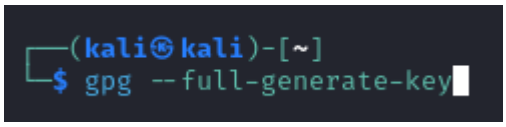
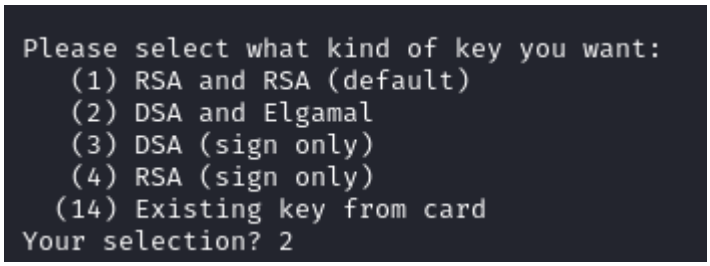
Ejercicio 3

gpg también sirve para el cifrado asimétrico:

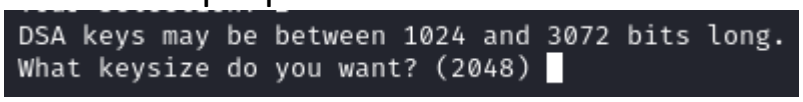
3.1. Lo primero será generar un par de claves, es decir, nuestra propia clave pública y clave privada. Esto lo haremos en primera instancia para el usuario kali. Para ello, añade el parámetro --full-generate-key al comando gpg para generar las claves.



3.2. En el proceso de generación de la clave nos preguntarán varios detalles. El primero es el tipo de clave. La herramienta nos ofrece cuatro opciones. Los nombres se corresponden con el tipo de algoritmo asimétrico asociado (hay varios tipos, como también ocurría en criptografía simétrica: DES, AES, etc.). Es decir, en una clave de tipo DSA se utiliza un algoritmo de cifrado DSA, y en una clave Elgamal, un algoritmo Elgamal. Las dos primeras opciones ofrecen dos algoritmos, que generan dos pares de claves: en total, cuatro claves: dos públicas y dos privadas. El motivo es que generalmente se utiliza un par de claves para cifrar, y otro par diferente para firmar. Nosotros elegimos la opción 2, que tiene algoritmos distintos: de esta forma veremos claramente cuándo se utiliza cada clave.

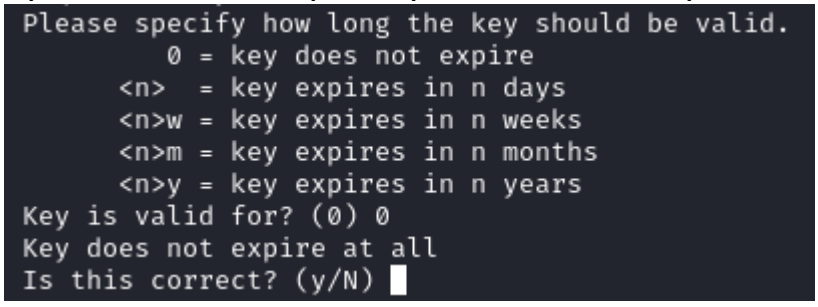


3.3. A continuación nos pregunta el tamaño de la clave del algoritmo DSA. ¿Qué tamaño ofrece por defecto? ¿Qué tamaño tendremos que elegir para que el proceso de creación de la clave tarde menos? ¿Qué inconveniente presentaría esto? Elige el tamaño que permita tardar menos.



Tamaño por defecto DSA: 2048 bits.
Tamaño que permite un proceso más rápido: 1024 bits.
Inconvenientes: Menor seguridad y posible incompatibilidad con ciertos sistemas.

3.4. La siguiente pregunta es el periodo de validez de la clave. ¿Qué sentido puede tener indicar un periodo de validez concreto? Explica las distintas opciones a elegir. Selecciona la opción adecuada para que nunca caduque.



Sentido de indicar un periodo de validez: Mejora la seguridad, facilita el manejo de claves y controla el acceso.
Opciones: No expira, expiración específica (días, meses, años).
Opción adecuada para que nunca caduque: Seleccionar "no expira".

3.5. A continuación nos pide algunos datos para identificar la clave. Generaremos varios pares de claves, por lo que convendría ser cuidadosos a la hora de rellenar estos datos, para que nos permitan identificar de manera unívoca las claves y no den lugar a equívocos.

```
pub   dsa2048 2024-10-24 [SC]
      0F4E82571B1C17422536EBE8A636EAAC3A0F2C30
uid                      joseA (claves para joseA) <josebenitez-19@hotmailm.com>
sub   elg2048 2024-10-24 [E]
```

3.6. El siguiente paso es elegir la clave simétrica que protegerá nuestras claves. Si la olvidamos,no podremos utilizar nuestras claves asimétricas.

Parte anterior

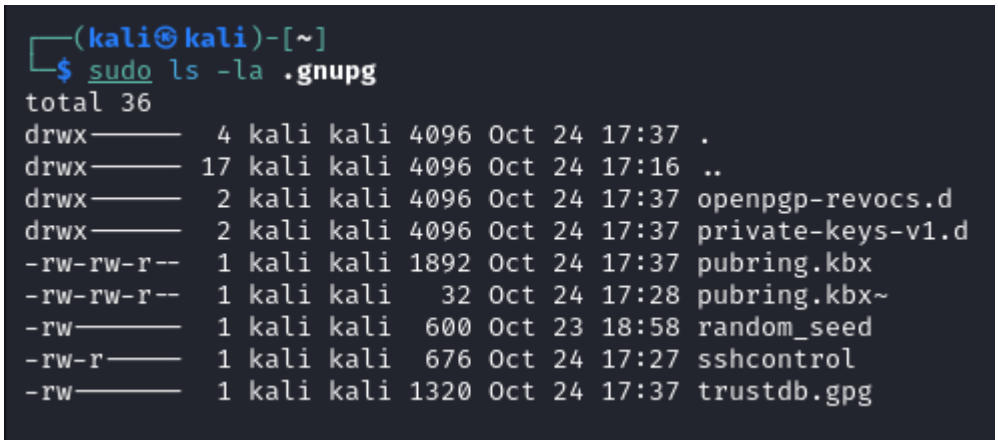
3.7. Ahora la herramienta ejecuta los procedimientos matemáticos para obtener las claves asimétricas que hemos solicitado. Estos procedimientos necesitan muchos datos aleatorios, por lo que nos pide que generemos actividad en el sistema para ayudarle.

Ejercicio 3.5

3.8. Una vez acabe el proceso de generación, si nos fijamos en los mensajes que aparecen en la terminal, veremos una clave primaria (pub) de tipo DSA con tamaño de clave 1024 (1024D). Debajo hay una clave subordinada de tipo Elgamal con tamaño de clave 1024 (1024g).

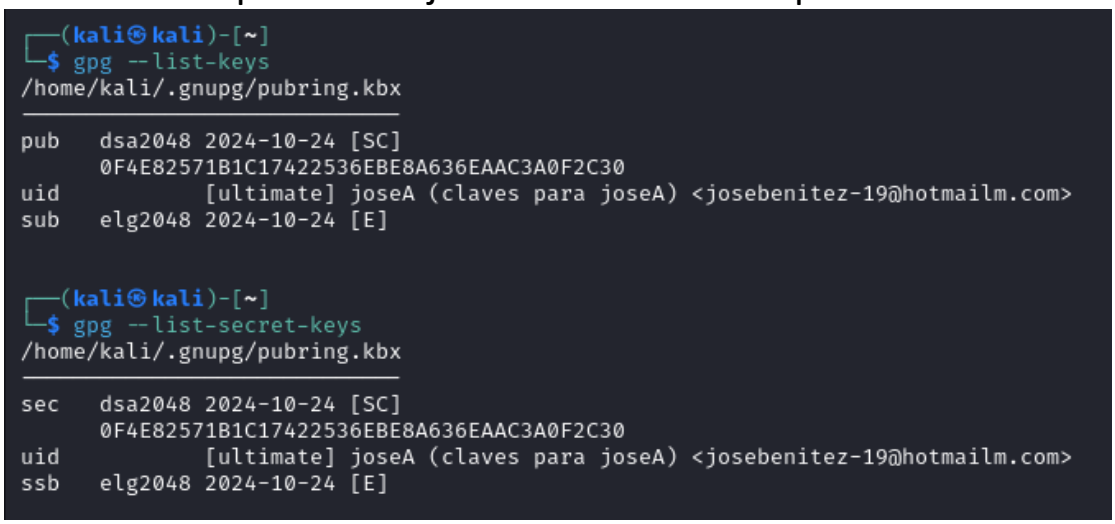
Ejercicio 3.5

3.9. Comprueba la aparición de un directorio oculto llamado .gnupg. ¿Qué contiene este directorio? ¿Qué es el fichero private-keys-v1.d ? ¿Y el fichero pubring.kbx? ¿Puedes ver su contenido?



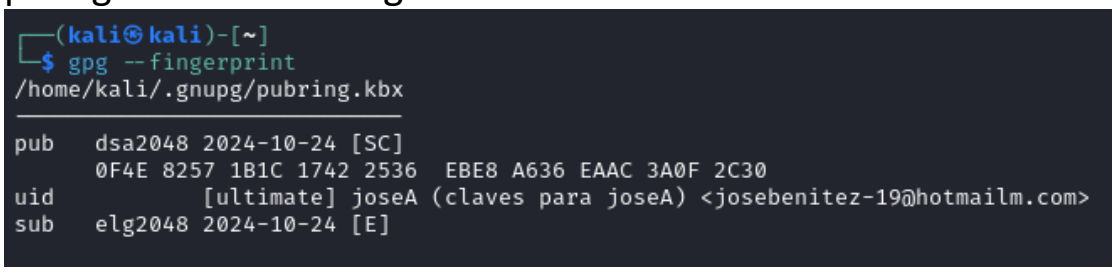
Directorio .gnupg: Almacena claves y configuraciones de GPG.
private-keys-v1.d: Contiene las claves privadas del usuario.
pubring.kbx: Almacena las claves públicas.

3.10. Para trabajar con las claves debemos utilizar la propia herramienta. Averigua qué parámetro utilizar para obtener la lista de claves disponibles, y úsalo para ver las nuestras. Nótese que las claves pueden públicas y privadas, por lo que tendremos 2 parámetros distintos… Fíjate en el identificador de usuario de la clave (UID). ¿Qué relación tiene con la información que introdujimos anteriormente por teclado?



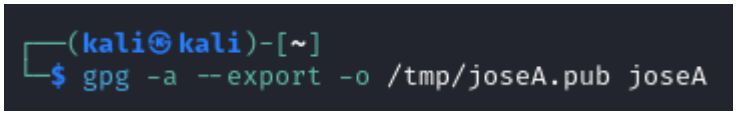
La información que introdujiste al generar la clave (como tu nombre y dirección de correo electrónico) se asocia directamente con el UID de la clave.

3.11. Anota la huella de la clave pública apenas generada. Para ello, te servirá el parámetro fingerprint. Esto podría servirnos para garantizar la integridad de las comunicaciones.



3.12. Ahora tenemos que comunicar nuestra clave pública a quien esté interesado en enviarnos un mensaje cifrado. Esto lo logramos mediante la siguiente sintáxis:
gpg -a --export -o /ruta/de/salida/identificadorClave.pub UID

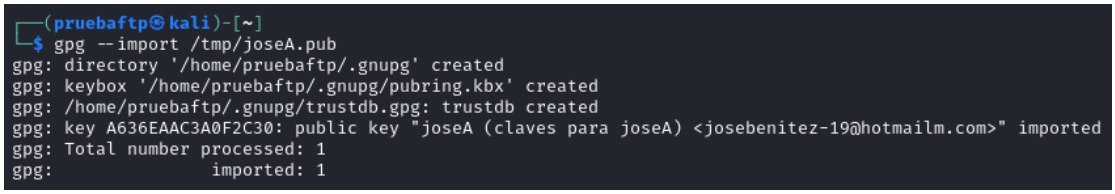
¿Qué pasaría si no utilizáramos los parámetros -a y -o? Coloca la clave en el directorio /tmp para que así esté accesible a otros usuarios del sistema.



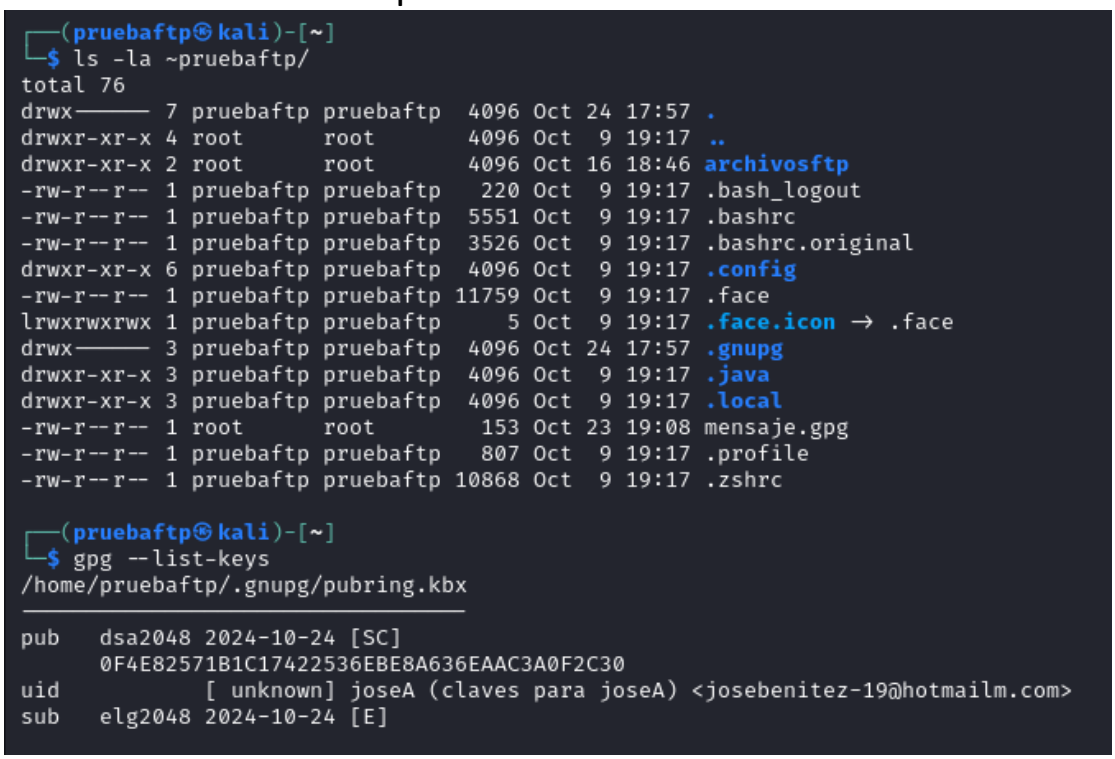
Sin -a: Si omites el parámetro -a, la clave se exportará en un formato binario. Esto puede ser problemático si intentas compartir la clave a través de medios que no manejan datos binarios correctamente, como algunos clientes de correo electrónico, ya que podrían corromper la clave.

Sin -o: Si no usas el parámetro -o, GPG mostrará la clave en la salida estándar (terminal). Esto puede ser útil si deseas copiar y pegar la clave, pero no es práctico si deseas guardarla en un archivo para compartirla posteriormente.

3.13. La idea en este paso es que distintos usuarios del sistema puedan comunicarse entre sí utilizando criptografía asimétrica. Para ello, tendrán que tener acceso a las claves públicas del resto de usuarios para así poder cifrar sus mensajes con dichas claves públicas. Posteriormente, el destinatario tendrá que descifrar el mensaje utilizando su clave privada.
Por lo que cambiaremos a otro usuario del sistema (por ejemplo, usuarioftp), e importaremos la clave pública anteriormente exportada, mediante el parámetro --import:
gpg -–import ruta



3.14. Lo que podemos hacer en este punto, es comprobar la creación del directorio oculto .gnupg para este otro usuario; y comprobar el listado de claves disponibles: tanto las públicas como las privadas. Si nos equivocamos en algún paso del proceso, quizá sea conveniente también aprender cómo eliminar las claves públicas y privadas: de nuevo, habrá 2 parámetros distintos; y además no podremos eliminar una clave pública sin antes haber eliminado la clave privada asociada a dicha clave pública.



3.15. Con usuarioftp, crearemos un fichero llamado mensaje y lo cifraremos con la clave pública de kali para que éste pueda descifrarlo. De nuevo, el mensaje lo podemos crear a partir de la herramienta fortune. El comando para cifrar sería:

gpg –v –a –o /tmp/mensaje.cifrado --encrypt --recipient UID mensaje

Los parámetros -a y -o ya los conocemos. Dejamos el fichero en /tmp porque no nos importa que otros usuarios puedan leerlo: sin la clave privada, el contenido es ininteligible. Hemos añadido el parámetro -v (verbose) para obtener más

información. El parámetro encrypt indica que deseamos cifrado asimétrico y el parámetro recipient va seguido del UID de la clave pública que queremos utilizar. Como ya sabemos, el cifrado utiliza la clave pública del receptor.
¿Qué información aparece al ejecutar este comando?

```
┌──(pruebaftp㉿kali)-[/tmp]
└─$ gpg -v -a -o /tmp/mensaje --encrypt --recipient joseA mensaje
gpg: enabled compatibility flags:
gpg: using pgp trust model
gpg: using subkey ED9DE45B6A89074E instead of primary key A636EAAC3A0F2C30
gpg: ED9DE45B6A89074E: There is no assurance this key belongs to the named user

sub  elg2048/ED9DE45B6A89074E 2024-10-24 joseA (claves para joseA) <josebenitez-19@hotmailm.com>
 Primary key fingerprint: 0F4E 8257 1B1C 1742 2536  EBE8 A636 EAAC 3A0F 2C30
      Subkey fingerprint: D83B F049 D9A9 6A7A E27B  88F2 ED9D E45B 6A89 074E

It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y
gpg: reading from 'mensaje'
File '/tmp/mensaje' exists. Overwrite? (y/N) y
gpg: writing to '/tmp/mensaje'
gpg: ELG/AES256.CFB encrypted for: "ED9DE45B6A89074E joseA (claves para joseA) <josebenitez-19@hotmailm.com>"
gpg: WARNING: 'mensaje' is an empty file
```

3.16. Una vez introducido ese comando, es posible que aparezca una advertencia: no hay seguridad de que esa clave pública sea realmente la clave pública de kali. Cualquiera podría haber cambiado el fichero /tmp/clavePublica.pub antes de que usuarioftp importara las claves. Entonces, la huella anotada en el paso 11 nos permitiría comprobar la integridad del mensaje que pensamos enviar de manera confidencial. Haz por ti mismo esta comprobación.

3.17. Una vez cifrado el mensaje, cambiaremos de nuevo de sesión a la del usuario kali , y descifraremos -utilizando la clave privada- el mensaje que acaba de cifrar usuarioftp con nuestra la clave pública. Para ello necesitaremos el parámetro --decrypt. El comando nos solicita la contraseña que da acceso a la clave privada que introdujimos en el paso 6. Aparecerá una ventana indicando qué clave necesita (en nuestro caso, la clave del algoritmo Elgamal). Podremos elegir que el sistema recuerde esta clave; así nos ahorraremos volver a teclearla. ¿Sería recomendable que el sistema recordara la clave para siempre? ¿Por qué? Si introducimos bien la contraseña, debería aparecer el mensaje enviado por el primer usuario.

```
┌──(kali㉿kali)-[/tmp]
└─$ gpg --decrypt /tmp/mensaje.cifrado
gpg: encrypted with 2048-bit ELG key, ID ED9DE45B6A89074E, created 2024-10-24
      "joseA (claves para joseA) <josebenitez-19@hotmailm.com>"
-----BEGIN PGP MESSAGE-----

hQIOA+2d5FtqiQdOEAgAgZH5ZYwbNQyNsrxZ/eJNX1YSny9n2hi/ILgd6sKAt9Nf
AhiSy5jGD37alPaE6CDWVpIxDYZqXYskIFe24gCaUBjHmKf8a732PVfG5kLjatQv
rxDWbeZfhL8FUx/Dpn3M90g5LCG+UjIAbFQZZm67pwV5Abf9KX+EoHsm9BNkJbzb
jhs74qRLhPE6Nu08csAqh8/2emmbZRX7aurUTQ+n4d70RiStjVOXSFEeDEO2dflY
Bk0zCJPiKuYt8MJl41rb5di+l1VGrlPVO1FLlkRjkYIL1jVv7jxrdUmddrhPcMFe
+NGRTzHHBFidRUxmmrJExyRsOdHXyn5SD9/DoMoGDAgAhM+oM3hxygtZ5RO7wn7D
AF693/MQ8gOx9BYUqYnw9+YAhfF8rFKzFv45vTzd9+l7YFXW7vQINJlGH2IMeX8M
YsfhgZjyXgMeClOjxZHJ9NtlvWuy2QcVrxdtzaMij5K90pcdIeAPPXK8fYmiXRuH
mE8r3diKAR1Idp9fRfOm5eqZJI56E6VKyZEo5uamsex7gNV21xNChL/cIC0viogl
hDGbIuNDwLT1R4W8bigGKL9vog0ptqKM18siQIScOe0oiNsvGnlJ/A+7iKN8bAbY
X6lkSWqcyIJJMpTQzf71HbNKlyWvv7xqGgg3OX4v7QfDiSQKk4+T2fN/eGggKhBT
j9JCAXAm3b96js2QLRgXxksMES1VOK0×CU8EnzuThDnd0WQhjf/jdnSaRRoooCv/
fOB6pDjzorFuMM9VtOj7DVGIwF2O
=2V2c
-----END PGP MESSAGE-----
```

3.18. Por último, veremos que sucede si por algún casual el mensaje cifrado se daña en el proceso de transmisión. Para ello, crea una copia del mensaje cifrado, y altéralo escribiendo cualquier cosa en él.
Después prueba a descifrar el mensaje e indica lo que sucede.

```
┌──(kali㉿kali)-[/tmp]
└─$ cp mensaje.cifrado mensaje.cifrado.bak

┌──(kali㉿kali)-[/tmp]
└─$ gpg --decrypt /tmp/mensaje.cifrado
gpg: CRC error; 90095F - 8B3EEF
gpg: mpi larger than indicated length (2064 bits)
```

Ejercicio 4

Genera un par de claves mediante el parámetro --gen-key e indica las diferencias entre las claves generadas por defecto, y las generadas en el ejercicio anterior. A partir de ahora será el parámetro que utilicemos cada vez que queramos generar un par de claves.

Flexibilidad: Al utilizar --gen-key, GPG ofrece más opciones para personalizar la configuración de las claves, como elegir el algoritmo de cifrado, el tamaño de la clave, la fecha de expiración, etc. Esto permite crear claves que se adapten mejor a tus necesidades específicas de seguridad y gestión de claves.

Seguridad: Al generar claves con --gen-key, puedes elegir algoritmos y tamaños de clave más seguros, como RSA de 4096 bits, para proporcionar una mejor protección contra ataques de fuerza bruta.

Gestión de Claves: Con --gen-key, puedes generar subclaves para diferentes propósitos, como cifrado, firma o autenticación. Esto facilita la gestión de tus claves, ya que puedes asignar diferentes permisos y restricciones a cada subclave.

```
└─$ gpg --gen-key
gpg (GnuPG) 2.2.43; Copyright (C) 2023 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: juanA
Email address: josebenitez-20@hotmail.com
You selected this USER-ID:
    "juanA <josebenitez-20@hotmail.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: revocation certificate stored as '/home/kali/.gnupg/openpgp-revocs.d/2CD3D4D910DC2A7C9D0D2FA1E2A6A17496AF295A.rev'
public and secret key created and signed.

pub   rsa3072 2024-10-24 [SC] [expires: 2027-10-24]
      2CD3D4D910DC2A7C9D0D2FA1E2A6A17496AF295A
uid                      juanA <josebenitez-20@hotmail.com>
sub   rsa3072 2024-10-24 [E] [expires: 2027-10-24]
```

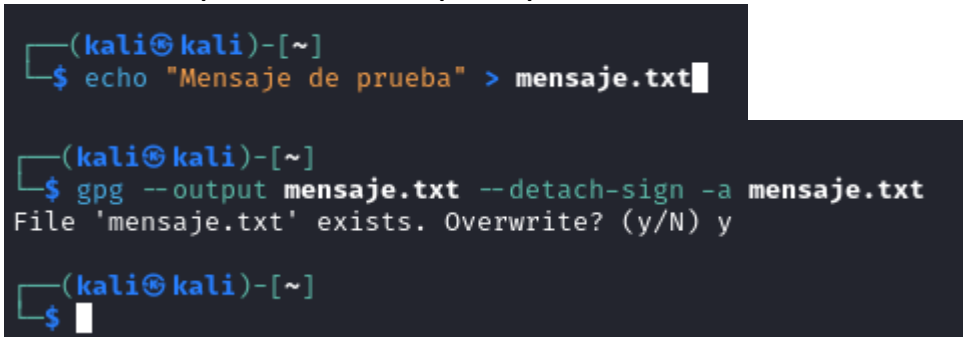
Ejercicio 5

Este ejercicio será una ampliación de los ejercicios 1 y 3, consistente en llevarlos a un escenario algo más real; para así afianzar conceptos y procedimientos en interrelación. En este escenario tenemos una máquina cliente Windows que mediante FTP transmitirá su clave pública al servidor Linux; para que éste pueda utilizar dicha clave pública para cifrar un mensaje que mandará de vuelta mediante FTP al cliente Windows. Así el cliente, mediante el uso de su clave privada (habrá de generarse un par de claves como paso previo), podrá obtener el texto en claro que el servidor Linux le quiso transmitir. Nótese que al estar usuarioftp enjaulado, el mensaje a transmitir se habrá de colocar en el directorio adecuado para que el cliente pueda tener acceso a él...

Ejercicio 6

En el Ejercicio 3 creamos 2 pares de claves, y utilizamos las claves Elgamal para cifrar mensajes. Ahora emplearemos el otro par de claves para firmar documentos.

6.1. Con el usuario kali creamos un fichero mensaje y lo firmamos con nuestra clave privada para que cualquiera pueda confirmar que es nuestro. Usaremos el parámetro --detach-sign, que crea un fichero nuevo solo con la firma (el cifrado del resultado de aplicar el hash al fichero original). Utilizaremos también el parámetro -a para poder examinar el fichero.



6.2. Veremos que la herramienta nos pide la contraseña de nuestra clave secreta de tipo DSA (directamente utiliza esa, no Elgamal). Como al cifrar, es recomendable evitar que se recuerde la contraseña.
6.3. Examina el documento apenas creado.

```
┌──(kali㉿kali)-[~]
└─$ cat mensaje.txt.sig
-----BEGIN PGP SIGNATURE-----

iHUEABEIAB0WIQTpLRCQqvxyiks+FOsZjo2DDJGk4AUCZyOBiwAKCRAZjo2DDJGk
4B0CAP9pM3t6X8RfKvR+uRnGTONP3ju1S5WVrMiJ7haKDazARwEArRj9NwLwIPdT
ueu/wXTEajS5J0pEhI7274U/c3/mahg=
=S1lD
-----END PGP SIGNATURE-----

┌──(kali㉿kali)-[~]
└─$
```

6.4. Ahora transferiremos mediante FTP a nuestra máquina Windows tanto el mensaje y la firma recientemente generadas; como la clave pública del servidor. Recordemos que ésta última se encuentra en /tmp/clave.pub y que usuarioftp está enjaulado. Tampoco te olvides de importar la clave pública en el cliente.

```
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
archivosftp
clave.pub
mensaje.gpg
mensaje.txt
mensaje.txt.sig
226 Directory send OK.
ftp: 70 bytes recibidos en 0.00segundos 70000.00a KB/s.
ftp> get clave.pub
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for clave.pub (2322 bytes).
226 Transfer complete.
ftp: 2322 bytes recibidos en 0.00segundos 2322000.00a KB/s.
ftp> get mensaje.txt
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for mensaje.txt (228 bytes).
226 Transfer complete.
ftp: 228 bytes recibidos en 0.00segundos 228000.00a KB/s.
ftp> get mensaje.txt.sig
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for mensaje.txt.sig (228 bytes).
226 Transfer complete.
```

6.5. Hecho esto, y trabajando ya desde el cliente, comprobaremos que el mensaje se ha firmado con la clave privada asociada a la clave pública del servidor que tenemos almacenada. Esto lo hacemos mediante el parámetro --verify. No obstante, nos percataremos que la autoría aún no está confirmada. Es decir, cabe la posibilidad de que se haya "falsificado". Esto se conoce como un "problema de confianza". Hay varias formas de resolverlo: ◦ Si estamos seguros de que la firma se corresponde con quien dice haberlo firmado, podemos firmar la clave pública con nuestra clave privada para así otorgar confianza: esto es lo que haremos más adelante mediante el parámetro --sign-key. Se sigue una red de confianza descentralizada. Es la estrategia empleada con la herramienta gpg. ◦ Mediante un certificado emitido por alguna entidad reconocida. Se sigue una estrategia jerárquica en el contexto de una infraestructura de clave pública (PKI).Es la estrategia seguida con la herramienta OpenSSL.

```
┌──(kali㉿kali)-[~]
└─$ gpg --verify mensaje.txt.sig mensaje.txt
gpg: Signature made Thu 31 Oct 2024 02:09:31 PM CET
gpg:                using DSA key E92D1090AAFC728A4B3E14EB198E8D830C91A4E0
gpg: Good signature from "jose antonio <josebenitez-19@hotmail.com>" [ultimate]
```

6.6. Prueba ahora a alterar el mensaje y a verificar de nuevo la firma. ¿Qué sucede?

```
┌──(kali㉿kali)-[~]
└─$ gpg --verify mensaje.txt.sig mensaje.txt
gpg: Signature made Thu 31 Oct 2024 02:09:31 PM CET
gpg:                using DSA key E92D1090AAFC728A4B3E14EB198E8D830C91A4E0
gpg: BAD signature from "jose antonio <josebenitez-19@hotmail.com>" [ultimate]
```

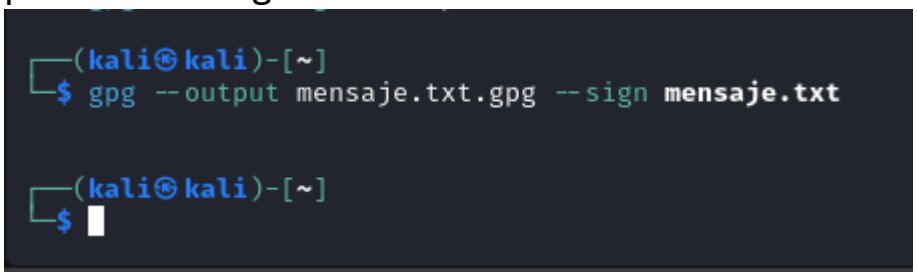
6.7. Repite los puntos anteriores, pero esta vez firma mediante el parámetro --clearsign en vez de con –detach-sign. Comenta y documenta las diferencias encontradas.

```
┌──(kali㉿kali)-[~]
└─$ gpg --clear-sign mensaje.txt

┌──(kali㉿kali)-[~]
└─$
```

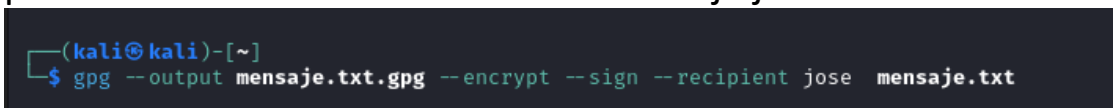
Esta opción permite leer el contenido original junto con la firma.

6.8. Repite el apartado anterior, pero esta vez firma con el parámetro --sign.



6.9. Habrás comprobado que con este último parámetro, ciframos el contenido del mensaje transmitido. Así que descífralo en el cliente y comenta y documenta lo que obtienes. Windows me dice que no se puede descifrar

6.10. Quizá te hayas percatado de que este mecanismo de cifrado de un documento junto con su firma es bastante débil: cualquiera con acceso a la clave pública del servidor podría descifrar el documento. Una idea mejor es cifrar con la clave pública del cliente (en el Ejercicio 5 generamos los correspondientes pares de claves) para que así solo éste pudiera descifrar el contenido del mensaje junto con su firma.



6.11.Como ya adelantamos, para confirmar que la clave pública del servidor es auténtica, utilizaremos el parámetro sign-key y el UID de la clave: gpg --sign-key UIDservidor Desde este momento las verificaciones ya no deberían emitir el aviso de que la clave es sospechosa. Haz las comprobaciones necesarias. No se hacerlo

6.12. Crea un nuevo mensaje. Firma la clave pública del cliente (que deberá estar debidamente importada en el servidor) para evitar mensajes de falta de confianza.

6.13. Firma y cifra con la clave pública del cliente el mensaje apenas creado, de tal manera que se genere un solo documento que contenga tanto el mensaje cifrado como la firma en formato legible para los humanos.

6.14. Ahora, transfiere el mensaje mediante FTP al cliente.

6.15.Allí verifica la firma, y descifra el mensaje usando la clave privada del cliente.
6.16.Documenta todo el proceso descrito en los últimos 4 puntos.

6.17. Para acabar, borra las claves del cliente y prueba de nuevo a verificar y descifrar el mensaje. ¿Qué ocurre? Documéntalo.