Source: Bing Chat AI

Laboratorio 4. Aplicaciones de la Criptografía (II)

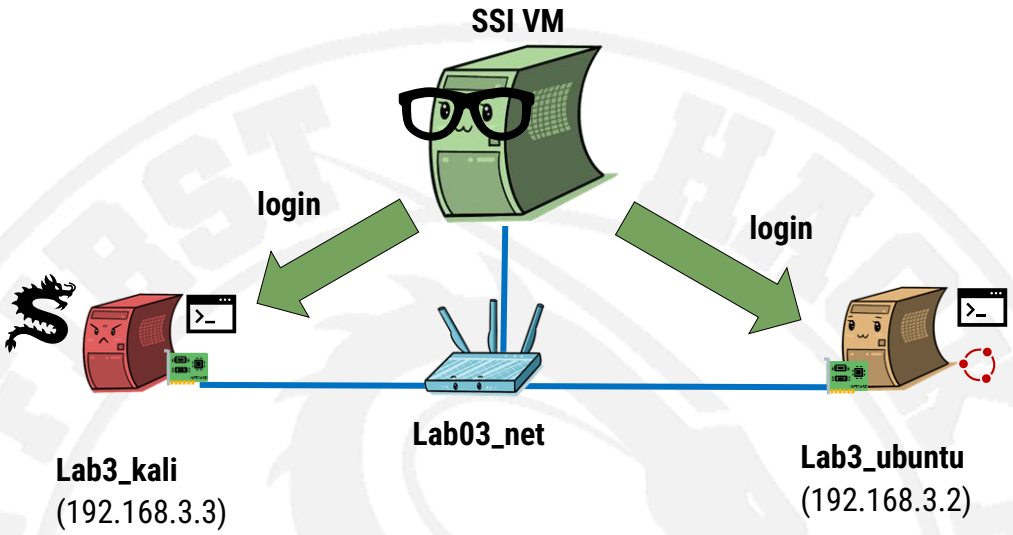
DEPARTMENTO DE INFORMÁTICA. UNIVERSIDAD DE OVIEDO

Seguridad de Sistemas Informáticos | 2023 – 2024 (V4.0 *"S-81 Isaac Peral"*)

Contenido

Infraestructura de este laboratorio

🔍 ***Este laboratorio utiliza exactamente la misma infraestructura que el anterior. Este dibujo solo está aquí para recordarte cómo era***



Bloque 1: Cifrado asimétrico

Ejercicio L4B1_KEYPAIR: Generar un par de claves (pública – privada)

Descripción de la actividad / Aplicación práctica

Se trata de **crear las condiciones necesarias** para permitir que dos usuarios puedan transferirse información en el futuro usando cifrado asimétrico.

Resultados Esperados

Puedes **crear un par de claves distinto para cada usuario** que forme parte de una comunicación secreta que use cifrado asimétrico. Puedes además contestar las preguntas que se hacen durante la ejecución del ejercicio.

Otra información necesaria para su realización

Como hemos visto en la teoría, para que sea posible una comunicación cifrada usando cifrado asimétrico cada usuario participante en la misma debe tener un par de claves generado asociado a su identidad. El algoritmos que vamos a usar para crear dichos pares de claves es **RSA**, también visto en la teoría.

🔍 ***RSA funciona con claves de hasta 4096 bits. Cuanto más larga sea la clave, mejor, ya que mejora la seguridad contra ataques de fuerza bruta. Además, el algoritmo puede cifrar más datos con una clave más larga (aunque el cifrado será más lento).***

Usando la imagen siguiente (https://cheatography.com/albertx/cheat-sheets/openssl/), busca el comando para generar un par de claves RSA de 4096 bits.

ASYMMETRIC ENCRYPTION
List elliptic curves available **openssl ecparam -list_curves**
Create 4096 bits RSA public-private key pair **openssl genrsa -out pub_priv.key 4096**
Display detailed private key information **openssl rsa -text -in pub_priv.key -noout**
Encrypt public-private key pair using AES-256 algorithm **openssl rsa -in pub_priv.key -out encrypted.key -aes256**
Remove keys file encryption and save them to another file **openssl rsa -in encrypted.key -out cleartext.key**
Copy the public key of the public-private key pair file to another file **openssl rsa -in pub_priv.key -pubout -out pubkey.key**
Encrypt a file using RSA public key **openssl** pkeyutl **-encrypt -inkey pubkey.key -pubin -in cleartext.file -out ciphertext.file**
Decrypt a file using RSA private key **openssl** pkeyutl **-decrypt -inkey pub_priv.key -in ciphertext.file -out decrypted.file**
Create private key using the P-224 elliptic curve **openssl ecparam -name secp224k1 -genkey -out ecpriv.key**
Encrypt private key using 3DES algorithm **openssl ec -in ecP384priv.key -des3 -out ecP384priv_enc.key**

Cuando lo encuentres, **genera un par de claves** para el usuario del contenedor *Ubuntu* y otro para el usuario del contenedor *Kali*, llamando a los ficheros generados de distinta forma para evitar problemas futuros. Cuando hayas generado cada par de claves, busca también la opción para **extraer de ellos la clave pública** de cada par, escribiéndola en un fichero independiente que se llame también de una manera distinta para cada usuario. *¿Tienes claro qué fichero de esos es el que le tendrías que enviar a quien quiera cifrar información que solo tú puedas leer? ¿Para qué crees que serviría copiar el fichero con el par de claves completo a otra máquina de tu propiedad?*

En el siguiente paso de este ejercicio, comprueba el contenido de los dos ficheros generados para cada usuario. *¿Has visto el tamaño y la aleatoriedad de las claves generadas? ¿Te parece que son claves que son fácilmente crackeables por* `john` *como vimos en el laboratorio anterior si se usasen para cifrar contenidos?*

Finalmente, solo a efectos de comprobar lo que se dice en teoría, usa los comandos de la imagen anterior para **generar una clave privada que use criptografía de curva elíptica** a partir de la curva **secp256r1** que se usa para el algoritmo *ECDH (Elliptic Curve Diffie-Helman). ¿Qué tamaño tiene? ¿Es más grande o pequeña que las que se usan con Diffie-Hellman estándar, que suelen ser entre 2048 y 3072 bits?*

Ejercicio L4B1_ASSYMCRYPT: Uso del cifrado asimétrico para enviar mensajes confidenciales

Descripción de la actividad / Aplicación práctica

Se trata de **cifrar y descifrar un fichero** usando los conceptos de criptografía asimétrica

Resultados Esperados

Puedes **crear un fichero cifrado** de manera que solo una persona concreta de tu elección (pero de la que tienes la clave necesaria para ello) sea capaz de leerlo. Puedes contestar a estas preguntas:

- *¿Entiendes que para que este cifrado funcione tiene que haber un medio para que todo el mundo pueda acceder a las claves públicas reales de los interlocutores a los que quiere mandar el mensaje?*
- *¿Comprendes por tanto la función tan primordial que hacen los anillos de confianza y las autoridades certificadoras a la hora de expedir/validar certificados que asocian claves públicas con identidades de personas?*
- *En consecuencia, ¿Entiendes por qué sin estas entidades el esquema no funcionaría?*
- *Piensa además lo siguiente: A nivel de programación, ¿Qué es lo más oportuno, que te dé un error que puedes capturar/detectar si el descifrado es incorrecto o que simplemente descifre lo que sea, aunque la clave no sea válida?*

Otra información necesaria para su realización

Para probar el cifrado que hemos visto en la teoría, busca los comandos adecuados de la imagen anterior para hacer estas operaciones:

- Date cuenta de que ahora mismo tenemos **dos contenedores con dos usuarios**, a los que llamaremos U y K, y cada uno de ellos tiene su propio par de claves. En estas condiciones no podemos enviar nada a nadie. Para permitir la comunicación, piensa bien lo que hemos dicho en la teoría y **hazle llegar a U la clave correcta de K que permita a U enviar mensajes que solo K pueda descifrar**. Haz la operación para permitir también la comunicación secreta opuesta, entre K y U.
- **Crea un fichero de texto muy pequeño** el contenedor de U.
- **Cifra el fichero de manera que solo K pueda leerlo.** Comprueba que efectivamente K puede descifrar el fichero. *¿Qué estamos asumiendo para qué afirmemos que sólo K puede hacerlo? ¿Qué tendría que filtrarse o robarse para que esta comunicación secreta dejara de funcionar y un tercero pudiera leer los contenidos?*

Finalmente, trata de descifrar uno de los fichero cifrados con la clave que no es (por ejemplo, intenta descifrarlo con una de las claves públicas y luego con la clave privada del usuario que no es). *¿Qué ocurre en cada caso? ¿Se sabe por tanto a ciencia cierta cuando una operación de descifrado falla y, por tanto, cuando te han dado la clave equivocada?*

Bloque 2: Combinando cifrado simétrico y asimétrico

Ejercicio L4B2_KDFPASS: Usar un KDF como contraseña para un cifrado simétrico

Descripción de la actividad / Aplicación práctica

Consiste en **generar una contraseña verdaderamente fuerte para un cifrado simétrico**, de manera que ya no sea crackeable como en el laboratorio anterior.

Resultados Esperados

Sabes hacer cifrado simétrico fuerte con **una clave muy robusta**, tal y como se haría para usos reales.

Otra información necesaria para su realización

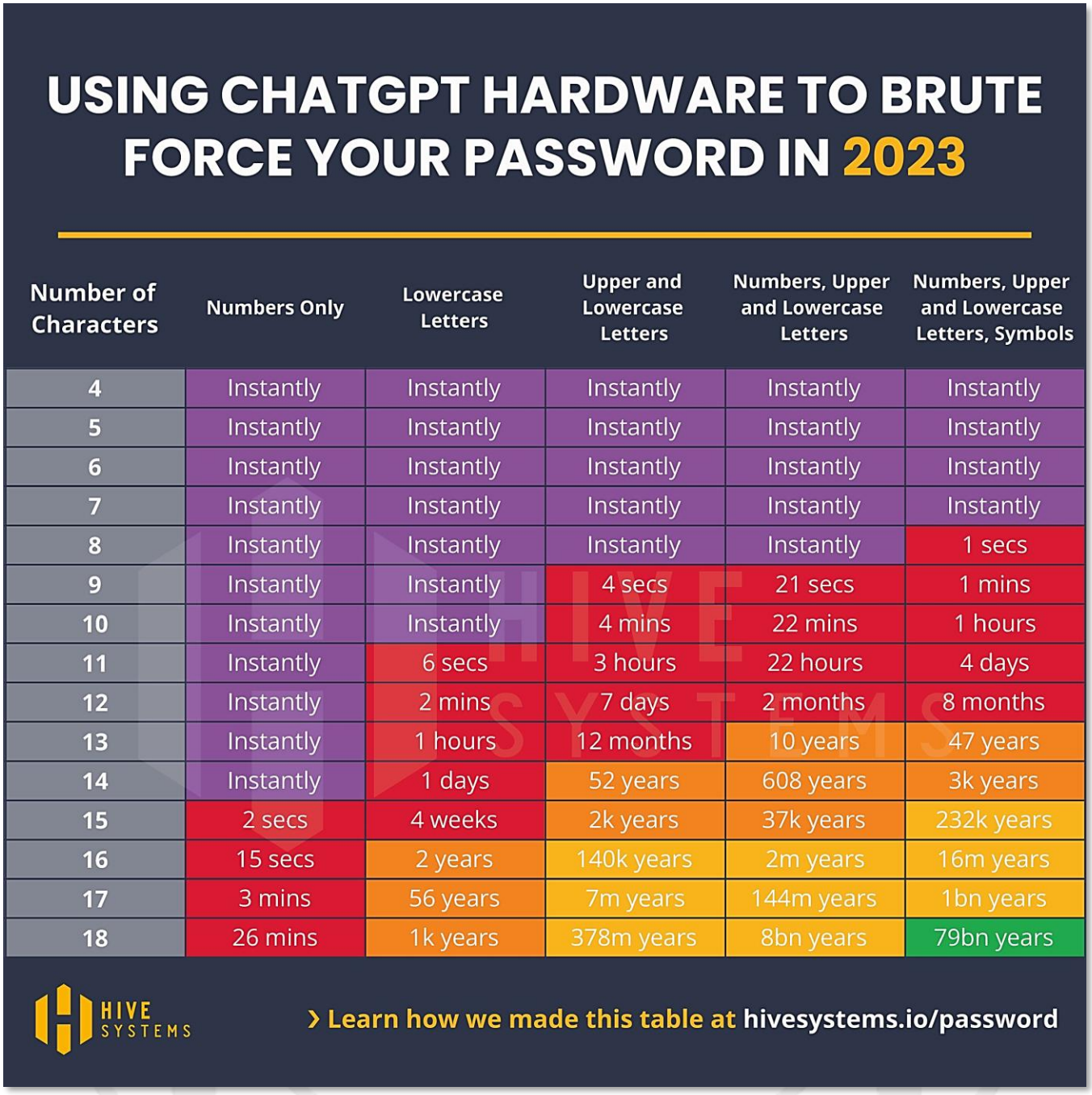
🔍 ***Una clave de cifrado es una cadena aleatoria de bits creada explícitamente para codificar y descifrar datos. En este escenario, cuanto más aleatoria sea dicha cadena mejor para estos fines. Por este motivo, tener un generador de nºs aleatorios robusto y probado es capital en temas de criptografía.***

Como teclear caracteres a mano hasta completar una cadena de 128bits no es una buena forma de generar nada aleatorio (no, no lo es, créeme 😊), podemos usar uno de los conceptos vistos en teoría para generar una cadena realmente aleatoria: un algoritmo de derivación de claves o ***Key Derivation Function*** (KDF). Como se vio en la teoría, este algoritmo procesa la clave inicial que damos en texto plano para crear a través de una serie de operaciones matemáticas y rondas (***rounds***) una hash aleatoria a partir de ella. **Esta hash aleatoria es la que podemos usar como clave de cifrado**, en lugar de usar "password" como antes.

Para generar una clave aleatoria con este procedimiento solo tenemos que hacer esto: `openssl enc -pbkdf2 -aes-128-ecb -k <clave en texto inicial> -P`. Este comando le pide a OpenSSL que cree una clave AES de 128 bits (`-aes-128-ecb`) usando una función KDF de las mencionadas en teoría, **PBKDF2** (`-pbkdf2`) e imprima el resultado en el terminal (`-P`). Para poder manejarla mejor, podemos guardarla en un fichero llamado `csim.priv` o con el nombre que quieras (por favor, **guarda solo el contenido de la clave privada**, no el de la sal ni las cadenas `key=` o `salt=`).

Ahora ya podremos **cifrar y descifrar el archivo de manera simétrica** exactamente como vimos en el **laboratorio 3**, solo que esta vez la clave se pasará en un parámetro `-K` así `-K $(cat csim.priv)`. Prueba ahora a cifrar y descifrar un archivo que crees de ejemplo (o al mismo del laboratorio anterior) con esta nueva clave robusta para comprobar que todo funciona correctamente.

Usa finalmente este dibujo para determinar si tu nueva clave es o no crackeable por fuerza bruta. Asume que las letras pueden ser mayúsculas o minúsculas indistintamente.



USING CHATGPT HARDWARE TO BRUTE FORCE YOUR PASSWORD IN 2023

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	Instantly	Instantly
7	Instantly	Instantly	Instantly	Instantly	Instantly
8	Instantly	Instantly	Instantly	Instantly	1 secs
9	Instantly	Instantly	4 secs	21 secs	1 mins
10	Instantly	Instantly	4 mins	22 mins	1 hours
11	Instantly	6 secs	3 hours	22 hours	4 days
12	Instantly	2 mins	7 days	2 months	8 months
13	Instantly	1 hours	12 months	10 years	47 years
14	Instantly	1 days	52 years	608 years	3k years
15	2 secs	4 weeks	2k years	37k years	232k years
16	15 secs	2 years	140k years	2m years	16m years
17	3 mins	56 years	7m years	144m years	1bn years
18	26 mins	1k years	378m years	8bn years	79bn years

HIVE SYSTEMS

› Learn how we made this table at hivesystems.io/password

Ejercicio L4B2_REALASSYMCRIPT: Entender cómo funciona en la vida real el cifrado asimétrico

Descripción de la actividad / Aplicación práctica

Consiste en **reproducir el proceso que usa SSL y otros servicios de transmisión de información cifrada** similares por Internet para combinar el uso de cifrado simétrico y asimétrico para cifrar gantes cantidades de datos, aprovechándose de lo mejor de cada tipo de cifrado.

Resultados Esperados

Puedes **enviar un fichero grande cifrado a alguien** y la información necesaria para que lo descifre **combinando cifrados simétricos y asimétricos** como se hace en la vida real, entendiendo cómo es el proceso y qué se cifra con cada cosa. Puedes contestar a estas preguntas:

- *¿Entiendes que ahora puedes enviar el fichero cifrado por un medio y la clave para descifrarlo (que también estará cifrada, y sólo la persona autorizada podrá descifrarla) por otro, para maximizar la seguridad de la transmisión?*
- *¿Te das cuenta de que este procedimiento híbrido garantiza la confidencialidad y la autenticación de los interlocutores al mismo tiempo, de manera que solo las personas adecuadas puedan conocer los secretos?*
- *¿Crees que todo esto es muy complejo para un uso continuo por ejemplo en el envío de contenidos cifrados por email? Si te lo parece, puedes usar para tu día un servicio de correo seguro como **ProtonMail** (https://proton.me/es-es/mail), que integra todos estos conceptos, es gratuito y te da todas las ventajas que se ven en este ejercicio muy fácilmente.*

Otra información necesaria para su realización

Se da la circunstancia de que, para que RSA trabaje con datos grandes, **habría que partir el fichero con los datos en bloques de 512 bytes** (si los datos son más pequeños de 512 bytes se rellena automáticamente el bloque hasta alcanzar este tamaño), porque si no no funciona. Pruébalo **intentando cifrar** con una clave pública como en el bloque 1 **un fichero grande**, por ejemplo de más de 1 Mb, que te descargues de Internet. *¿Qué ocurre?*

🔍 ***Aunque nos molestásemos en partir un fichero en bloques de 512 bytes, como dijimos en la teoría RSA es muy lento para cifrar datos grandes***

Es hora de resolver este problema **combinando ambos tipos de cifrado**, para ilustrar lo que hemos visto en teoría acerca de cómo se usan en la vida real. Sigue estos pasos:

- **Cifra el fichero completo con cifrado simétrico** y una clave realmente robusta como vimos en el ejercicio anterior. Si en lugar del modo ECB de AES decides usar el modo CBC como se vio en el **laboratorio 3**, acuérdate de generar un IV con `openssl rand` como vimos entonces, para pasárselo en un parámetro adicional `-iv`.
- **Cifra la clave que has usado para hacer este cifrado simétrico** (Ej.: `csim.priv` del ejercicio anterior) con la clave adecuada del par de claves del destinatario para que solo dicho destinatario pueda ver esta clave de descifrado (ya sabes cuál 😉).
- **Hazle llegar al destinatario** tanto el fichero con los datos, cifrado con criptografía simétrica, como el fichero con la clave que necesitas para descifrarlo que acabas de cifrar asimétricamente.
- Haz lo necesario para obtener la clave de descifrado del fichero y, con ella, el fichero original. *¿Entiendes ahora cómo funcionan estos intercambios que usan ambos métodos?*

NOTA: Este ejercicio realmente puedes usarlo como prueba de que has entendido cómo se usan estos tipos de cifrados, porque te obliga a trabajar con ambos 😉.

Bloque 3: Integridad y firmas digitales

Ejercicio L4B3_AE: Uso de cifrado autenticado (*Authenticated Encryption o AE*)

Descripción de la actividad / Aplicación práctica

Consiste en que entiendas **cómo usar el cifrado autenticado** y sus características especiales para comprender mejor los conceptos de teoría.

Resultados Esperados

Puedes usar el cifrado autenticado para cifrar contenidos y **entender las ventajas que aporta** usando *CyberChef*. Puedes contestar a estas preguntas y a las que se plantean en el texto del ejercicio:

- *¿Qué ocurre si cambio un carácter de la clave de cifrado o del tag?*
- *¿Qué ocurre si cambio un carácter de la entrada cifrada al intentar descifrarlo?*

Otra información necesaria para su realización

Para practicar con *Authenticathed Encryption* una de las cosas que debemos hacer es **seleccionar un algoritmo de cifrado simétrico con esta característic**a. Uno de ellos es **AES-256-GCM** (*Galois Counter Message*, una de las tres formas vistas en la teoría para crear un MAC). Intenta cifrar un fichero cualquiera con este algoritmo con `openssl` con este comando: `openssl enc -aes-256-gcm -nosalt -p -in fichero.txt -out file.out` *¿Funciona?*

Lo que ocurre es que `openssl` ha decidido por el momento no implementar cifrados AE o AEAD por una serie de razones. El principal motivo es que la salida no se puede validar completamente hasta que se tenga toda la entrada, lo que excluye su uso con tráfico que se va recibiendo por streaming, como es el caso típico cuando se usa SSL.

En otras palabras, solo es posible saber si el tráfico es correcto cuando se tiene todo el tráfico que se va a emitir al completo. Si el tráfico va llegando paulatinamente, el receptor procesa lo que va llegando (y actúa en consecuencia) y al final de la transmisión se descubre que estaba corrupto, todo el trabajo que ha hecho el receptor con los datos que iban llegando tendrá que deshacerlo, lo cual no solo es problemático de implementar, sino que además puede suponer un problema de seguridad muy grave en este escenario.

¡Pero eso no quiere decir que no podamos probarlo! ***CyberChef***, que ya aprendimos a usar en el **Laboratorio 3** tiene un modo para cifrar con **AES-GCM** que si podemos usar, ya que la entrada que admite es **finita y completamente conocida** de antemano (o la escribimos nosotros o cargamos un fichero con ella). Para probarlo abre la herramienta y sigue estas instrucciones:

- El algoritmo **necesita una clave**. Usa una clave robusta **creada con un KDF** como vimos en la sección anterior. Cuando generes una, **guárdalo para después**.

- También **necesita un IV**. Puedes generar uno aleatorio como vimos en la **Laboratorio 3** cuando hablamos de como usar el modo CBC de AES. Si generas uno de 16 bits la herramienta seleccionará automáticamente **AES-128-GCM** por ti, que sirve para este ejercicio perfectamente. Cuando generes uno, **guárdalo para después**.
- No hace falta que añadas ***Additional Authenticated Data*** (para este ejercicio da igual usar AE que AEAD).
- El ***Input*** se recomienda introducirlo ***Raw*** y puede ser una simple cadena de texto. El ***Output*** puedes seleccionarlo en formato ***hexadecimal***.
- **Cifra el contenido** y copia el resultado, tanto la salida como el *Tag*.
- Ahora, usando la clave y el IV que has usado para el cifrado, **descifra con *CyberChef*** lo que te ha salido (¡el *Tag* no!) para comprobar que funciona. No te olvides que la salida también te dio una cadena de caracteres llamada Tag, que también debes copiar e incluir como parámetro a la hora de descifrar, ya que es **lo que posibilita la autenticación de los datos cifrados**..
- **Altera cualquiera de los datos** (la entrada, la clave, el IV, el *Tag*...). *¿Se detecta el problema de cifrado?*
- Cifra ahora con *CyberChef* cualquier dato **en modo AES-CBC** (cifrado sin autenticar, el que vimos en el **Laboratorio 3**) usando la clave e IV que quieras (puedes reutilizar las que ya tienes) y apunta el resultado. Trata de descifrarlo después con esos mismos datos y comprueba que funciona. Ahora, **reemplaza algunos caracteres del IV por** otros y mira lo que ocurre. *¿Se detecta el problema? ¿Entiendes por tanto por qué es necesario el cifrado autenticado y la diferencia entre este y el no autenticado?*

Ejercicio L4B3_DSIG: Firmas digitales para integridad

Descripción de la actividad / Aplicación práctica

Consiste en entender la utilidad de las firmas de ficheros, así como el procedimiento para generarlas y validarlas correctamente.

Resultados Esperados

Puedes generar la firma digital de un fichero y verificarla correctamente. Puedes contestar a estas preguntas:

- *¿Crees que tiene sentido enviar el fichero y la firma por separado a un destinatario en algún caso concreto?*
- *¿Qué ocurre si cambio un carácter del fichero recibido correctamente tras validarlo y lo intento validar de nuevo?*

Otra información necesaria para su realización

El objetivo de este ejercicio es **practicar con el uso de firmas digitales** para asegurar la autenticidad de un mensaje, en el sentido de que no se ha modificado desde que se emitió.

🔍 ***Para ello, conviene recordar que una firma digital de un mensaje no es más que cifrar la hash de este con la clave privada del emisor, de manera que, cuando llegue a destino, se puede usar la clave pública del***

emisor complementaria a la misma para descifrar la hash recibida, y compararla con la del mensaje que se ha recibido calculada en destino.

Cualquier discrepancia en la comprobación de la firma indica que el mensaje ha sido alterado o corrompido de alguna forma. `openssl` permite hacer los procesos de firma y verificación descritos muy fácilmente.

DIGITAL SIGNATURES
Generate DSA parameters for the private key. 2048 bits length **openssl dsaparam -out dsaparam.pem 2048**
Generate DSA public-private key for signing documents and protect it using AES128 algorithm **openssl gendsa -out dsaprivatekey.pem -aes-128-cbc dsaparam.pem**
Copy the public key of the DSA public-private key file to another file **openssl dsa -in dsaprivatekey.pem -pubout -out dsapublickey.pem**
To print out the contents of a DSA key pair file **openssl dsa -in dsaprivatekey.pem -text -noout**
Signing the sha-256 hash of a file using RSA private key **openssl dgst -sha256 -sign rsakey.key -out signature.data document.pdf**
Verify a SHA-256 file signature using a public key **openssl dgst -sha256 -verify publickey.pem -signature signature.data original.file**
Signing the sha3-512 hash of a file using DSA private key **openssl pkeyutl -sign -pkeyopt digest:sha3-512 -in document.docx -inkey dsaprivatekey.pem -out signature.data**
Verify DSA signature **openssl pkeyutl -verify -sigfile dsasignature.data -inkey dsakey.pem -in document.docx**
Create a private key using P-384 Elliptic Curve **openssl ecparam -name secp384r1 -genkey -out ecP384priv.key**
Encrypt private key using 3DES algorithm **openssl ec -in ecP384priv.key -des3 -out ecP384priv_enc.key**
Sign a PDF file using Elliptic Curves with the generated key **openssl pkeyutl -sign -inkey ecP384priv_enc.key -pkeyopt digest:sha3-512 -in document.pdf -out signature.data**
Verify the file's signature. If it's ok you must receive "Signature Verified Successfully" **openssl pkeyutl -verify -in document.pdf -sigfile signature.data -inkey ecP384priv_enc.key**

Por tanto, para practicar el uso de firmas necesitamos un par de claves pública y privada de un emisor (nos sirven las que generamos en el bloque 1) y usar uno de los comandos de la imagen anterior (https://cheatography.com/albertx/cheat-sheets/openssl/) para firmar un fichero cualquiera y escribir la firma a un fichero independiente. Por favor comprueba la teoría para saber qué algoritmo puede cons8iderarse seguro para generar firmas.

- Hecho esto, envía al receptor del mensaje **tanto el mensaje en sí como su firma**.
- El receptor ahora **debería usar la clave pública del emisor** (que también debería estar en su poder, y sino puedes copiársela) para verificar que el mensaje que le ha llegado no se ha alterado desde su emisión.
- Comprueba finalmente qué pasa si ahora el mensaje recibido **lo modificas** de cualquier forma y tratas de validarlo de nuevo.

Aquí es donde aparece el principal problema de este esquema de cifrado. Cuando alguien te envía un mensaje firmado y una clave pública, *¿Por qué deberías confiar en esta clave pública? ¿Qué sucede si la clave pública y el mensaje firmado* ***fueron alterados o reemplazados durante la transmisión****?* Como vimos en la teoría, esto se resuelve con **certificados y una infraestructura de clave pública** (PKI) (o un anillo de confianza) que le den validez a la clave de un usuario X. De no tenerlos, dependemos de la buena fe de los emisores para confiar en que son quienes dicen. Y la experiencia dice que la buena fe y la seguridad no son buenas amigas ☹️. El siguiente ejercicio tratará de demostrártelo.

🛠 Ejercicio L4B3_CERTSIGNPDF: Firmar un PDF con un certificado autogenerado

🥷 Descripción de la actividad / Aplicación práctica

Consiste en que entiendas el proceso de **generación de un certificado que represente a alguien** y cómo puede usarse para **firmar digitalmente contenidos** con el mismo. No obstante, en el proceso se trata de que entiendas también como sin una PKI esta firma tiene muy poco o nulo valor.

🏅 Resultados Esperados

Puedes **entender el proceso completo de generación de certificados** y contestar a las siguientes preguntas, además de las que se planteen en el ejercicio:

- *¿Entiendes bien las diferencias entre un certificado CRT y otro PFX y lo que contiene cada uno?*
- *¿Comprendes porque no puedes firmar con un certificado CRT?*
- *¿Eres ahora consciente del problema que tiene no contar con una autoridad certificadora cuando quieres firmar contenidos con un certificado?*
- *¿Sabías que tenías a tu alcance la forma de firmar documentos de manera legal y válida gracias a tu DNIe?*

📱 Otra información necesaria para su realización

Con `openssl` puedes generar directamente certificados digitales **tipo X.509** como los vistos en la teoría, que llevarán extensión `.crt`.

🔍 ***Un certificado con esta extensión contiene la clave pública de una persona o entidad, cuyos datos (nombre, empresa, correo, dominio...) se especifican en los distintos campos de la estructura del formato X.509 que contiene el fichero del certificado. Los*** `.crt` ***se usan por ejemplo para establecer conexiones seguras desde el servidor web a un navegador cuando se usa el protocolo HTTPS (HyperText Transfer Protocol Secure), ya que para eso solo hace falta una clave pública.***

La primera parte de este ejercicio consiste en que uses el contenido de esta imagen (https://cheatography.com/albertx/cheat-sheets/openssl/) **para generar un certificado CRT** a nombre de quien quieras, inventándote los datos (puede ser famoso, un compañero, un profesor, una cuenta B de un *streamer*...da igual 😊).

DIGITAL CERTIFICATES
Generating a CSR file and a 4096 bits RSA key pair **openssl req -newkey rsa:4096 -keyout private.key -out request.csr**
Display Certificate Signing Request (CSR) content **openssl req -text -noout -in request.csr**
Display the public key contained in the CSR file **openssl req -pubkey -noout -in request.csr**
Creating a Certificate Signing Request (CSR) using an existing private key. *This can be useful when you need to renew the public digital certificate without changing the private key.* **openssl req -new -key private.key -out request.csr**
Create EC P384 curve parameters file to generate a CSR using Elliptic Curves in the next step. **openssl genpkey -genparam -algorithm EC -out EC_params.pem -pkeyopt ec_paramgen_curve:secp384r1 -pkeyopt ec_param_enc:named_curve**
Create a CSR file using Elliptic Curve P384 parameters file created in the previous step. *Instead of using RSA keys.* **openssl req -newkey ec:EC_params.pem -keyout EC_P384_priv.key -out EC_request.csr**
Create a self-signed certificate, a new 2048 bits RSA key pair with one year of validity **openssl req -newkey rsa:2048 -nodes -keyout priv.key -x509 -days 365 -out cert.crt**
Create and sign a new certificate using the CSR file and the private key for signing (you must have a openssl.cnf file prepared) **openssl ca -in request.csr -out certificate.crt -config ./CA/config/openssl.cnf**
Display PEM format certificate information **openssl x509 -text -noout -in cert.crt**
Display certificate information in Abstract Sintax Notation One (ASN.1) **openssl asn1parse -in cert.crt**
Extract the certificate's public key **openssl x509 -pubkey -noout -in cert.crt**
Extract the public key's modulus in the certificate **openssl x509 -modulus -noout -in cert.crt**
Extract the domain certificate from an HTTPS/TLS connection **openssl s_client -connect domain.com:443 | openssl x509 -out certificate.crt**
Convert a certificate from PEM to DER format **openssl x509 -inform PEM -outform DER -in cert.crt -out cert.der**
Checking whether the certificate pubic key matches a private key and request file. One step per file. Must match in the output hashes. **openssl x509 -modulus -in certificate.crt -noout | openssl dgst -sha256** **openssl rsa -modulus -in private.key -noout | openssl dgst -sha256** **openssl req -modulus -in request.csr -noout | openssl dgst -sha256**

Una vez que has generado el certificado `.crt`, puedes exportarlo a tu host y **ver sus propiedades** por ejemplo en *Windows*. *¿Qué dice el SO sobre el certificado? ¿Es de confianza?*

🔍 ***No obstante, nos estamos olvidando de un detalle importante, y es que queremos un certificado para firmar documentos y con un CRT no podemos. La razón es sencilla: como dijimos antes, este tipo de certificado no tiene la clave privada de su dueño, por lo que no nos sirve para firmar. ¿Recuerdas por qué?.***

Para firmar **necesitamos otro tipo de certificado,** también de tipo X.509: el de extensión `.pfx`. Un certificado ***PFX*** (*Personal inFormation eXchange*) **es un certificado protegido por contraseña**. Y la razón es sencilla de entender: **contiene tanto la clave pública como la privada** de la persona que representa. Además contiene

otras cosas, como el certificado de la autoridad certificadora que lo valida, pero eso no importa ahora. Lo que sí es importante es que entiendas que un `.pfx` es algo así como una copia de seguridad de un certificado de una persona que además tiene su clave privada, y que es la que necesitamos para firmar documentos.

¿Cómo obtener un `.pfx` *a partir de un* `.crt`*?* Si te das cuenta, cuando generaste el certificado `.crt` (si lo has hecho bien 😊) la clave privada generada **se guardó en un fichero aparte del propio certificado CRT** (que solo tiene la pública correspondiente, recuerda), para que puedas guardar ambos por separado con distintos permisos *¿Recuerdas por qué esto es necesario?*.

Puedes usar el `.crt` y el fichero con la clave privada para unirlos formando un `.pfx` con este comando:

```
openssl pkcs12 -export -out <nombre para el certificado PFX>.pfx -inkey <el fichero de la clave privada que generaste al crear el CRT> -in <certificado CRT generado antes>
```

Al crear el certificado PFX **se te pedirá una clave para protegerlo**. Puedes dar una cualquiera, en esta prueba solo importa que la recuerdes (¡en la vida real más te vale que sea fuerte!)

Firmando "de mentira": Firmar un PDF con un certificado autofirmado

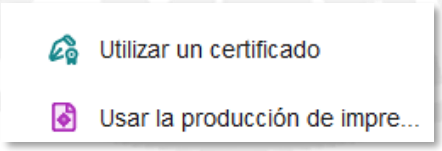
Este certificado PFX que acabas de generar te imaginarás que **no tiene validez** alguna ya que ninguna CA lo avala (¡lo acabas de crear tú mismo!), así que en principio no podría usarse para firmar por si solo.

🔍 ***Vamos a firmar PDFs con Acrobat Reader, que aunque sea el lector más popular de PDFs, no está disponible para Linux, solo para Windows y MAC. El ejemplo lo vamos a hacer con Windows.***

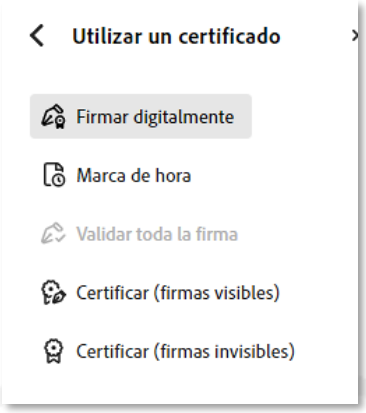
Pero pasa una cosa, y es que ***Windows* da validez a cualquier certificado que importes a su almacén de certificados local**. Dicho de otra forma, cualquier certificado que importes en un *Windows* será perfectamente válido para ese *Windows*, aunque no haya una CA que lo firme. *¿Entiendes por qué eso puede ser un problema de seguridad?*

Por tanto, para este ejercicio **haz doble clic en el certificado**, sigue el tutorial de importación (acuérdate de introducir la clave que pusiste al crear el PFX) **deja todas las opciones por defecto**, y comprueba que todo es correcto. Ahora ya puedes abrir un PDF (vale este boletín de ejercicios 😊) para firmarlo con él.

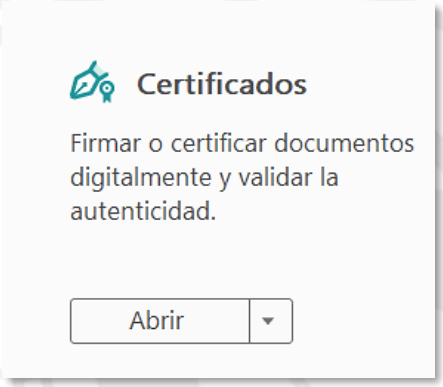
Adobe Reader no tiene la herramienta para firmar con certificados de manera muy accesible. En función de la versión que tengas cambia dónde está. Para la más moderna, tendrás una barra de herramientas a la izquierda donde tienes que ir al final y pulsar en "***Ver más***" hasta que salga esta opción.



Dentro de la cual ya podrás firmar digitalmente con el certificado:



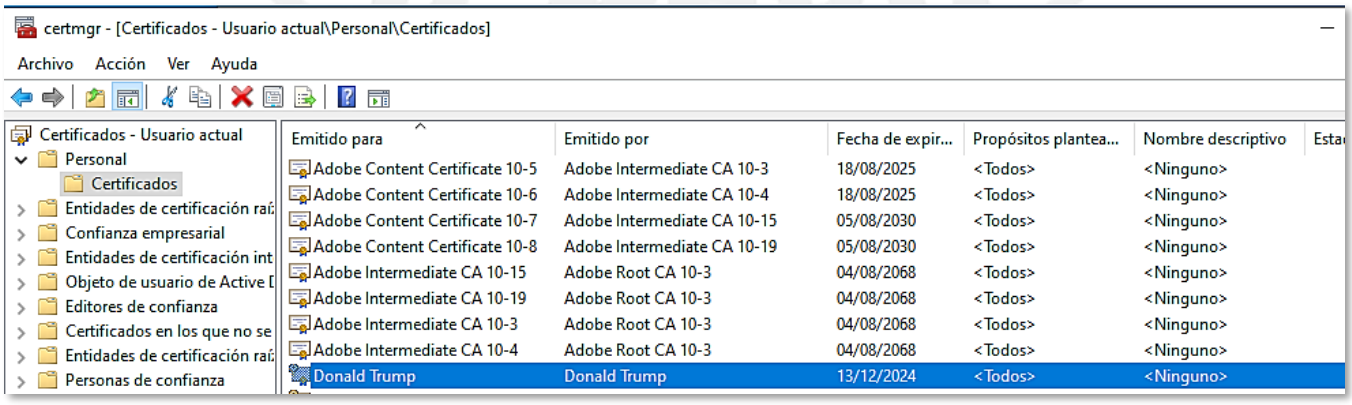
En la versión más antigua tienes que ir al menú *Herramientas* que aparece a la izquierda de las pestañas de todos los PDFs abiertos y localizar esta opción:



🔍 ***En cualquiera de los casos tienes que pintar un rectángulo con el ratón en la parte del documento donde quieras la firma.***

Si lo has hecho bien, saldrá el certificado que hemos creado a nombre de una persona cualquiera como opción para firmar. Hazlo (opcionalmente puedes **bloquear el archivo**, lo que impide a cualquiera modificarlo una vez lo firmes. Es muy útil en la vida real pero para este ejercicio no es relevante) . *¿Es válida esta firma? ¿Entiendes por qué?*

Borra ahora el certificado importado abriendo el programa "***Administrador de Certificados de Usuario***" en tu *Windows* (búscalo en el menú de inicio por nombre, es más fácil). El certificado (en mi caso generado a nombre de Donald Trump 😊) está en esta carpeta si lo has importado con las opciones por defecto

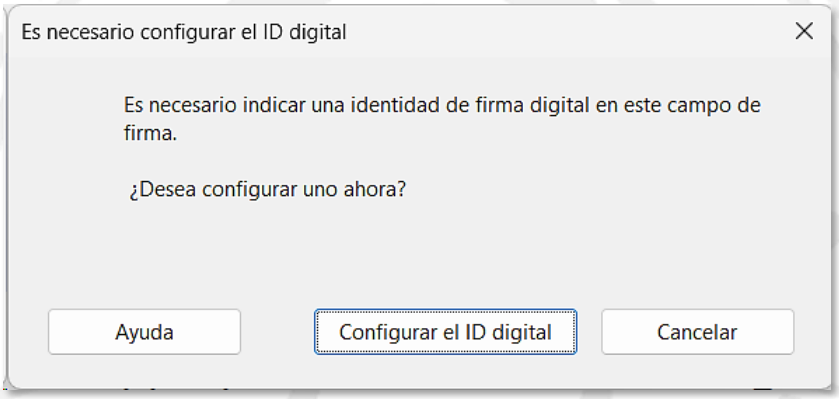


Abre el PDF que acabas de firmar de nuevo. *¿Es válida la firma ahora?* Para asegurarte de que has entendido bien lo que pasa, *¿Qué crees que ocurriría si ahora mueves el documento a otra máquina distinta a la tuya? ¿Sería válida la firma?* ¿Qué tendrías que hacer para que lo fuera? *¿Entiendes ahora por tanto la labor tan importante que hacen las CA que son accesibles vía Internet desde cualquier parte o aquellas que, habiendo sido nombradas de confianza por los fabricantes de los SO, instalan sus certificados en cada una de las instalaciones de estos por todo el mundo para validar los certificados que emiten?*

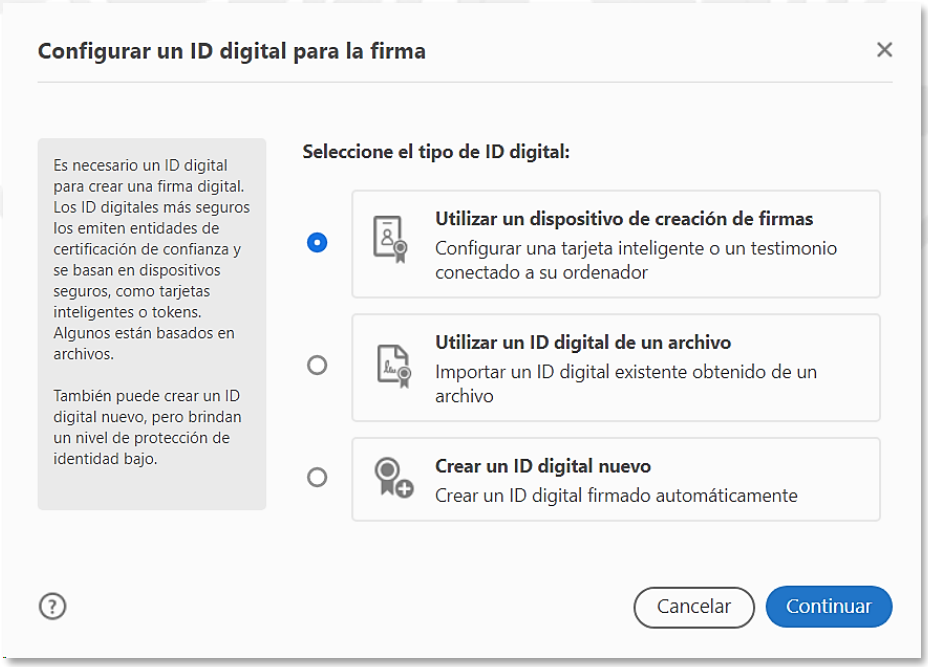
Firmando "de verdad": Certificados gratuitos y válidos que puedes conseguir para tus necesidades

🔍 ***Esta sección no es para que hagas realmente ningún trabajo ahora, sino que es para usar estos conceptos en tu vida diaria.***

¿Puedes firmar documentos o, en general, disponer de certificados válidos sin coste? Sí. Y aquí os vamos a explicar cómo. Cuando intentas firmar un documento en *Adobe Reader* y **no tienes certificados instalados de antes**, verás esta pantalla:



Y entonces tendrás tres opciones de firma:



- **Primera opción**: La forma más sencilla de usarla para firmar es con tu DNI electrónico. Necesitas tener a mano el PIN que te dieron en la policía al sacarlo, que el certificado no esté caducado (caduca cada 2 años, independientemente de cuando caduque el DNI, en cuyo caso tendrás que ir a una máquina de renovación de la comisaría a renovarlo) y comprar un lector de DNIs. Con esto podrás firmar lo que quieras de forma legal, porque **tu identidad la valida el estado**
- **Segunda opción**: Permite usar un certificado válido que le hayas pedido por ejemplo a la **FMNT**: https://www.sede.fnmt.gob.es/certificados/persona-fisica, entidad que se encargará de validarlo. Tienes que hacer el mismo proceso que vimos con el certificado autofirmado para poder usarlo
- **Tercera opción**: Lo mismo que hemos hecho con `openssl` nosotros (certificado autofirmado), pero creado por el *software* propio de *Adobe* y con las mismas restricciones y problemas, por lo que no es útil para firmar documentos legalmente.

🔍 ***¿Quieres soportar HTTPS en un servidor web a tu cargo sin coste? Consulta Let's Encrypt (https://letsencrypt.org/es/) y, si tienes un dominio válido en internet, Certbot (https://certbot.eff.org/) para automatizar su instalación y gestión. Pero esto ya es un tema más vinculado a la asignatura Administración de Servidores y Redes. O quizá te sea útil en tu TFG si contratas un dominio gratuito o muy barato...*** 😊

🗂️ Bloque 4: Los conceptos de Confidencialidad, Integridad y Autenticación (CIA) en tu día a día

🛠️ Ejercicio L4B4_CERTSSH: SSH sin contraseña

🥷 Descripción de la actividad / Aplicación práctica

Necesitas **establecer conexiones seguras automáticas** con una maquina remota, de forma que no se pida ninguna *password* cuando se haga la conexión y ésta siga siendo segura

🏅 Resultados Esperados

Puedes conectarte desde el contenedor *Kali* al de *Ubuntu* usando tu identidad de usuario actual y **no se te pida la contraseña para ello.** Puedes contestar a estas preguntas:

- *¿Por qué crees que una vez transferida la clave pública al sistema de destino ya no hace falta contraseña y el tráfico se cifra igualmente?*
- *¿Qué se están usando para garantizar la autenticación y la confidencialidad?*
- *¿Entiendes por qué los usuarios en el sistema origen y destino no tienen necesariamente que llamarse igual para que funcione?*

📱 Otra información necesaria para su realización

Una de las aplicaciones más comunes de claves públicas y privadas es **habilitar el inicio de sesión directo (sin preguntar por contraseña) a través de SSH en una máquina remota que tiene nuestra clave pública registrada y asociada a una identidad de usuario**.

🔍 ***De esta manera, cada vez que intentemos iniciar sesión en esta máquina, nuestro par de claves pública-privada se usarán para identificarnos como este usuario de forma remota, y no se nos pedirá una contraseña***

Para hacer esto, es necesario seguir este procedimiento: https://linuxize.com/post/how-to-set-up-ssh-keys-on-ubuntu-1804/. Ahí se habla de usar `ssh-keygen` para generar un par de claves pública / privada RSA para el usuario que quiere hacer *login*, al que llamaremos **USERKALI**, aunque se pueden usar las RSA que hemos generado en un ejercicio anterior. A la hora de generar las claves, no pongas ninguna contraseña para leer en el par de claves generado cuando el proceso te lo pida.

Una vez creadas esas claves **transferiremos la clave pública generada al sistema de destino** validándonos con una identidad de usuario válida en dicho sistema, al que llamaremos **USERUBUNTU**. Si lo hacemos, cada vez que **USERKALI** se conecte al contenedor *Ubuntu* vía SSH especificando que quiere conectarse como **USERUBUNTU** el inicio de sesión ocurra automáticamente y no se pedirá la contraseña del usuario.

🔍 ***NOTA: No es necesario desactivar la autenticación basada en contraseñas en `ssh` (ambas formas pueden "convivir") aunque ahora podrías hacerlo, obteniendo algunas ventajas de seguridad...*** 😉

🛠 Ejercicio L4B4_CHECKSIGN: Comprobación manual de la firma de un programa descargado

🥷 Descripción de la actividad / Aplicación práctica

Necesitas **comprobar la integridad** de un fichero que has descargado, para asegurarte de que **no se ha corrompido** o alguien lo ha alterado maliciosamente

🎖 Resultados Esperados

Esta actividad finalizará cuando puedas asegurarte de que el programa descargado **no se haya modificado**, usando la firma que te ha proporcionado su autor.

📖 Otra información necesaria para su realización

Las firmas de archivos se utilizan comúnmente para verificar que los programas / documentos / etc. no hayan sido manipulados o dañados cuando se descargan.

🔍 ***Muchos programas incluyen en su web de descarga instrucciones sobre cómo verificar su ejecutable contra una firma que se muestra en el sitio web (o se descarga dentro de un fichero comprimido junto con el ejecutable) para garantizar que el ejecutable no esté dañado / manipulado.***

Para saber cómo hacerlo, sigue este procedimiento (puedes hacerlo **en la propia máquina virtual *Ubuntu*):**

- Vete a la página de descarga de un programa **que ofrezca firmas de verificación de sus archivos descargados**. Por ejemplo: http://archive.ubuntu.com/ubuntu/dists/bionic-updates/main/installer-amd64/current/images/netboot/
- Descarga el `mini.iso` de *Ubuntu* (o el fichero que hayas encontrado que ofrezca validar su firma) y comprueba su firma con la herramienta `sha256sum`.
- Vete a la página donde se listan las firmas de los programas y abre el archivo apropiado con las firmas en el formato que utilizaste: http://archive.ubuntu.com/ubuntu/dists/bionic-updates/main/installer-amd64/current/images/
- Compara tu resultado con el que aparece allí.

🛠 Ejercicio L4B4_WATERMARK: Marcas de agua

🥷 Descripción de la actividad / Aplicación práctica

Necesitas marcar una imagen con algo que la identifique, **para que sea más difícil para alguien mentir y decir que es suya**

Resultados Esperados

Esta actividad finalizará cuando puedas **poner texto como marca de agua** a cualquier imagen que descargues de Internet. Puedes contestar a estas preguntas:

- *¿Dónde pondrías la marca de agua para dificultar al máximo la apropiación indebida de tu trabajo?*
- *¿Usarías esteganografía combinada con esto para el mismo fin?*

Otra información necesaria para su realización

Aunque **esto no es firmar un archivo,** las marcas de agua de imágenes son un procedimiento para que una imagen con derechos de autor no se pueda robar tan fácilmente por un tercero que reclame su autoría.

🔍 ***Por supuesto, la imagen aún podría ser manipulada, pero eliminar una marca de agua es un proceso que impediría que algunos usuarios sin conocimientos tecnológicos las roben.***

Una marca de agua también podría usarse para crear copias físicas de documentos con marca de agua y que no puedan distribuirse sin ella, ya que eliminarlas de las copias físicas es mucho más difícil. Podemos marcar una imagen usando el paquete *Image Magick* (`sudo apt install imagemagick` y usando su herramienta `convert`). Dos ejemplos que pintan texto rojo girado (aunque puedes cambiar fácilmente los parámetros para probar otras variantes) son:

- **Texto corto, marca de agua grande**: `convert -density 150 -fill "rgba(255,0,0,0.25)" -gravity Center -pointsize 80 -draw "rotate -45 text 0,0 <text>" <original image> <watermarked image>`
- **Marca de agua de dos líneas:** `convert -density 150 -fill "rgba(255,0,0,0.50)" -pointsize 15 -draw "rotate -15 text 0,200 '<line of text 1>'" -draw "rotate -15 text -25,260 '<line of text 2>'" <original image> <watermarked image>`

Utiliza estos ejemplos para hacer marcas de agua a cualquier imagen que tengas. **Puedes hacer esto en la máquina virtual de *Ubuntu*.**

Ejercicio L4B4_META: Metadatos

Descripción de la actividad / Aplicación práctica

Necesitas **investigar o eliminar los metadatos** de cualquier fichero

Resultados Esperados

Esta actividad finalizará cuando puedas **leer y eliminar los metadatos no esenciales** de cualquier imagen (o del tipo del fichero que quieras). Puedes contestar a estas preguntas:

- *¿Te das cuenta de que habrá casos en los que quieras identificar perfectamente a la imagen con tu nombre, etc. en los metadatos y casos en los que quieras todo lo contrario? ¿se te ocurren ejemplos de ambos?*
- *¿Qué utilidad crees que tendría introducir como metadato de un fichero un dato cifrado con un algoritmo fuerte como vimos antes?*
- *¿Has pensado qué aplicaciones podría tener introducir metadatos falsos a sabiendas en un fichero?*

Otra información necesaria para su realización

A veces los metadatos pueden **asociar** un archivo (como una imagen) con nuestra identidad de usuario (pueden contener nuestro nombre, nuestra IP, datos de uno de nuestros dispositivos...). Esto hay veces que puede ser deseable y veces que puede ser **muy contraproducente**.

🔍 ***Unas veces necesitarás indicar que un fichero es tuyo y otras romper cualquier asociación con tu identidad...todo depende del propósito, dónde se publique, el contexto...***

Los metadatos son posiblemente la forma más olvidada de **tener datos comprometedores sin saberlo**, por lo que terminaremos este laboratorio enseñándote las siguientes operaciones con el popular programa de manipulación de metadatos de imágenes `exiftool` (`sudo apt install exiftool`). Ten en cuenta que hay **metadatos de archivo esenciales** (información que **debe** estar en cualquier archivo de un tipo como requisito) y **metadatos adicionales, introducidos por un programa** que los crea u otra herramienta. Lógicamente, `exiftool` solo elimina metadatos adicionales.

- **Consultar los metadatos actuales de una imagen**: `exiftool <fichero de imagen>`
- **Eliminar todos los metadatos de una imagen**: `exiftool -all= <fichero de imagen>` (cuidado porque hay un espacio después del `=`)

Utiliza estos ejemplos para consultar y eliminar los metadatos de cualquier imagen que tengas o descargues. Por ejemplo: https://biosferadigital.com/sites/default/files/archivos/AK-interior.jpg. A continuación, usa estos ejemplos para introducir en la imagen un valor para un metadato cualquiera que tú quieras (aunque te lo inventes):

- https://exiftool.org/examples.html
- https://exiftool.org/exiftool_pod.html

NOTA: Los metadatos **NO tienen relación con el concepto de firma criptográfica de información que vimos anteriormente.**

✍️ Insignias y Autoevaluación

NOTA: Tienes una versión de esta tabla de insignias en formato editable disponible en el *Campus Virtual*. Puedes usar este archivo para crear un documento en el formato que desees y tomar notas extendidas de tus actividades para crear un log de lo que has hecho. Recuerda que el material que elabores se puede llevar a los exámenes de laboratorio.

Nivel de Insignia	Desbloqueado cuando	¿Desbloqueado?
	Puedas crear un par de claves para un usuario concreto y comprendas la necesidad de protección que tiene cada una	
	Puedas ver un archivo de claves generado y comprendas su contenido.	
	Sepas cómo generar una clave de cifrado simétrica a partir de un algoritmo KDF	
	Entiendas cómo trabaja el cifrado autenticado y sus particularidades	
	Puedas crear una firma digital para cualquier documento y usarla para comprobar que no se ha alterado	
	Sepas cómo comprobar la firma de un programa descargado con `sha256sum`. También sabes qué algoritmos evitar para asegurarse de que la firma no es fácilmente falsificable.	
	Puedas poner una marca de agua a cualquier imagen y usarla para intentar proteger tu propiedad intelectual	
	Puedas leer los metadatos de cualquier imagen. Entiendes claramente por qué esto es importante de cara a la seguridad	
	Puedas eliminar los metadatos de cualquier imagen y entiendes claramente por qué esto es importante por razones de seguridad	
	Seas capaz de cifrar y descifrar información con cifrado asimétrico	
	Puedas responder a esta pregunta: *¿Crees que puedes enviar tu clave pública por correo electrónico a otra persona? ¿Y publicarla en Internet? ¿Es un problema de seguridad o no?*	
	Entiendas el proceso de generar un certificado y usarlo para firmar un PDF	
	Entiendas cuál es el propósito de copiar una clave privada y volver a usarla en otra máquina. Puedes responder a esta pregunta: *¿Deberías enviar esta clave por correo electrónico? ¿Por qué?*	
	Seas capaz de iniciar sesión como usuario a través de SSH mediante una clave pública, entendiendo el proceso.	
	Comprendas que un certificado autofirmado, aunque firme un documento, no tiene realmente validez legal y por qué	

	Sepas de dónde sacar un certificado válido, legal y gratuito para firmar lo que necesites en la vida real	
	Comprendas cuántas claves necesitas para realizar un proceso de cifrado asimétrico para un usuario determinado.	
	Puedas responder a esta pregunta: *¿Qué harías si debes eliminar los posibles metadatos adicionales de cualquier imagen en una aplicación web que tengas que poner en producción?*	
	Puedas enviar mensajes cifrados a usuarios concretos de los que conoces su clave pública y recibir una respuesta que también solo tu puedas descifrar	
	Puedas enviar mensajes cifrados y firmados (ambas cosas) a usuarios concretos de los que conoces su clave pública.	
	Puedas responder a esta pregunta: *¿Qué debes hacer con tus pares de claves si se vas a otra máquina, o reinstalas la actual, para evitar la pérdida de datos?*	
	Puedas responder a esta cuestión: Una vez que entiendes el proceso de inicio de sesión de forma remota utilizando tu clave pública, considera que el inicio de sesión con contraseña a través de SSH se puede deshabilitar totalmente para cualquier usuario. Si hacemos eso, *¿es más seguro? ¿Evitaremos ciertos ataques? ¿Qué debemos hacer si queremos que otro usuario inicie sesión en estas circunstancias?*	
	Comprendas las limitaciones del cifrado asimétrico y sepas usar una combinación de cifrado simétrico y asimétrico para resolverlas	
	Puedas responder a esta cuestión: En el proceso de crear una autenticación sin contraseña a través de SSH, *¿no podemos simplemente copiar nuestra clave pública generada a cualquier usuario remoto y suplantar a cualquiera de ellos?* Si no, *¿en qué momento hicimos algo que nos impidió hacerlo?*	
	**You can't touch this**: Tu dominio de la criptografía asimétrica te permite utilizar el cifrado asimétrico para enviar datos confidenciales de forma privada a cualquier usuario que desees, asegurando también que los datos que envías han sido inalterados. Además, puedes establecer canales de comunicación confidenciales con máquinas remotas para usuarios determinados. Por último, debido a la comprensión general del proceso de garantizar la CIA de la información, puedes utilizar eficazmente marcas de agua para identificar tus imágenes si es necesario, y determinar si una imagen puede tener información adicional que pueda identificar algo acerca de sus autores.	