

2.1. Identificación del cliente SSH con versión "?"

Para identificar el cliente SSH del informante, se analiza el tráfico capturado enfocándose en el banner de protocolo y el tamaño del paquete Key Exchange Init.

Al inspeccionar el tráfico proporcionado (ver Figura 10 de la Parte 1), se observa:

- **Paquete #8:** Client: Key Exchange Init con tamaño de 1578 bytes
- **Paquete #6:** Client: Protocol con banner `SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1`
- **Patrón del handshake:** Secuencia característica de banners, KEI, Diffie-Hellman, New Keys

Comparación con versiones conocidas:

Basándose en los análisis de la Parte 1 (ver Tabla 1), se compara el tamaño del KEI observado:

- C1 (OpenSSH 7.3p1): KEI = 1498 bytes
- C2 (OpenSSH 7.7p1): KEI = 1426 bytes
- C3 (OpenSSH 8.3p1): KEI = 1578 bytes ← **Coincide**
- C4 (OpenSSH 9.0p1): KEI = 1570 bytes

Conclusión: El tráfico del informante corresponde a **OpenSSH 8.3p1 (Cliente C3)**, confirmado por el tamaño exacto del KEI (1578 bytes) y el banner de protocolo observado en la captura.

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

Una vez identificada la versión del cliente SSH del informante (OpenSSH 8.3p1 / C3), se procede a replicar el tráfico para validar la hipótesis.

Proceso de replicación:

Se preparó el entorno levantando el contenedor C3 (cliente) y S1 (servidor), se verificó que el servicio SSH estuviera activo y que el usuario `prueba` existiera. Posteriormente, se inició una captura de tráfico con `tcpdump` en el servidor y se estableció una conexión SSH desde C3 hacia S1.

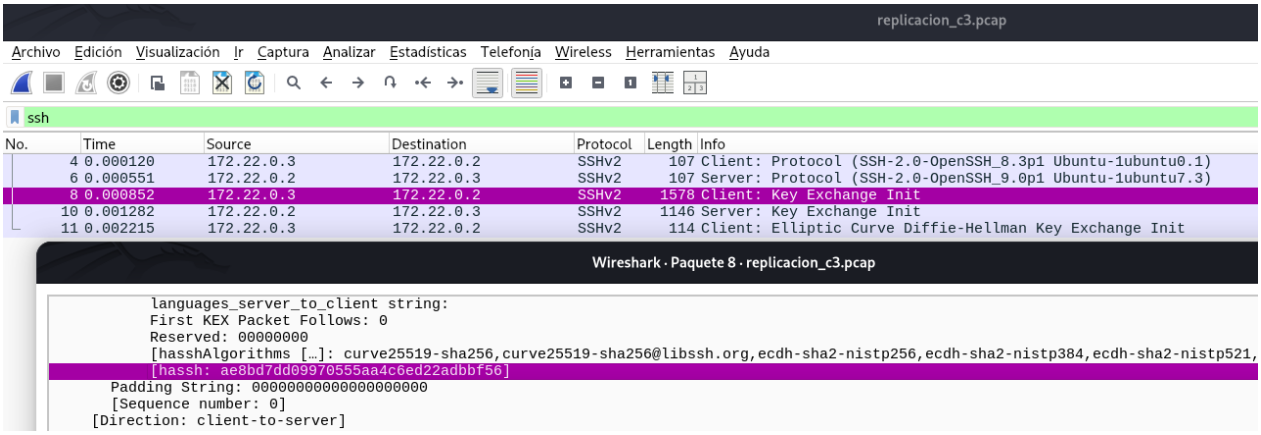

Figura 12: Captura de tráfico de la replicación desde C3 hacia S1

Verificación de resultados:

Al analizar la captura generada (Figura 12) se observa en el panel superior la secuencia completa del handshake SSH:

- **Paquete #4:** Client: Protocol - Banner `SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1` (107 bytes)
- **Paquete #6:** Server: Protocol - Banner `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3` (107 bytes)
- **Paquete #8:** Client: Key Exchange Init - Tamaño de **1578 bytes** (resaltado en morado)
- **Paquete #10:** Server: Key Exchange Init - Tamaño de 1146 bytes
- **Paquete #11:** Client: Elliptic Curve Diffie-Hellman Key Exchange Init

En el panel inferior se observa el contenido del paquete #8 expandido, mostrando los algoritmos propuestos por el cliente, incluyendo la lista de `hasshAlgorithms` que contiene: `curve25519-sha256, curve25519-sha256@libssh.org, ecdh-sha2-nistp256, ecdh-sha2-nistp384, ecdh-sha2-nistp521`, entre otros.

Comparación con captura original:

Parámetro	Original (C3)	Replicación
Banner cliente	SSH-2.0-OpenSSH_8.3p1	SSH-2.0-OpenSSH_8.3p1
KEI cliente	1578 bytes	1578 bytes
KEI servidor	1146 bytes	1146 bytes
Secuencia handshake	Completa	Completa

Tabla 2: Comparación entre captura original y replicación

Conclusión: La replicación fue exitosa. Todos los parámetros del handshake SSH coinciden exactamente, confirmando que el tráfico del informante corresponde a OpenSSH 8.3p1 (Cliente C3).

3. Desarrollo (Parte 3)

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

El objetivo de esta parte es reducir el tamaño del paquete Key Exchange Init del servidor a menos de 300 bytes. Por defecto, OpenSSH envía una lista extensa de algoritmos soportados, resultando en paquetes de más de 1100 bytes.

Tamaño original del KEI del servidor: 1146 bytes (observado en las capturas de la Parte 1)

Proceso de reducción:

Para reducir el tamaño del paquete KEI, se modificó la configuración del servidor SSH limitando los algoritmos ofrecidos a uno solo por categoría. Se editó el archivo `/etc/ssh/sshd_config` agregando las siguientes líneas:

```
# Configuracion minimalista para KEI < 300 bytes
KexAlgorithms curve25519-sha256
HostKeyAlgorithms ssh-ed25519
Ciphers aes128-ctr
MACs hmac-sha2-256
```

Después de reiniciar el servicio SSH, se capturó nuevamente el tráfico de una conexión desde un cliente.

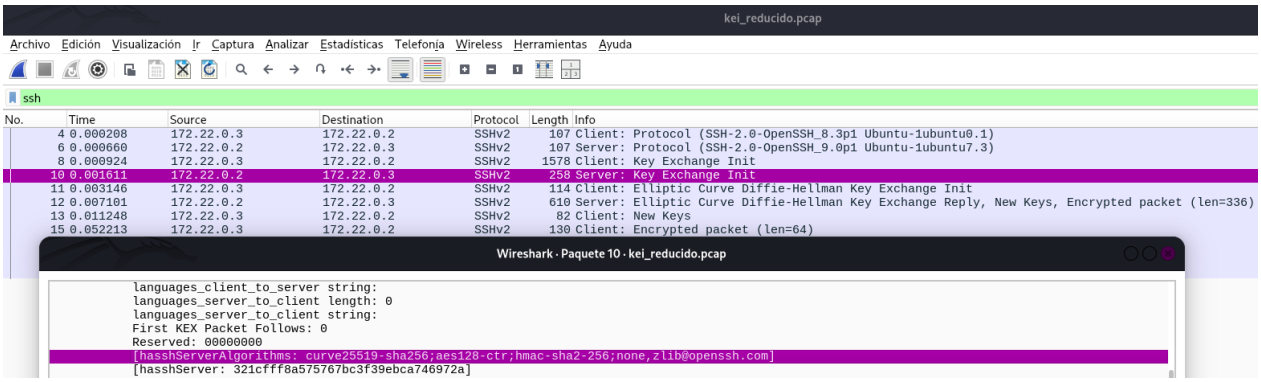

Figura 13: Captura mostrando el KEI del servidor reducido a 258 bytes

Resultado obtenido:
Al analizar la captura (Figura 13) se observa en el panel superior:

- **Paquete #4:** Client: Protocol - Banner `SSH-2.0-OpenSSH_8.3p1 Ubuntu-1ubuntu0.1` (107 bytes)
- **Paquete #6:** Server: Protocol - Banner `SSH-2.0-OpenSSH_9.0p1 Ubuntu-1ubuntu7.3` (107 bytes)
- **Paquete #8:** Client: Key Exchange Init - 1578 bytes (sin cambios)
- **Paquete #10:** Server: Key Exchange Init - **258 bytes** (resaltado en morado, reducido desde 1146 bytes)
- **Paquete #11:** Client: Elliptic Curve Diffie-Hellman Key Exchange Init
- **Paquete #12:** Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet

En el panel inferior se observa el contenido del paquete #10 expandido, mostrando la lista reducida de algoritmos del servidor. Se puede ver claramente que `hasshServerAlgorithms` contiene únicamente: `curve25519-sha256;aes128-ctr;hmac-sha2-256;none,zlib@openssh.com`, confirmando que solo se ofrece un algoritmo por categoría.

Reducción lograda: 888 bytes (77 % de reducción). El KEI del servidor pasó de 1146 bytes a 258 bytes, cumpliendo exitosamente el objetivo de ser menor a 300 bytes.

Algoritmos seleccionados:

- **curve25519-sha256:** Algoritmo de intercambio de claves moderno y eficiente
- **ssh-ed25519:** Clave de host basada en curva elíptica, más pequeña que RSA
- **aes128-ctr:** Cifrado simétrico robusto y ampliamente soportado
- **hmac-sha2-256:** MAC con buen balance entre seguridad y rendimiento

Conclusión: Se logró reducir exitosamente el tamaño del paquete Key Exchange Init del servidor de 1146 bytes a 258 bytes, cumpliendo con el requisito de ser menor a 300 bytes. La reducción se logró limitando cada categoría de algoritmos a una única opción segura y moderna.

4. Desarrollo (Parte 4)

4.1. Explicación OpenSSH en general

OpenSSH (Open Secure Shell) es una suite de herramientas de conectividad segura basada en el protocolo SSH. Proporciona acceso remoto cifrado y se ha convertido en el estándar para administración remota de sistemas Unix y Linux.

Arquitectura del protocolo SSH:

El protocolo SSH se estructura en tres capas principales:

1. **Capa de Transporte:** Proporciona autenticación del servidor, establece confidencialidad mediante cifrado y garantiza integridad de datos. Se ejecuta sobre TCP (puerto 22).

2. **Capa de Autenticación:** Autentica al cliente ante el servidor mediante contraseña, clave pública u otros métodos. Se ejecuta sobre la capa de transporte ya cifrada.

3. **Capa de Conexión:** Multiplexa múltiples canales lógicos sobre una única conexión SSH, soportando sesiones interactivas, ejecución remota y reenvío de puertos.

Proceso de establecimiento de conexión observado en las capturas:

1. Handshake TCP (SYN, SYN-ACK, ACK)

2. Intercambio de versiones de protocolo (texto plano)

3. Key Exchange Init (negociación de algoritmos)

4. Diffie-Hellman (derivación de secreto compartido)

5. New Keys (activación del cifrado)

6. Autenticación del usuario (ya cifrada)

7. Establecimiento de canales

4.2. Capas de Seguridad en OpenSSH

Basándose en el análisis del tráfico SSH interceptado durante el laboratorio, se evalúa cómo OpenSSH implementa los principios de seguridad de la información:

4.2.1. Confidencialidad - Cumplido

Implementación: SSH utiliza cifrado simétrico después del Key Exchange (AES-128-CTR, ChaCha20-Poly1305) con claves de sesión únicas derivadas de Diffie-Hellman. Se generan claves separadas para cada dirección de comunicación (cliente→servidor, servidor→cliente), garantizando que la interceptación de una clave no comprometa ambas direcciones.

Evidencia en las capturas: Los paquetes anteriores a "New Keys" contienen información en texto plano (versiones, algoritmos), pero después de este mensaje todos los paquetes aparecen como "Encrypted packet". El contenido de comandos y contraseñas no es visible en las capturas, confirmando el cifrado efectivo de la sesión.

4.2.2. Integridad - Cumplido

Implementación: SSH utiliza Message Authentication Codes (MAC) en cada paquete, típicamente HMAC-SHA2-256 o HMAC-SHA2-512. Las versiones modernas implementan modos ETM (Encrypt-Then-MAC) que calculan el MAC sobre el texto cifrado, proporcionando mayor seguridad. Los números de secuencia implícitos previenen ataques de replay.

Evidencia en las capturas: Todos los clientes proponen algoritmos MAC robustos en el KEI, con los clientes modernos (C3, C4) priorizando algoritmos ETM. El tamaño de paquetes cifrados incluye bytes adicionales para el MAC, verificable en el análisis de Wireshark.

4.2.3. Autenticidad - Cumplido

Implementación: SSH implementa autenticación mutua mediante criptografía asimétrica. El servidor posee un par de claves de host y firma el intercambio con su clave privada, mientras el cliente verifica la firma usando la clave pública del servidor. El usuario se autentica mediante contraseña cifrada o clave pública.

Evidencia en las capturas: El paquete "Elliptic Curve Diffie-Hellman Key Exchange Reply" incluye la firma digital del servidor. Los algoritmos de clave de host negociados (ssh-ed25519, rsa-sha2-512) proporcionan firmas robustas. En primera conexión, el cliente solicita confirmación del fingerprint del servidor.

4.2.4. No Repudio - Parcial

Implementación: SSH utiliza firmas digitales que proporcionan prueba criptográfica de identidad, logging de conexiones y comandos en el servidor, y claves únicas por usuario que permiten atribuir acciones específicas.

Limitaciones observadas: No existen timestamps firmados por terceros confiables, no hay PKI con CAs por defecto, los logs pueden ser modificados por administradores con privilegios, y la autenticación por contraseña (usada en el laboratorio) no proporciona no repudio fuerte.

Conclusión: SSH proporciona no repudio técnico mediante firmas digitales, pero no alcanza el nivel de no repudio legal que requeriría una PKI completa con CAs confiables y timestamps firmados.

4.2.5. Disponibilidad - No Cumplido

Análisis: SSH no incluye mecanismos inherentes de alta disponibilidad, no protege contra ataques de denegación de servicio (DoS), no proporciona redundancia a nivel de protocolo y depende completamente de la infraestructura subyacente.

Evidencia en las capturas: El handshake TCP muestra la dependencia de la red subyacente. No existen mecanismos de failover en el protocolo, y una conexión TCP interrumpida termina inmediatamente la sesión SSH sin posibilidad de recuperación automática.

Justificación: La disponibilidad no es un objetivo de diseño del protocolo SSH. Este principio debe implementarse mediante balanceadores de carga, múltiples servidores, protección DDoS y redundancia de infraestructura.

4.3. Identificación de que protocolos no se cumplen

Basándose en el análisis del tráfico SSH interceptado, se concluye:

Principios plenamente cumplidos: Confidencialidad mediante cifrado robusto de extremo a extremo, Integridad con MACs criptográficos en cada paquete, y Autenticidad () a través de autenticación mutua con firmas digitales.

Principios parcialmente cumplidos: No Repudio presenta firmas digitales, pero sin PKI completa ni timestamps de terceros. Es suficiente para la mayoría de casos de uso, pero no para escenarios legales estrictos que requieren evidencia irrefutable ante tribunales.

Principios no cumplidos: Disponibilidad no es objetivo del protocolo SSH. Este principio debe implementarse a nivel de infraestructura mediante balanceadores de carga, múltiples servidores, protección DDoS y redundancia de red.

Conclusión del análisis: SSH está diseñado para garantizar comunicaciones seguras (confidencialidad, integridad y autenticidad), cumpliendo estos objetivos de manera excelente. El no repudio se cumple parcialmente, siendo suficiente para la mayoría de casos pero no para escenarios legales estrictos. La disponibilidad queda fuera del alcance del protocolo y debe ser manejada por la infraestructura.

Conclusiones y comentarios

Este laboratorio permitió realizar un análisis profundo del protocolo SSH mediante la captura y análisis de tráfico de red entre diferentes versiones de OpenSSH. Las principales conclusiones son:

Evolución de OpenSSH: Se observó claramente la evolución de las políticas de seguridad a través de las versiones analizadas (7.3p1, 7.7p1, 8.3p1 y 9.0p1). La tendencia muestra eliminación de algoritmos débiles, incorporación de grupos Diffie-Hellman más grandes y optimización del handshake inicial.

Fingerprinting SSH: La técnica de identificación basada en el tamaño del paquete Key Exchange Init y el análisis del banner de protocolo demostró ser efectiva. Cada versión de OpenSSH tiene una firma única determinada por los algoritmos soportados, permitiendo identificación precisa del cliente utilizado.

Manipulación del tráfico: Se comprobó que es posible modificar las características del tráfico SSH mediante la configuración del servidor. La reducción del KEI de 1146 bytes a 258 bytes demuestra la flexibilidad del protocolo para crear firmas de tráfico únicas.

Principios de seguridad: El análisis confirmó que SSH cumple plenamente con confidencialidad, integridad y autenticidad mediante cifrado robusto, MACs criptográficos y firmas digitales. El no repudio se cumple parcialmente y la disponibilidad debe implementarse a nivel de infraestructura.

Este laboratorio proporcionó una comprensión práctica de los mecanismos criptográficos que sustentan SSH, uno de los protocolos más importantes para la administración remota segura de sistemas.

- Repositorio de github: https://github.com/matiasts4/Lab-Criptografia