Universidad Central del Ecuador

Facultad de Ingeniería y Ciencias Aplicadas

Computación

Criptografía y Seguridad de la Información

Tema:

Aplicaciones Criptográficas

Curso:

C8-001

Estudiantes:

- Michael Barrionuevo
- Kevin Celi
- Jhony Ninabanda
- Dylan Lema
- Joan Santamaria

30/11/2025

Análisis de la Aplicación Criptográfica: Secure Shell (SSH)

1. Definición de la Tecnología

El Secure Shell (SSH) es un protocolo de red criptográfico diseñado para permitir la operación segura de servicios de red sobre una red insegura. Su función principal es proporcionar un canal seguro mediante una arquitectura cliente-servidor, garantizando la confidencialidad e integridad de los datos intercambiados.

Según definen Ylonen y Lonvick en el estándar oficial (RFC 4251), SSH remplaza protocolos heredados inseguros (como Telnet o rlogin) que transmitían contraseñas en texto plano [1]. La tecnología opera típicamente en el puerto TCP 22 y establece una sesión cifrada antes de que el usuario se autentique, protegiendo así las credenciales contra ataques de interceptación (sniffing) [2].

2. Lista de Algoritmos Criptográficos Involucrados

SSH es flexible y negocia los algoritmos al inicio de la conexión. Según Barrett y Silverman, la seguridad de SSH se basa en una combinación de diferentes primitivas criptográficas [3]. Ellingwood detalla que el proceso de conexión utiliza los siguientes tipos de algoritmos [4]:

- **Intercambio de Claves (Key Exchange - KEX):** Se utilizan para establecer una clave de sesión compartida de forma segura sin transmitirla directamente.

  o *Algoritmos comunes:* Diffie-Hellman (DH), Elliptic Curve Diffie-Hellman (ECDH).

- **Cifrado Simétrico (Symmetric Encryption):** Una vez establecida la clave de sesión, estos algoritmos cifran el flujo de datos real (payload).

  o *Algoritmos comunes:* AES (Advanced Encryption Standard), ChaCha20, 3DES (obsoleto pero histórico), Blowfish.

- **Autenticación de Clave Pública (Public Key Authentication):** Utilizados para verificar la identidad del servidor y del usuario mediante pares de claves asimétricas.

  o *Algoritmos comunes:* RSA (Rivest-Shamir-Adleman), DSA (Digital Signature Algorithm), ECDSA (Elliptic Curve DSA), Ed25519.

- **Códigos de Autenticación de Mensajes (MAC):** Garantizan la integridad de los datos, asegurando que los paquetes no hayan sido alterados en tránsito.

o *Algoritmos comunes:* HMAC-SHA2 (SHA-256, SHA-512), HMAC-MD5 (obsoleto).

3. Lista de Protocolos Criptográficos Involucrados

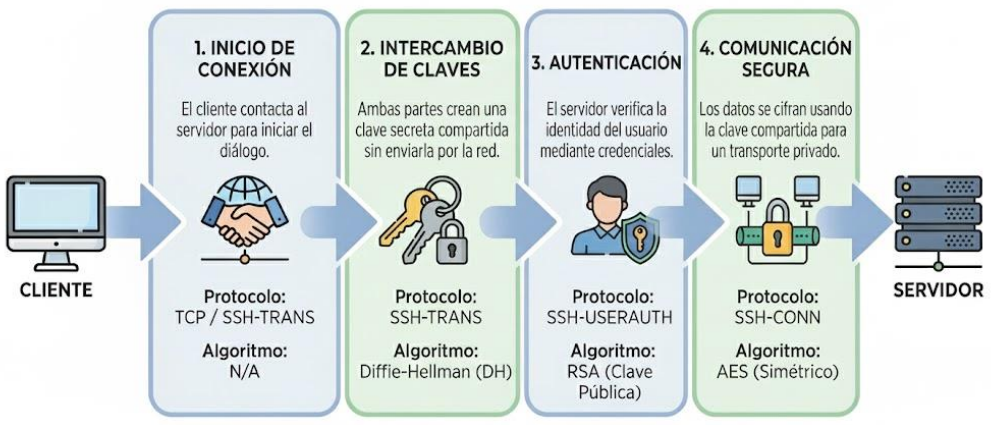
A diferencia de lo que se suele pensar, SSH no es un único protocolo monolítico. Como explica la arquitectura definida en el RFC 4251, SSH es una suite compuesta por tres protocolos jerárquicos principales [1]:

1. **SSH Transport Layer Protocol (SSH-TRANS):** Es la base de la suite. Provee la autenticación del servidor, confidencialidad e integridad. Es responsable de la negociación de algoritmos y el intercambio de claves inicial. Funciona típicamente sobre TCP.

2. **SSH User Authentication Protocol (SSH-USERAUTH):** Se ejecuta sobre la capa de transporte. Su única función es autenticar al cliente ante el servidor (ya sea mediante contraseña, clave pública, tarjeta inteligente, etc.).

3. **SSH Connection Protocol (SSH-CONN):** Se ejecuta sobre las dos anteriores. Multiplexa el túnel cifrado en varios canales lógicos, permitiendo múltiples sesiones simultáneas (como una terminal interactiva y una transferencia de archivos) sobre una única conexión física.

4. Diseño esquemático de su funcionamiento.



5. Escenarios de Uso Frecuente

A continuación, se describen tres escenarios donde la aplicación de SSH es crítica:

1. **Administración Remota de Servidores (Remote Administration):** Este es el uso más extendido. Permite a los administradores de sistemas gestionar servidores (especialmente en entornos Linux/Unix) a través de una interfaz de línea de comandos segura desde cualquier lugar del mundo. Gardoki destaca que una correcta configuración en este escenario es vital para evitar accesos no autorizados en entornos empresariales [5].

2. **Transferencia Segura de Archivos (SFTP/SCP):** SSH sirve como túnel para protocolos de transferencia de archivos. SFTP (SSH File Transfer Protocol) utiliza la capa de conexión de SSH para mover datos. A diferencia de FTP tradicional, SFTP cifra tanto los comandos como los datos, protegiendo la información sensible durante el tránsito [3].

3. **Tunelización y Reenvío de Puertos (Port Forwarding):** SSH permite encapsular otros protocolos de red. Por ejemplo, se puede crear un túnel SSH para acceder a una base de datos interna que no está expuesta a internet, o para navegar de forma segura (como un proxy) desde una red Wi-Fi pública. Anbalagan explora cómo estas capacidades pueden extenderse incluso para autenticación sobre HTTP, demostrando la versatilidad del protocolo más allá de la simple terminal [2].

6. 5 preguntas relacionadas con la tecnología.

1.- ¿Cuál es la función principal del protocolo Secure Shell (SSH) según la definición de la tecnología?

- A) Transmitir contraseñas en texto plano para compatibilidad con protocolos heredados.
- B) Operar exclusivamente en el puerto TCP 80 para la transferencia de archivos.
- C) Proporcionar un canal seguro garantizando la confidencialidad e integridad de los datos intercambiados mediante una arquitectura cliente-servidor.
- D) Ser un protocolo monolítico que realiza la transferencia de archivos y la autenticación simultáneamente.

- E) Remplazar el protocolo Telnet, operando en la capa de conexión (SSH-CONN) como única función.

Respuesta: C

2.- ¿Cuáles de los siguientes son tipos de algoritmos criptográficos utilizados por SSH en su proceso de conexión? (Elija 3)

- A) Algoritmos de Codificación de Video (Video Encoding).
- B) Intercambio de Claves (Key Exchange - KEX).
- C) Cifrado Simétrico (Symmetric Encryption).
- D) Algoritmos de Compresión de Datos (Data Compression).
- E) Autenticación de Clave Pública (Public Key Authentication).

Respuesta: B, C, E

3.- Empareja cada algoritmo criptográfico con la categoría o función principal dentro del proceso de conexión SSH:

Algoritmo	**Categoría / Función**
1. AES	A. Intercambio de Claves (KEX)
2. Diffie-Hellman (DH)	B. Códigos de Autenticación de Mensajes (MAC)
3. RSA	C. Cifrado Simétrico
4. HMAC-SHA2	D. Algoritmos heredados inseguros reemplazados por SSH
5. Telnet	E. Autenticación de Clave Pública

Respuesta: 1-C, 2-A, 3-E, 4-B, 5-D

4.- ¿Qué tipo de cifrado utiliza SSH, una vez establecida la clave de sesión, para cifrar el flujo de datos real?

Respuesta: Simétrico

5.- VERDADERO / FALSO: El protocolo Secure Shell (SSH) se diseñó para permitir la operación segura de servicios de red sobre una red insegura.

Respuesta: Verdadero

<u>7.- Ejemplo de implementación práctica.</u>

Demostración Didáctica de Simulación Mini-SSH

A continuación, se mostrará un código escrito en Python que simula de manera didáctica los pasos cruciales del establecimiento de una sesión Secure Shell (SSH), enfocándose en la combinación de criptografía asimétrica y simétrica. El script utiliza la librería cryptography para replicar el proceso de intercambio inicial de una clave de sesión segura, seguido por el cifrado y descifrado de comandos y respuestas, ilustrando cómo se logra la confidencialidad en el canal de comunicación.

Servidor Didáctico

La sección del Servidor Didáctico es responsable de la preparación del entorno seguro. Inicialmente, genera un par de claves asimétricas RSA (pública y privada), siendo la clave privada el secreto intransferible del servidor. Su función esencial es esperar la clave de sesión simétrica generada por el cliente y descifrarla utilizando su clave privada RSA, garantizando que solo el servidor legítimo pueda establecer el canal. Una vez descifrada, el servidor utiliza esta clave simétrica para crear un cifrador Fernet (simulando un algoritmo simétrico rápido como AES), que luego emplea para procesar y responder a los comandos entrantes.

```python
# ==========================================
#             SERVIDOR DIDÁCTICO
# ==========================================
class ServidorSSH:
    def __init__(self):
        print("🛠️ Servidor: Generando claves RSA...")

        # Crear un par de claves RSA  (Rivest-Shamir-Adleman).
        # Esta es CRIPTOGRAFÍA ASIMÉTRICA:
        # - private_key: clave privada (NO se comparte)
        # - public_key: clave pública (se comparte)
        # SSH real también usa claves públicas, pero RSA no siempre.
        self.private_key = rsa.generate_private_key(
            public_exponent=65537,  # valor estándar en RSA
            key_size=2048           # tamaño seguro hoy en día
        )

        # Extraer clave pública desde la clave privada
        self.public_key = self.private_key.public_key()

        # Variables para almacenar la clave simétrica de sesión
        self.session_key = None
        self.cipher = None

    def obtener_clave_publica(self):
        """Devuelve la clave pública.
        El cliente la recibirá y la usará para cifrar.
        """

        return self.public_key.public_bytes(
            encoding=serialization.Encoding.PEM,              # Formato de texto
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        )
```

```python
def recibir_clave_simetrica(self, clave_cifrada):
    """Recibe la clave simétrica cifrada por el cliente."""
    print("🛠️ Servidor: Descifrando clave simétrica enviada por el cliente...")

    # Aquí el servidor DESCIFRA usando su CLAVE PRIVADA RSA.
    # Esto demuestra:
    #  - Solo el servidor podía descifrar la clave simétrica
    self.session_key = self.private_key.decrypt(
        clave_cifrada,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )

    # Crear un cifrador simétrico Fernet
    # Aquí empieza el canal seguro.
    self.cipher = Fernet(self.session_key)

    print("🔐 Servidor: Clave de sesión establecida con éxito.\n")

def procesar_comando(self, comando_cifrado):
    """El servidor descifra el comando, procesa una respuesta y la cifra."""

    # Descifrado simétrico con la clave de sesión.
    # Esta es CRIPTOGRAFÍA SIMÉTRICA.
    comando = self.cipher.decrypt(comando_cifrado).decode()
    print(f"📥 Servidor recibió comando:   {comando}")

    # Respuesta simulada (en SSH real aquí se ejecuta en la terminal)
    respuesta = f"Ejecutado correctamente → {comando}"

    # Devuelve la respuesta cifrada simétricamente
    return self.cipher.encrypt(respuesta.encode())
```

Cliente Didáctico

El componente Cliente modela el inicio de la conexión segura. Después de cargar la clave pública del servidor, el cliente genera su propia clave de sesión simétrica (Fernet), la cual será utilizada para toda la comunicación posterior de datos. Es crucial que esta clave de sesión sea transmitida de forma segura, por lo que el cliente utiliza la clave pública RSA del servidor para cifrar la clave simétrica. Este paso emula el "Intercambio de Claves" de SSH (como Diffie-Hellman), logrando que la clave simétrica viaje cifrada por el canal inseguro inicial, para luego ser usada en el envío y recepción de comandos de forma rápida y confidencial.

```python
# ==========================================
#                CLIENTE
# ==========================================
class ClienteSSH:
    def __init__(self, clave_publica_servidor):
        print("🔑 Cliente: Cargando clave pública del servidor...")

        # Carga de la clave pública del servidor.
        # Esta clave se usará SOLO para cifrar.
        # El cliente NO puede descifrar lo que cifra con ella.
        self.server_public_key = serialization.load_pem_public_key(
            clave_publica_servidor
        )

        # Crear clave simétrica para la sesión segura.
        # Esta clave será compartida SOLO UNA VEZ.
        print("🔑 Cliente: Generando clave de sesión (simétrica AES/Fernet)...")
        self.session_key = Fernet.generate_key()
        self.cipher = Fernet(self.session_key)

    def cifrar_clave_simetrica(self):
        """Cifra la clave simétrica usando RSA (clave pública del servidor)."""

        print("🔑 Cliente: Cifrando clave de sesión con RSA...")

        # Aquí el cliente usa la CLAVE PÚBLICA del servidor para cifrar.
        # Solo el servidor con su CLAVE PRIVADA puede descifrar.
        # Esto es criptografía ASIMÉTRICA.
        return self.server_public_key.encrypt(
            self.session_key,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )

    def enviar_comando(self, texto):
        """Cifra un comando con la clave simétrica y lo envía."""

        print(f"\n📤 Cliente enviando comando cifrado:   {texto}")

        # Esto usa CIFRADO SIMÉTRICO (rápido), típico en SSH.
        return self.cipher.encrypt(texto.encode())

    def recibir_respuesta(self, respuesta_cifrada):
        """Descifra la respuesta usando la clave simétrica."""

        respuesta = self.cipher.decrypt(respuesta_cifrada).decode()
        print(f"📥 Cliente recibió respuesta:   {respuesta}")

        return respuesta
```

Prueba de Ejecución

La sección final de "Simulación Tipo SSH Completa" coordina el diálogo entre el cliente y el servidor para demostrar el flujo de trabajo completo. Comienza con la generación de claves y el intercambio de la clave pública, seguido por el crucial paso de cifrado y envío de la clave de sesión simétrica por parte del cliente. Una vez que ambos lados poseen la

misma clave simétrica secreta, el cliente envía un comando ("ls -al") cifrado simétricamente. El servidor recibe, descifra, simula el procesamiento y devuelve una respuesta cifrada, que finalmente el cliente descifra y muestra en la consola. Este resultado demuestra la seguridad del canal, ya que todos los datos sensibles fueron protegidos por criptografía.

```
# ================================================
#           SIMULACIÓN TIPO SSH COMPLETA
# ================================================

print("================================================")
print("           SIMULACIÓN DE MINI SSH")
print("================================================\n")

# 1) Crear servidor y generar claves RSA
servidor = ServidorSSH()

# 2) Cliente recibe la clave pública del servidor (como hace SSH real con las host keys)
clave_publica = servidor.obtener_clave_publica()

# 3) Crear cliente con la clave pública del servidor
cliente = ClienteSSH(clave_publica)

# 4) Cliente cifra su clave simétrica y la "manda" al servidor
clave_simetrica_cifrada = cliente.cifrar_clave_simetrica()

# 5) Servidor descifra la clave simétrica y establece la sesión
servidor.recibir_clave_simetrica(clave_simetrica_cifrada)

# 6) Cliente envía un comando cifrado
comando = cliente.enviar_comando("ls -al")

# 7) Servidor procesa y responde
respuesta_cifrada = servidor.procesar_comando(comando)

# 8) Cliente descifra la respuesta
cliente.recibir_respuesta(respuesta_cifrada)

print("\n FIN DE LA DEMO")
```

```
================================================
           SIMULACIÓN DE MINI SSH
================================================

Servidor: Generando claves RSA...
Cliente: Cargando clave pública del servidor...
Cliente: Generando clave de sesión (simétrica AES/Fernet)...
Cliente: Cifrando clave de sesión con RSA...
Servidor: Descifrando clave simétrica enviada por el cliente...
Servidor: Clave de sesión establecida con éxito.


Cliente enviando comando cifrado:   ls -al
Servidor recibió comando:   ls -al
Cliente recibió respuesta:   Ejecutado correctamente → ls -al

FIN DE LA DEMO
```

Referencias

[1] T. Ylonen y C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," RFC 4251, Internet Engineering Task Force, Jan. 2006. [En línea]. Disponible: https://doi.org/10.17487/RFC4251

[2] S. Anbalagan, "Secure Shell (SSH): Public key authentication over Hypertext Transfer Protocol (HTTP)," arXiv preprint arXiv:1506.05073, 2015. [En línea]. Disponible: https://arxiv.org/abs/1506.05073

[3] D. J. Barrett y R. E. Silverman, SSH, The Secure Shell: The Definitive Guide, 2nd ed. Sebastopol, CA: O'Reilly Media, 2001. ISBN: 978-0596008956.

[4] J. Ellingwood, "Understanding the SSH Encryption and Connection Process," DigitalOcean Tutorials, 2022. [En línea]. Disponible: https://www.digitalocean.com/community/tutorials/understanding-the-ssh-encryption-and-connection-process

[5] C. Gardoki, "Qué es el protocolo SSH y cómo configurarlo para mejorar la seguridad de acceso a los servidores Linux," Hostalia Whitepapers, pp. 1-8. [En línea]. Disponible: https://www.hostalia.com/white-papers/protocolo-ssh/