

Criptografía y Protección de Datos

La criptografía es el uso de técnicas matemáticas para asegurar la información y las comunicaciones, garantizando que solo las personas a quienes va dirigida puedan leerla y procesarla.

Clasificación general de algoritmos criptográficos

A. Según el tipo de clave utilizada

Tipo	Definición	Ejemplo de Algoritmo
Criptografía Simétrica	Usa la misma clave para cifrar y descifrar. Es muy rápida y eficiente para grandes volúmenes de datos, pero el reto es cómo compartir la clave de forma segura. Ejemplo, descargas de archivos ISO	AES-256, DES, ChaCha20
Criptografía Asimétrica	En la criptografía asimétrica, el uso de las llaves se invierte dependiendo del objetivo que quieras lograr: Para Confidencialidad (Proteger un mensaje): - Se usa la Clave Pública para cifrar y la Clave Privada para descifrar. - Cualquiera puede usar tu Clave Pública para cerrar la caja fuerte (cifrar), pero solo tú, con tu Clave Privada, puedes abrirla (descifrar). Para Firmas Digitales (Garantizar identidad e integridad): - El emisor toma el Hash de un archivo y lo cifra con su Clave Privada. - A esto se le llama Firma Digital, y garantiza tanto la integridad del documento como la identidad irrefutable de quien lo envió. En ambos casos, la regla de oro se mantiene intacta: la Clave Privada nunca debe compartirse. Es más lenta, ideal para intercambiar claves o firmas digitales.	RSA, ECC (Curvas Elípticas)

Tipo	Definición	Ejemplo de Algoritmo
Funciones Hash	Algoritmos unidireccionales que transforman cualquier cantidad de datos en una cadena de texto de longitud fija. No se puede "descifrar" o revertir. Sirven para verificar integridad.	SHA-256, MD5 (obsoleto)

Práctica de Laboratorio: Cifrado, Descifrado y Verificación de Integridad

Objetivo de la práctica:

- Comprender la diferencia práctica entre cifrado simétrico, asimétrico y hashing.
- Demostrar cómo se protege la confidencialidad de un archivo mediante una contraseña.
- Comprobar cómo un solo cambio en un archivo altera por completo su firma criptográfica (Hash).

Herramientas necesarias:

- Entorno: Ubuntu WSL o consola de Linux (también compatible con Git Bash en Windows).
- Herramienta principal: openssl y sha256sum (preinstalados en la mayoría de distribuciones Linux/WSL).

Guía de Ejecución:

Para la versión en Windows, ver anexo 1

Paso 1: Preparación y Hashing (Integridad)

Vamos a crear un documento importante y sacarle su "huella digital" (Hash).

1. Abre tu terminal de Ubuntu WSL.
2. Crea un archivo de texto con un mensaje confidencial:

Bash
echo "Este es un secreto corporativo por valor de 1 millon de dolares" > secreto_1.txt

3. Calcula el Hash SHA-256 del archivo:

Bash
sha256sum secreto_1.txt
(Anota mentalmente los primeros caracteres del código generado).

4. Modifica ligeramente el archivo (simulando que un atacante alteró el mensaje):

Bash
echo "Este es un secreto corporativo por valor de 2 millones de dolares" > secreto_2.txt

5. Vuelve a calcular el Hash:

Bash
sha256sum secreto_2.txt

Análisis: Notaremos que, aunque solo cambió un número, la cadena alfanumérica resultante es completamente diferente. Así se garantiza que un archivo no ha sido manipulado.

Ejemplo: Descarga de Sistemas Operativos o Software: Cuando un usuario descarga una imagen ISO de Ubuntu o un programa de código abierto, la página web oficial siempre publica el Hash (SHA-256 o MD5) junto al botón de descarga. El archivo viaja por la red sin cifrar, pero el usuario puede calcular el hash en su computadora al finalizar para confirmar que la descarga no se corrompió y que ningún atacante inyectó malware en el ejecutable.

Nota sobre Firmas Digitales: El Hash garantiza que el archivo no fue alterado, pero el archivo sigue siendo legible. En el mundo real, para evitar que un atacante modifique el archivo y genere un nuevo Hash falso, el emisor toma este Hash y lo cifra con su **Clave Privada**. A esto se le llama *Firma Digital*, y garantiza tanto la integridad del documento como la identidad irrefutable de quien lo envió.

Paso 2: Cifrado Simétrico (Confidencialidad)

Vamos a proteger el archivo usando el estándar de la industria (AES-256).

1. Cifra el archivo secreto.txt usando AES-256. El sistema te pedirá que inventes una contraseña:

```
Bash  
openssl enc -aes-256-cbc -salt -in secreto.txt -out secreto.enc
```

2. Intenta leer el archivo cifrado para comprobar que es ilegible:

```
Bash  
cat secreto.enc  
(Verán caracteres extraños en la consola, demostrando que la información está protegida).
```

3. Elimina el archivo original en texto plano para no dejar rastros:

```
Bash  
rm secreto.txt
```

4. **Para descifrarlo:** Simula que eres el receptor legítimo e ingresa la contraseña cuando se solicite:

```
Bash  
openssl enc -aes-256-cbc -d -in secreto.enc -out secreto_recuperado.txt
```

5. Lee el archivo recuperado:

```
Bash  
cat secreto_recuperado.txt
```

Paso 3: Cifrado Asimétrico Básico (Generación de Llaves)

Demostraremos cómo se generan las famosas claves pública y privada.

1. Genera la **Clave Privada** (Esta nunca debe compartirse):

```
Bash  
openssl genpkey -algorithm RSA -out clave_privada.pem -pkeyopt  
rsa_keygen_bits:2048
```

2. Extrae la **Clave Pública** a partir de la privada (Esta es la que le das a todo el mundo para que te envíen mensajes seguros):

```
Bash
openssl rsa -pubout -in clave_privada.pem -out clave_publica.pem
```

3. Abrir y observar ambas claves usando el comando cat:

```
Bash
cat clave_publica.pem
cat clave_privada.pem
```

Explicación Técnica:

- El cifrado simétrico (AES) es como una caja fuerte que usa la misma llave para abrir y cerrar.
- El cifrado asimétrico (RSA) resuelve el problema de tener que enviarle la llave a la otra persona por internet. Cualquiera puede usar tu *Clave Pública* para cerrar la caja fuerte (cifrar), pero solo tú, con tu *Clave Privada*, puedes abrirla (descifrar).
- Estas técnicas son la base matemática de cómo funcionan los certificados HTTPS, los cuales bloquean ataques como el *Sniffing* y alertan al usuario ante redirecciones maliciosas provocadas por un *DNS Spoofing*.

Paso 4: Encriptar y Desencriptar con Criptografía Asimétrica (WSL)

Vamos a utilizar las llaves que generamos en el paso anterior para enviar un mensaje seguro, demostrando que lo que se cifra con la clave pública solo puede abrirse con la clave privada.

1. Crea un nuevo mensaje corto:

```
Bash
echo "Atacaremos al amanecer" > mensaje.txt
```

2. **Cifrar usando la Clave Pública:** Simula que eres un cliente enviando el mensaje al servidor. Usarás el "candado" público para proteger el archivo:

```
Bash
openssl pkeyutl -encrypt -pubin -inkey clave_publica.pem -in mensaje.txt -out
mensaje_cifrado.enc
```

3. Verifica que está encriptado:

Bash

```
cat mensaje_cifrado.enc
```

(Veremos símbolos ilegibles en la terminal).

4. **Descifrar usando la Clave Privada:** Ahora simula que eres el dueño del servidor y usas tu llave maestra para leer el mensaje:

Bash

```
openssl pkeyutil -decrypt -inkey clave_privada.pem -in mensaje_cifrado.enc -out  
mensaje_descifrado.txt
```

5. **Lee el mensaje recuperado:**

Bash

```
cat mensaje_descifrado.txt
```

Aquí se puede contrastar perfectamente ambos mundos desde su entorno Linux: en el paso de cifrado simétrico usaron una contraseña que tuvieron que inventar y digitar manualmente, mientras que aquí utilizan archivos matemáticos (.pem), automatizando la seguridad sin necesidad de teclear contraseñas frágiles. Esto sienta las bases para entender cómo negocian su seguridad protocolos web reales.

Usamos un texto tan corto: como la criptografía asimétrica (RSA) consume muchos recursos de CPU y es lenta, no se usa para cifrar archivos grandes (como sí lo hace AES en el Paso 2), sino para información pequeña como hashes o llaves.

Preguntas

1. Si un atacante intercepta tu tráfico de red usando Wireshark (como en la práctica de Sniffing) y captura el archivo secreto.enc creado en el Paso 2, ¿ha comprometido la información? *(Respuesta esperada: No, porque el archivo viaja cifrado. El atacante solo verá ruido a menos que conozca la contraseña simétrica o logre romperla por fuerza bruta).*
2. ¿Por qué usamos SHA-256 para guardar contraseñas en bases de datos en lugar de cifrarlas con AES? *(Respuesta esperada: Porque el Hash no se puede revertir. Si un atacante roba la base de datos (Ej. Inyección SQL), no podrá obtener las contraseñas originales, solo sus Hashes).*

Aspectos Avanzados de Criptografía y Seguridad Operativa

Una vez que comprendemos cómo funcionan los algoritmos de cifrado y el hashing, es vital entender cómo se implementan, mantienen y regulan en el mundo corporativo real.

Protocolos Criptográficos

Los algoritmos (como AES o RSA) casi nunca se utilizan de forma aislada. Se integran dentro de **protocolos criptográficos**, que son conjuntos de reglas estandarizadas que definen cómo dos sistemas deben negociar, autenticarse y compartir datos de forma segura.

- **TLS (Transport Layer Security):** Es el estándar de oro en internet. Utiliza una combinación maestra: emplea criptografía asimétrica (certificados) para que el servidor y el cliente se identifiquen y compartan una llave secreta de forma segura, y luego pasa a usar criptografía simétrica para cifrar la comunicación a gran velocidad.
- **SSH (Secure Shell):** El protocolo que configuramos en nuestra práctica entre Windows y Ubuntu WSL. Garantiza un canal de administración remota donde incluso el proceso de "login" es ilegible para atacantes en la red.
- **IPsec:** Utilizado a nivel de infraestructura para establecer Redes Privadas Virtuales (VPNs), cifrando los paquetes IP directamente desde el origen hasta el destino corporativo.

Relación con los Mecanismos de Seguridad Web

La criptografía es el cimiento sobre el cual los navegadores web modernos construyen sus defensas para mitigar los ataques del *OWASP Top 10*.

- **Mitigación del Sniffing y Spoofing:** Al implementar HTTPS (apoyado en TLS), se anulan por completo los ataques pasivos de interceptación. Además, mecanismos como **HSTS (HTTP Strict Transport Security)** obligan a los navegadores a conectarse únicamente por canales cifrados, evitando que un ataque de *DNS Spoofing* degrade la conexión a texto plano.
- **Protección de Sesiones:** En nuestro laboratorio de *XSS y Session Hijacking*, demostramos cómo se roba una cookie. Al utilizar el atributo criptográfico Secure al crear la cookie, el navegador se negará a enviarla si la red no está cifrada, protegiendo el token de sesión.
- **Integridad (CORS y CSP):** Las firmas digitales aseguran que los recursos cargados por una página (scripts, librerías) provengan de orígenes legítimos y no hayan sido inyectados por un *Man in the Middle*.

Consideraciones Operativas

Implementar criptografía en una empresa no es simplemente "activar un botón"; conlleva retos operativos diarios para los administradores de sistemas:

- **Gestión del Ciclo de Vida de las Claves:** Los certificados digitales tienen fecha de caducidad (por ejemplo, Let's Encrypt dura 90 días). Un certificado expirado hace que el navegador bloquee el acceso al sitio web, causando la caída inmediata del servicio de cara al usuario.
- **Costo de Rendimiento (Overhead):** Cifrar y descifrar paquetes exige poder de CPU. En arquitecturas en la nube o ataques masivos (como DDoS), el esfuerzo computacional para sostener miles de "saludos criptográficos" (TLS Handshakes) puede sobrecargar y tumbar los servidores.
- **Almacenamiento Seguro (Key Management):** ¿Dónde guardas tu Clave Privada? Si un servidor Ubuntu web es hackeado y el atacante roba la clave privada .pem, podrá descifrar el tráfico pasado y futuro, o suplantar la identidad de la empresa.

Cumplimiento y Estándares (Compliance)

A nivel institucional, el uso de la criptografía ha dejado de ser una recomendación técnica para convertirse en un **requisito legal y normativo**:

- **PCI-DSS:** Si un sistema procesa o transmite tarjetas de crédito, este estándar global prohíbe algoritmos obsoletos y exige estrictamente el uso de TLS 1.2 o superior.
- **Leyes de Privacidad (GDPR, HIPAA):** Las regulaciones de protección de datos exigen que la Información de Identificación Personal (PII) o los historiales médicos estén cifrados tanto en **tránsito** (mientras viajan por la red) como en **reposo** (guardados en la base de datos).
- **Estándares Gubernamentales (NIST, FIPS):** Instituciones de seguridad que evalúan matemáticamente qué algoritmos son seguros. Por ejemplo, dictaminan que SHA-256 es seguro, pero ordenan la retirada obligatoria de algoritmos rotos como MD5 o SHA-1.

Debate 1:

1. Si un administrador de sistemas configura su servidor web en WSL con un certificado HTTPS robusto, pero se olvida de renovarlo y caduca, ¿es esto un problema de confidencialidad o de disponibilidad?
2. ¿Por qué las leyes (Compliance) exigen que las contraseñas se guarden usando algoritmos *Hash* en lugar de cifrado simétrico reversible como AES?

Práctica de Laboratorio: Autenticación Segura con SSH (Criptografía Asimétrica en Acción)

Objetivo de la práctica:

- Implementar los conceptos teóricos de criptografía asimétrica mediante la generación de un par de claves (Pública y Privada).
- Configurar una conexión segura simulando un entorno de producción, donde un cliente (Windows) administra remotamente un servidor (Ubuntu WSL).
- Comprobar que el uso de protocolos cifrados como SSH es la contramedida mandatoria para mitigar ataques de captura de red (Sniffing), haciendo que el tráfico sea completamente ilegible.

Herramientas necesarias:

- Entorno: Ubuntu WSL (Servidor) y Windows (Cliente).
- Herramienta principal: OpenSSH.

Guía de Ejecución:

Paso 1: Habilitar el Servidor SSH en Ubuntu WSL (Servidor)

Primero, necesitamos encender el servicio que escuchará las peticiones de conexión segura en nuestra máquina Linux.

1. Abre tu terminal de **Ubuntu WSL**.
2. Reinstala o actualiza el servidor OpenSSH (por si no viene completo de fábrica en tu versión de WSL) e inicia el servicio:

Bash

sudo apt update

sudo apt install openssh-server -y

sudo service ssh start

3. Identifica la IP de tu servidor WSL ejecutando el comando:

Bash

ip a

Anotar la IP asignada a la interfaz eth0, por ejemplo: 172.19.73.227.

Paso 2: Generación de Claves Criptográficas en Windows (Cliente)

Ahora, desde el equipo del administrador (Windows), generaremos nuestras "credenciales matemáticas" para no depender de contraseñas.

1. Abre **PowerShell** o el **Símbolo del sistema (cmd)** en Windows.

Genera el par de claves usando el algoritmo RSA:

PowerShell

```
ssh-keygen -t rsa -b 4096
```

(Presiona Enter a todas las preguntas para aceptar la ruta por defecto, que suele ser C:\Users\TuUsuario\.ssh\id_rsa, y déjala sin contraseña adicional ("passphrase") para agilizar el laboratorio).

2. Se han creado dos archivos: id_rsa es la **Clave Privada** (la llave maestra que se queda en Windows) y id_rsa.pub es la **Clave Pública** (el candado que enviaremos a Linux).

Paso 3: Intercambio de la Clave Pública (Autorización)

Debemos "entregarle" nuestra clave pública a Ubuntu para que autorice nuestra entrada. Como WSL comparte el disco con Windows, podemos copiarla fácilmente.

1. Vuelve a tu terminal de **Ubuntu WSL**.

Asegúrate de que exista la carpeta .ssh oculta en tu directorio de Linux:

Bash

```
mkdir -p ~/.ssh
```

2. Extrae la clave pública de tu usuario de Windows (cambia TuUsuario por tu nombre de usuario real en Windows) y guárdala en las llaves autorizadas de Linux:

Bash

```
cat /mnt/c/Users/TuUsuario/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

3. Ajusta los permisos por seguridad (SSH es muy estricto con esto):

Bash

```
chmod 600 ~/.ssh/authorized_keys
```

Paso 4: Ejecutar la Conexión Segura

¡Es hora de conectar el cliente con el servidor!

1. Regresa a tu consola de **Windows (PowerShell)**.

Conéctate a tu servidor Linux usando la IP que anotaste en el Paso 1 (reemplaza usuario_linux por tu nombre de usuario en WSL):

PowerShell

`ssh usuario_linux@172.19.73.227`

2. Escribe yes si te pide confirmar la autenticidad del host la primera vez.

Resultado esperado: ¡Acceso concedido! Entrarás inmediatamente a la consola de Ubuntu WSL sin que te haya pedido contraseña.

Explicación Técnica :

1. **¿Qué acaba de pasar bajo el capó?** Ubuntu WSL tenía guardada tu *Clave Pública*. Cuando Windows intentó conectarse, Ubuntu le envió un "mensaje cifrado" que solo podía ser resuelto por quien tuviera la *Clave Privada*. Windows lo resolvió internamente, demostrando su identidad sin que ninguna credencial viajara por la red.
2. **Defensa contra el Sniffing:** Si un atacante interceptara esta red local (como se simuló en la práctica con Wireshark capturando peticiones POST en texto plano), no vería absolutamente nada útil. Todo lo que el administrador teclee desde Windows y lo que responda el servidor Linux estará envuelto en un túnel criptográfico irrompible.

Debate 2:

1. Si usamos un ataque de diccionario o fuerza bruta (técnicas de ataque a contraseñas) contra nuestro servidor SSH, ¿tendría éxito?
2. Durante un ataque de *Man in the Middle (MitM)*, si el atacante roba el archivo `id_rsa.pub` (Clave Pública) mientras viaja hacia el servidor, ¿puede hackearnos?

Trabajo independiente

- **Lectura:** Conceptos de Infraestructura de Clave Pública (PKI) y Certificados Digitales.
- **Investigación:** Investigar qué tipo de cifrado utiliza la aplicación WhatsApp para su "Cifrado de extremo a extremo" (Signal Protocol).
- **Práctica:** Cifrar un archivo PDF en Windows/WSL usando la clave pública generada en clase y descifrarlo usando la clave privada.

Anexo 1

Práctica de Laboratorio: Criptografía y Protección de Datos (Entorno Windows)

Objetivo de la práctica:

- Comprender la diferencia práctica entre cifrado simétrico, asimétrico y hashing.
- Demostrar cómo se protege la confidencialidad de un archivo mediante una contraseña.
- Comprobar cómo un solo cambio en un archivo altera por completo su firma criptográfica (Hash).
- Utilizar herramientas nativas de Windows (PowerShell) junto con el estándar de la industria (OpenSSL).

Guía de Ejecución:

Paso 1: Preparación y Hashing (Integridad en PowerShell)

Vamos a crear un documento importante y sacarle su "huella digital" (Hash) utilizando las herramientas nativas de Windows.

1. Abre **PowerShell**.
2. Crea un archivo de texto con un mensaje confidencial ejecutando:

```
PowerShell  
echo "Este es un secreto corporativo por valor de 1 millon de dolares" > secreto_1.txt
```

3. Calcula el Hash SHA-256 del archivo con el comando integrado de Windows:

```
PowerShell  
Get-FileHash -Path .\secreto_1.txt -Algorithm SHA256
```

4. Modifica ligeramente el archivo (simulando que un atacante alteró el mensaje):

```
PowerShell
echo "Este es un secreto corporativo por valor de 2 millones de dolares" >
secreto_2.txt
```

5. Vuelve a calcular el Hash:

```
PowerShell
Get-FileHash -Path .\secreto_2.txt -Algorithm SHA256
```

Análisis: Notaremos que, aunque solo cambió un número, la cadena alfanumérica resultante es completamente diferente.

Nota sobre Firmas Digitales: El Hash garantiza que el archivo no fue alterado, pero el archivo sigue siendo legible. En el mundo real, para evitar que un atacante modifique el archivo y genere un nuevo Hash falso, el emisor toma este Hash y lo cifra con su **Clave Privada**. A esto se le llama *Firma Digital*, y garantiza tanto la integridad del documento como la identidad irrefutable de quien lo envió.

Paso 2: Instalación del estándar criptográfico (OpenSSL)

Dado que Windows no incluye herramientas de cifrado avanzadas por defecto en la consola, instalaremos OpenSSL, el estándar universal de la industria.

1. En la misma ventana de PowerShell, instala OpenSSL usando el gestor de paquetes de Windows:

```
PowerShell
winget install openssl
(Acepta los términos si el sistema te lo solicita presionando la tecla 'Y').
```

2. **¡Importante!** Cierra la ventana de PowerShell y vuelve a abrir una nueva para que Windows reconozca el nuevo comando instalado.
3. Usamos un texto tan corto: como la criptografía asimétrica (RSA) consume muchos recursos de CPU y es lenta, no se usa para cifrar archivos grandes (como sí lo hace AES en el Paso 2), sino para información pequeña como hashes o llaves.

Paso 3: Cifrado Simétrico (Confidencialidad)

Vamos a proteger el archivo que creamos usando el estándar de la industria (AES-256).

1. En tu nueva ventana de PowerShell, cifra el archivo secreto.txt. El sistema te pedirá que inventes una contraseña:

```
PowerShell  
openssl enc -aes-256-cbc -salt -in secreto.txt -out secreto.enc
```

2. Intenta leer el archivo cifrado para comprobar que es ilegible:

```
PowerShell  
cat secreto.enc  
(Verán caracteres extraños y pitidos en la consola, demostrando que la información  
está protegida).
```

3. Elimina el archivo original en texto plano para no dejar rastros:

```
PowerShell  
rm secreto.txt
```

4. **Para descifrarlo:** Simula que eres el receptor legítimo e ingresa la contraseña cuando se solicite:

```
PowerShell  
openssl enc -aes-256-cbc -d -in secreto.enc -out secreto_recuperado.txt
```

5. Lee el archivo recuperado para confirmar el éxito:

```
PowerShell  
cat secreto_recuperado.txt
```

Paso 4: Cifrado Asimétrico Básico (Generación de Llaves)

Demostraremos cómo se generan las famosas claves pública y privada que se utilizan en protocolos como HTTPS o SSH.

1. Genera la **Clave Privada** (Esta nunca debe compartirse, es la que abre la caja fuerte):

2. PowerShell
openssl genpkey -algorithm RSA -out clave_privada.pem -pkeyopt
rsa_keygen_bits:2048

3. Extrae la **Clave Pública** a partir de la privada (Esta es la que le das a todo el mundo para que te envíen mensajes seguros):

PowerShell

```
openssl rsa -pubout -in clave_privada.pem -out clave_publica.pem
```

4. Abrir y observar ambas claves usando el comando cat:

PowerShell

```
cat clave_publica.pem
```

```
cat clave_privada.pem
```

Paso 5: Encriptar y Desencriptar con Criptografía Asimétrica

Ahora vamos a utilizar las llaves que generamos en el paso anterior para enviar un mensaje seguro, demostrando que lo que se cifra con la clave pública solo puede abrirse con la clave privada.

1. **Crea un nuevo mensaje corto:**

PowerShell

```
echo "Atcaremos al amanecer" > mensaje.txt
```

2. **Cifrar usando la Clave Pública:** Simula que eres un cliente enviando el mensaje al servidor. Usarás el "candado" público para proteger el archivo:

PowerShell

```
openssl pkeyutl -encrypt -pubin -inkey clave_publica.pem -in mensaje.txt -out  
mensaje_cifrado.enc
```

3. **Verifica que está encriptado:**

PowerShell

```
cat mensaje_cifrado.enc
```

(Los estudiantes verán símbolos ilegibles).

4. **Descifrar usando la Clave Privada:** Ahora simula que eres el dueño del servidor y usas tu llave maestra para leer el mensaje:

PowerShell

```
openssl pkeyutl -decrypt -inkey clave_privada.pem -in mensaje_cifrado.enc -out  
mensaje_descifrado.txt
```

5. **Lee el mensaje recuperado:**

PowerShell

```
cat mensaje_descifrado.txt
```

Nota: Con esto, se podrá contrastar perfectamente ambos mundos:

- En el **Paso 2** usaron una **contraseña** que tuvieron que inventar y digitar manualmente (Criptografía Simétrica).
- En este **Paso 5** usaron **archivos matemáticos (.pem)**, automatizando la seguridad sin necesidad de teclear contraseñas frágiles, que es exactamente la base de cómo navegan seguros hoy en día en internet.

Respuestas :

Debate 1:

1. Si un administrador de sistemas configura su servidor web en WSL con un certificado HTTPS robusto, pero se olvida de renovarlo y caduca, ¿es esto un problema de confidencialidad o de disponibilidad? *(Respuesta esperada: De disponibilidad, porque aunque los datos sigan estando seguros matemáticamente, los navegadores de los usuarios bloquearán el acceso a la página asumiendo que es insegura, "tumbando" el sitio por una mala gestión operativa).*
2. ¿Por qué las leyes (Compliance) exigen que las contraseñas se guarden usando algoritmos *Hash* en lugar de cifrado simétrico reversible como AES? *(Respuesta esperada: Para asegurar que si la base de datos es vulnerada mediante Inyección SQL, ni siquiera el administrador de la empresa ni el atacante puedan revertir el proceso y obtener las contraseñas en texto plano).*

Debate 2:

1. Si usamos un ataque de diccionario o fuerza bruta (técnicas de ataque a contraseñas) contra nuestro servidor SSH, ¿tendría éxito? *(Respuesta esperada: No, porque el servidor ahora puede configurarse para rechazar contraseñas y exigir exclusivamente una Clave Privada de 4096 bits, la cual es matemáticamente imposible de adivinar).*
2. Durante un ataque de *Man in the Middle (MitM)*, si el atacante roba el archivo `id_rsa.pub` (Clave Pública) mientras viaja hacia el servidor, ¿puede hackearnos? *(Respuesta esperada: No. La clave pública es de libre distribución; se usa para cifrar, pero no sirve para descifrar ni para autenticarse sin su clave privada correspondiente).*