

Practica 3. Hashing, Cifrado y Certificados

Primera Parte

1. **A partir de la página de manual sobre la forma de uso de md5sum y sha1sum (y documentación extra que pueda obtener sobre MD5 y SHA-*), elabora un resumen de cómo se emplean estas aplicaciones para generar hashcodes a partir de documentos. Discuta brevemente las debilidades conocidas y reportadas.**

Las funciones hash son algoritmos que se encargan de codificar cadenas o archivos en una cadena de una longitud determinada de caracteres (Su número depende de la función hash utilizada). Los comandos md5sum y sha1sum codifican utilizando una función hash determinada, de distinta longitud.

Podemos utilizar md5sum y sha1sum de dos maneras distintas. La primera es pasando el texto plano que queramos codificar mediante una tubería. La otra opción es pasándole el nombre del archivo que queremos codificar. En el ejemplo primero uso la primera forma y después la segunda.

```
mallet@mallet:~$ echo hola | md5sum
916f4c31aaa35d6b867dae9a7f54270d -
mallet@mallet:~$ echo hola | sha1sum
63bbfea82b8880ed33cdb762aa11fab722a90a24 -
mallet@mallet:~$ md5sum testttttt.pdf
76379ef9ddf935d80b397646ece7f0bc testttttt.pdf
mallet@mallet:~$ sha1sum testttttt.pdf
bcf3202e577284717465cbe7d6ebbd76e7223dc0 testttttt.pdf
```

Estas dos formas de crear hashcodes tienen conocidas debilidades de colisión. Esto quiere decir que para dos archivos distintos puede crear el mismo código hash. Si queremos utilizar opciones referentes a sha1 o md5, como leer en modo binario, en modo texto... las pondremos entre la instrucción y el archivo nombrado.

2. **Use ambas herramientas para construir el hash asociado a diferentes tiras de caracteres que difieran en pocos caracteres. Analice y documente las diferencias entre ambas.**

Utilicemos las tuberías para codificar la palabra “hola” y “ola”.

```
mallet@mallet:~$ echo hola | md5sum
916f4c31aaa35d6b867dae9a7f54270d -
mallet@mallet:~$ echo ola | md5sum
5d3a61408eb4ba89aeaf09818561d0a7 -
mallet@mallet:~$ echo hola | sha1sum
63bbfea82b8880ed33cdb762aa11fab722a90a24 -
mallet@mallet:~$ echo ola | sha1sum
03c7085fb873c3b7fd2b1bde538f5b8e71f1cb9a -
mallet@mallet:~$
```

Vemos que las salidas son distintas en ambos comandos para ambas palabras.

Las diferencias entre md5 y sha1 es que el primero codifica devolviendo 32 caracteres en hexadecimal (128 bits) y el segundo sha1 devuelve 40 caracteres en hexadecimal (160 bits).

3. Elija cualquier fichero presente en tu ordenador (puedes crearlo) y obtenga el hashcode. Usando ambas herramientas, describiendo la salida que se obtiene

Creamos un archivo de texto plano para ejecutar los ejemplos. Ejecutaremos las dos maneras de crear hashcodes y posteriormente modificaremos el archivo añadiendo un salto de carro. Volveremos a ejecutar ambas instrucciones.

Este es el contenido inicial del archivo creado.

```
admin
1234
qwerty
```

Ejecutamos las instrucciones.

```
mallet@mallet:~$ md5sum claves.txt
9d7e1ef72c57c89e4ad19f0c02a7ca7c  claves.txt
mallet@mallet:~$ nano claves.txt
mallet@mallet:~$ md5sum claves.txt
ddc1be15ce35f635086008f229a3fab2  claves.txt
```

```
mallet@mallet:~$ shasum claves.txt
0ad43a4ebb0b45ba83cd123d61775abf21ebd5c0  claves.txt
mallet@mallet:~$ nano claves.txt
mallet@mallet:~$ shasum claves.txt
5cd1caed7d23e19f288b262e85335a71fb20cb09  claves.txt
```

Como se puede apreciar, a pesar de que el cambio es casi imperceptible de manera visual se generan dos hashcodes distintos. Esto nos da a entender que se ha modificado el archivo, cosa que es correcta.

La salida obtenida en md5 un código de 32 caracteres en hexadecimal (128 bits) y en sha1 es un código de 40 caracteres en hexadecimal (160 bits) seguidos del nombre del archivo codificado.

4. Sobre los hashes generados en el punto anterior, utiliza la herramienta hashid para tratar de obtener cuál ha sido la función de hash utilizada para crear los resúmenes anteriores.

A partir de aquí haré uso de una máquina Ubuntu para realizar los ejercicios planteados. Para este ejercicio creare un fichero txt, lo codificare con los dos comandos dados y utilizaré hashid para ver si identifica correctamente el algoritmo usado.

```

alejandro@alejandro-VirtualBox:~$ nano texto.txt
alejandro@alejandro-VirtualBox:~$ md5sum texto.txt
840ddb0e48ace8be70745da81cda04c4  texto.txt
alejandro@alejandro-VirtualBox:~$ hashid 840ddb0e48ace8be70745da81cda04c4
Analyzing '840ddb0e48ace8be70745da81cda04c4'
[+] MD2
[+] MD5
[+] MD4
[+] Double MD5
[+] LM
[+] RIPEMD-128
[+] Haval-128
[+] Tiger-128
[+] Skein-256(128)
[+] Skein-512(128)
[+] Lotus Notes/Domino 5
[+] Skype
[+] Snefru-128
[+] NTLM

```

```

alejandro@alejandro-VirtualBox:~$ sha1sum texto.txt
5cd1caed7d23e19f288b262e85335a71fb20cb09  texto.txt
alejandro@alejandro-VirtualBox:~$ hashid 5cd1caed7d23e19f288b262e85335a71fb20cb09
Analyzing '5cd1caed7d23e19f288b262e85335a71fb20cb09'
[+] SHA-1
[+] Double SHA-1
[+] RIPEMD-160
[+] Haval-160
[+] Tiger-160
[+] HAS-160
[+] LinkedIn
[+] Skein-256(160)
[+] Skein-512(160)

```

Como vemos, para md5 nos sale como segunda opción y para sha1 como primera, es decir, la herramienta funciona de manera adecuada.

5. A partir de la información sobre el fichero (propiedades) que se pueden obtener, por ejemplo, con la orden `ls -l`, genere un hashcode. Modifique alguna propiedad del fichero y vuelva a obtener el hashcode. ¿qué conclusiones obtienes?

```

mallet@mallet:~$ ls -l testttttt.pdf
-rw-r--r-- 1 mallet mallet 390834 2012-03-14 14:30 testttttt.pdf
mallet@mallet:~$ ls -l testttttt.pdf | md5sum
609da58c38edb45fae449f262ed783d4 -
mallet@mallet:~$ ls -l testttttt.pdf | sha1sum
f849a8fd9c257408ee446d0eaa34ecf24745dc0e -
mallet@mallet:~$ chmod 777 testttttt.pdf
mallet@mallet:~$ ls -l testttttt.pdf
-rwxrwxrwx 1 mallet mallet 390834 2012-03-14 14:30 testttttt.pdf
mallet@mallet:~$ ls -l testttttt.pdf | md5sum
5d07c644bb2b587a24de867204f532fa -
mallet@mallet:~$ ls -l testttttt.pdf | sha1sum
5e9cf44b94de227770fd2a99cd51d1730c89d6ec -

```

Como podemos ver, si cambiamos los permisos de un archivo también se cambia el hashcode producido. Esto asegura la integridad de los archivos codificados.

6. Sobre un sistema Ubuntu20.04.2 LTS, crea dos usuarios (root/toor). (admin/123456). Determina cuál es la función de hash utilizada para que las claves de los usuarios no se almacenen en texto plano y realiza un ataque de fuerza bruta sobre las contraseñas de los usuarios almacenadas por el sistema operativo en el fichero /etc/shadow. ¿Para qué puede ser útil la herramienta John The Ripper?

Creamos los usuarios con permisos y les damos una contraseña

```
alejandro@alejandro-VirtualBox:~$ sudo useradd -u 0 -o -g 0 Root
alejandro@alejandro-VirtualBox:~$ sudo passwd Root
New password:
Retype new password:
passwd: password updated successfully
alejandro@alejandro-VirtualBox:~$ sudo useradd -u 0 -o -g 0 Admin
alejandro@alejandro-VirtualBox:~$ sudo passwd Admin
New password:
Retype new password:
passwd: password updated successfully
alejandro@alejandro-VirtualBox:~$
```

Intentamos leer las contraseñas de los usuarios creados, pero están codificadas.

```
alejandro@alejandro-VirtualBox:~$ sudo cat /etc/shadow | grep -i Root
root:!:19279:0:99999:7:::
Root:$6$.Y5MQ4T4MTH1vjKG$/mSltqZxAHj3AqUiAuaKGptl8gbpllF0bU8uiwp/rab7fJPozbULLj
WTvbI4Wlsd.Ex3QKljPCq0ov/KIc1dT0:19279:0:99999:7:::
alejandro@alejandro-VirtualBox:~$ sudo cat /etc/shadow | grep -i Admin
Admin:$6$D8jZGqohfvN/SfN/$mEsKphP4yN8PuxdWA/5iTwZamLJ0o418uwMUDVZVJ1mptmUQ9Irfg
swqI/OJXIeWTw7.NKkpwiBmgKS0pD/Ov1:19279:0:99999:7:::
```

En el formato de codificación podemos obtener una pista sobre que función hashes está utilizando. Después del nombre de usuario (señalado en rojo) podemos ver un número entre \$, este número nos dirá el tipo de función hash utilizada. En la página web <https://linuxize.com/post/etc-shadow-file/> vemos algunos ejemplos.

- \$1\$ – MD5
- \$2a\$ – Blowfish
- \$2y\$ – Eksblowfish
- \$5\$ – SHA-256
- \$6\$ – SHA-512

En nuestro caso, Ubuntu utiliza sha512 para codificar las contraseñas de los usuarios.

Para hallar la contraseña sin cifrar, podemos utilizar la herramienta “John The Ripper”. Esta herramienta combina varios descifradores de contraseñas y autodetecta los tipo de hash utilizados en las contraseñas.

```

alejandro@alejandro-VirtualBox:~$ sudo john /etc/shadow
Created directory: /root/.john
Loaded 3 password hashes with 3 different salts (crypt, generic crypt(3) [?/64]
)
Press 'q' or Ctrl-C to abort, almost any other key for status
Toor          (Root)
123456        (Admin)
admin         (alejandro)
3g 0:00:00:20 100% 2/3 0.1458g/s 435.9p/s 440.7c/s 440.7C/s Johnson..buzz
Use the "--show" option to display all of the cracked passwords reliably
Session completed
alejandro@alejandro-VirtualBox:~$

```

Ahora crearé un pequeño diccionario que contenga algunas de las contraseñas más utilizadas para hacer un ataque de fuerza bruta.

```

GNU nano 4.8          diccionario.lst          Modified
1
12
123
1234
12345
123456
Toor
Root
Admin
admin

```

Tras haber creado el diccionario ejecutaremos un ataque de fuerza bruta utilizándolo. Posteriormente pediremos las contraseñas y usuarios a los que hemos conseguido acceder.

```

alejandro@alejandro-VirtualBox:~$ nano diccionario.lst
alejandro@alejandro-VirtualBox:~$ sudo john --wordlist=diccionario.lst /etc/sha
dow
Loaded 3 password hashes with 3 different salts (crypt, generic crypt(3) [?/64]
)
No password hashes left to crack (see FAQ)
alejandro@alejandro-VirtualBox:~$ sudo john --show /etc/shadow
alejandro:admin:19279:0:99999:7:::
Root:Toor:19279:0:99999:7:::
Admin:123456:19279:0:99999:7:::

3 password hashes cracked, 0 left

```

7. Construya un programa [buildhash] (shell script o como prefiera), que obtenga para todos y cada uno de los ficheros cuyos nombres aparecen (uno por línea) en un fichero de texto de entrada dos hashcodes: uno asociado al propio fichero y otro a sus propiedades. Para cada fichero, se generará una línea que contenga el nombre de fichero, el hashcode del fichero y el hashcode de sus propiedades, separados por ';'.

Haremos el programa usando Python como lenguaje de programación. Utilizaremos sha256 para codificar los archivos ya que no tiene los problemas de colisión que hemos comentado anteriormente.

```
GNU nano 4.8 ej7.py
import hashlib,os,sys

# Se pide el fichero de entrada
P Files "Fichero de entrada: ")

# Se lee el fichero de entrada
File = open(input(), "r").read().splitlines()

# Se crea el fichero de salida
ExitFile = open("Salida7.txt", "w")

# Se lee linea por linea el fichero de entrada
for index in File:
    # Se consiguen las propiedades de los ficheros
    props = os.stat(index)

    # Se hashean tanto los ficheros como sus propiedades en sha256
    hash = hashlib.sha256(open(index, "r").read().encode()).hexdigest()
    hashprops = hashlib.sha256(str(props).encode()).hexdigest()

    # Se escriben los hash en el fichero de salida
    ExitFile.write(index + ";" + hash + ";" + hashprops + "\n")

ExitFile.close()
```

Para comprobar que su funcionamiento es correcto crearemos 3 ficheros cuya dirección se encontrará en un cuarto fichero llamado lanzadera.

```
alejandro@alejandro-VirtualBox:~$ cat lanzadera.txt
/home/alejandro/fichero.txt
/home/alejandro/fichero2.txt
/home/alejandro/fichero3.txt
```

Después ejecutamos el programa y vemos que por cada fichero saca 2 hashes, uno del propio fichero y otro de sus propiedades.

```
alejandro@alejandro-VirtualBox:~$ python3 ej7.py
Fichero de entrada:
lanzadera.txt
alejandro@alejandro-VirtualBox:~$ cat Salida7.txt
/home/alejandro/fichero.txt;33b280cff93237fa89d23faf9dfb9ae74b4e7da86b4d3ec8f77
8526a4675724c;166887f32487e618f6ae4cddef8b6e2c21879c9be9e038012ddbfb1b62a3aad7
/home/alejandro/fichero2.txt;10efd8806dd355517980dbfe6a5d55759d6d9174d09f2708b7
c27e2c2a39e906;3b3ad56fee541d43ad2d0571229f9fdda7438f5fc7319f1007aa5fa888e62b13
/home/alejandro/fichero3.txt;33f0256a005f9ffb3170f0bdf991381b2a36875641c74cea
0e7828484879c0;ba5b2cd2c42b09ab9e8ea9086ceb9a6f580942a95e9befcd044d9a0a379b97ae
```

Por lo tanto, el programa funciona correctamente.

8. **Construya un programa [checkhash] (shell script o como prefiera), que tome como entrada el fichero generado por el anterior y compruebe, para cada fichero, si se han producido cambios en el mismo o en sus propiedades, generando una salida en que se muestre cada fichero y se indique si se ha modificado o no.**

Construimos el programa usando Python.

```
import hashlib,os,sys

# Se pide dirección fichero de entrada
print("Fichero de entrada: ")

# Se abre el fichero
File = open(input(), "r").read().splitlines()

# Se crea el fichero de salida
ExitFile = open("Salida8.txt", "w")

# Se abre el fichero original
origin = open("lanzadera.txt", "r")

# Se lee el fichero original
OriginalFile= origin.readlines()

contador=0

# Se lee el archivo de entrada línea por línea
for index in File:

    # Se separa el fichero en función de los ;
    split = index.split(";")

    # Se consiguen las propiedades de los archivos
    props = os.stat(OriginalFile[contador].rstrip())

    # Se hashean estas propiedades con sha256
    hashprops = hashlib.sha256(str(props).encode()).hexdigest()

    # Hallan cambiado o no se envia un mensaje informando
    if hashprops == split[2].rstrip():
        mensajeProps = "Props íntegras"
    else:
        mensajeProps = "Props comprometidas"

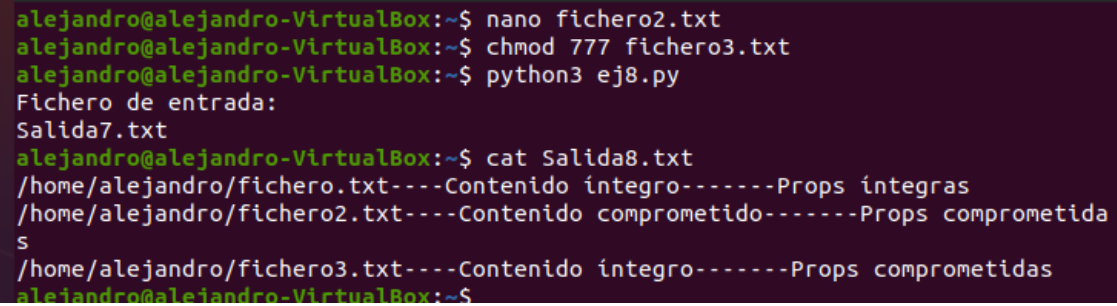
    # Se hashean los archivos
    hash = hashlib.sha256(open(OriginalFile[contador].rstrip(), "r").read().>

    # Hallan cambiado o no se informa por mensaje
    if hash == split[1]:
        mensajeCont = "Contenido íntegro"
    else:
        mensajeCont = "Contenido comprometido"

    # Se escriben los resultados en el fichero de salida
    ExitFile.write(split[0] + "----" + mensajeCont + "-----" + mensajePr>
    contador+=1
```

Para comprobar que funciona modificaremos las propiedades y el contenido del fichero2, las propiedades del fichero 3 y al primer fichero no le modificaremos nada.

Ejecutamos el programa y le pasamos el fichero de resultados del ejercicio anterior.

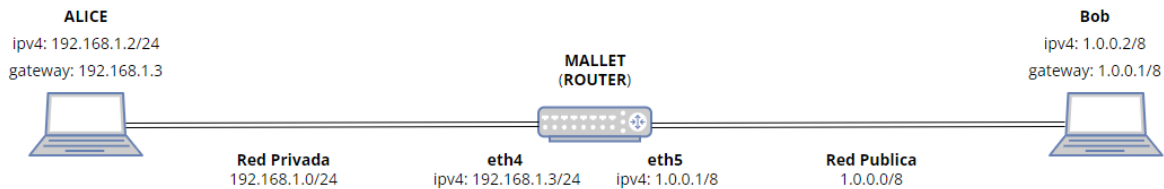
A terminal window with a dark purple background and light green text. The user is in a directory named 'alejandro' on a system named 'alejandro-VirtualBox'. The commands and output are as follows:

```
alejandro@alejandro-VirtualBox:~$ nano fichero2.txt
alejandro@alejandro-VirtualBox:~$ chmod 777 fichero3.txt
alejandro@alejandro-VirtualBox:~$ python3 ej8.py
Fichero de entrada:
Salida7.txt
alejandro@alejandro-VirtualBox:~$ cat Salida8.txt
/home/alejandro/fichero.txt---Contenido íntegro-----Props íntegras
/home/alejandro/fichero2.txt---Contenido comprometido-----Props comprometida
s
/home/alejandro/fichero3.txt---Contenido íntegro-----Props comprometidas
alejandro@alejandro-VirtualBox:~$
```

Como se aprecia en la imagen el programa se ejecuta correctamente.

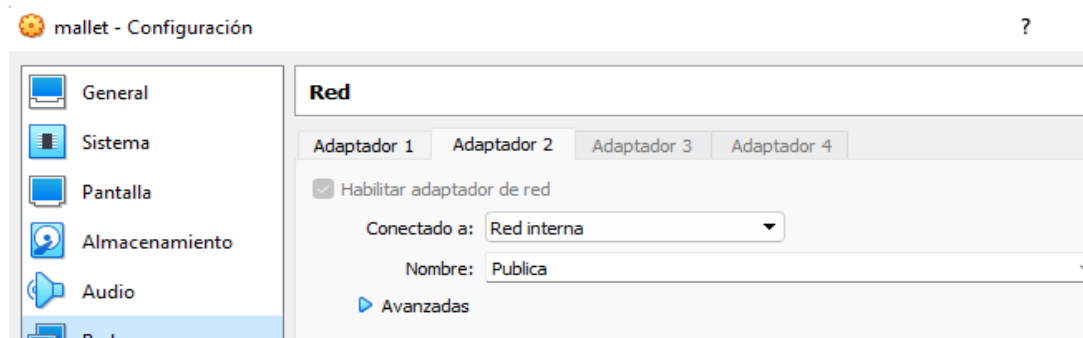
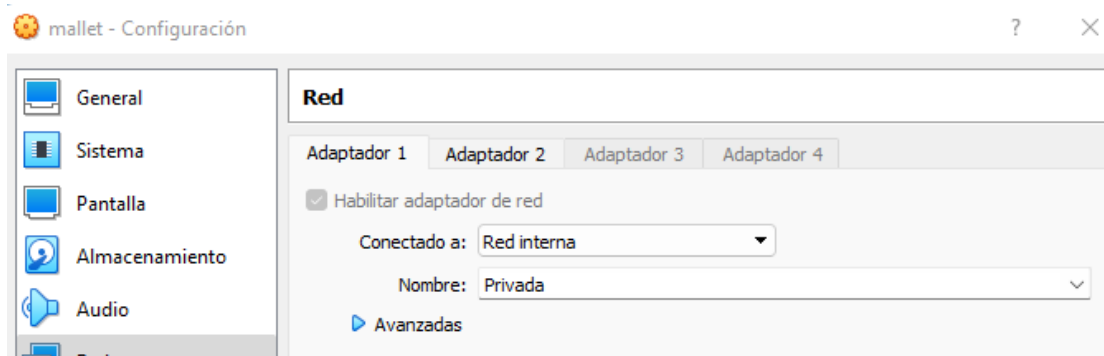
Segunda Parte

1. Dibuja el diagrama de red lo más detallado posible.

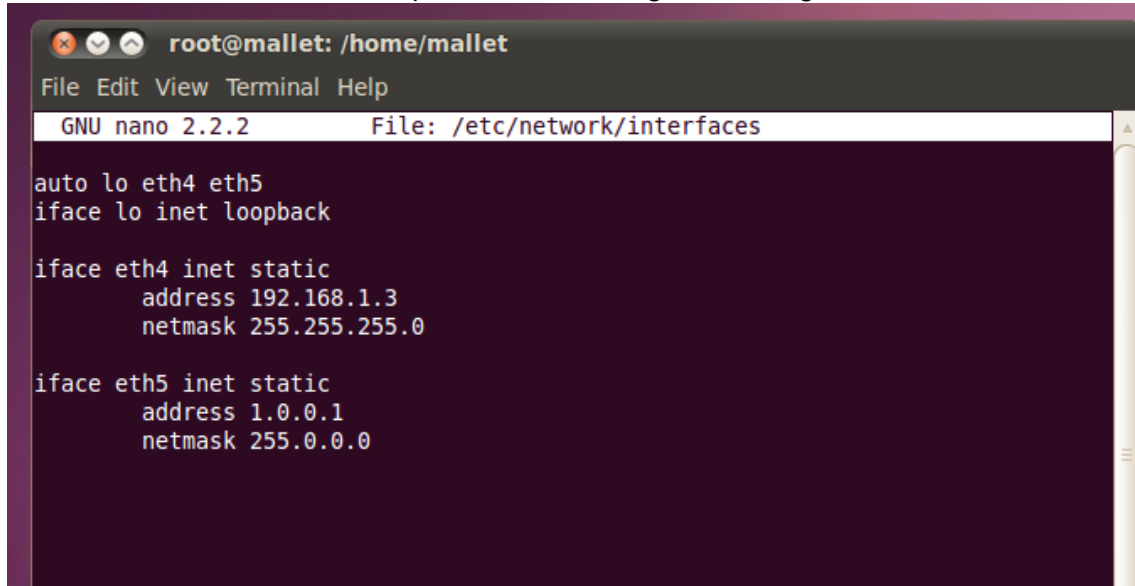


2. Configura la máquina mallet para que actúe como router. Tendrá dos interfaces de red, la eth4 con dirección 192.168.1.3/24 para eth4 en la red interna "Privada", y la eth5 con dirección 1.0.0.1/8 en la red interna "Pública". Deberá tener habilitado el bit de enrutamiento.

Primero nos iremos al menú de configuración de la máquina virtual de Mallet. Una vez allí nos iremos al apartado de red, donde configuraremos 2 adaptadores de red. Un adaptador estará conectado a una red interna llamada "Privada" y el otro a una llamada "Pública".



Una vez que tenemos configurados los adaptadores encenderemos la máquina de Mallet para configurar las interfaces de red. Para ello editaremos el archivo de interfaces mediante el comando “sudo nano /etc/network/interfaces”. Tras poner la contraseña para obtener los privilegios de super-usuario modificamos el archivo obteniendo como resultado lo que vemos en la siguiente imagen.



```

root@mallet: /home/mallet
File Edit View Terminal Help
GNU nano 2.2.2 File: /etc/network/interfaces

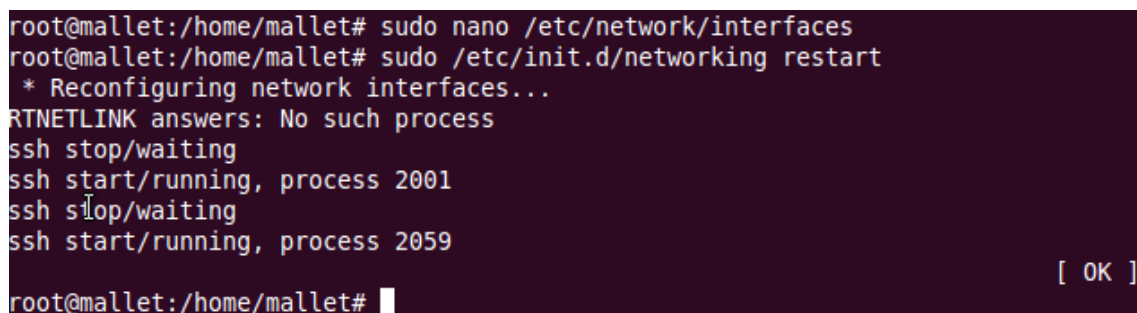
auto lo eth4 eth5
iface lo inet loopback

iface eth4 inet static
    address 192.168.1.3
    netmask 255.255.255.0

iface eth5 inet static
    address 1.0.0.1
    netmask 255.0.0.0
  
```

Como hemos visto en el diagrama de red del primer ejercicio, Mallet tendrá dos interfaces (una para cada red) llamadas eth4 y eth5. Las configuraremos estableciendo una ipv4 estática, con las direcciones y máscaras pedidas en el enunciado.

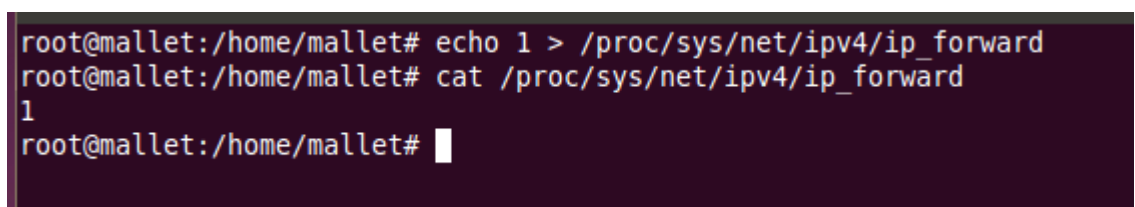
Ahora nos toca reiniciar los servicios de red para que se apliquen los cambios. Esto lo haremos con el comando “sudo /etc/init.d/networking restart”.



```

root@mallet:/home/mallet# sudo nano /etc/network/interfaces
root@mallet:/home/mallet# sudo /etc/init.d/networking restart
* Reconfiguring network interfaces...
RTNETLINK answers: No such process
ssh stop/waiting
ssh start/running, process 2001
ssh stop/waiting
ssh start/running, process 2059
[ OK ]
root@mallet:/home/mallet#
  
```

Para que la máquina de Mallet actúe como un router y permita la transmisión de paquetes debemos activar el bit de enrutamiento cambiando su valor de 0 a 1. Mediante una tubería insertamos el valor 1 en el archivo “ip_forward” ubicado en “/proc/sys/net/ipv4”.



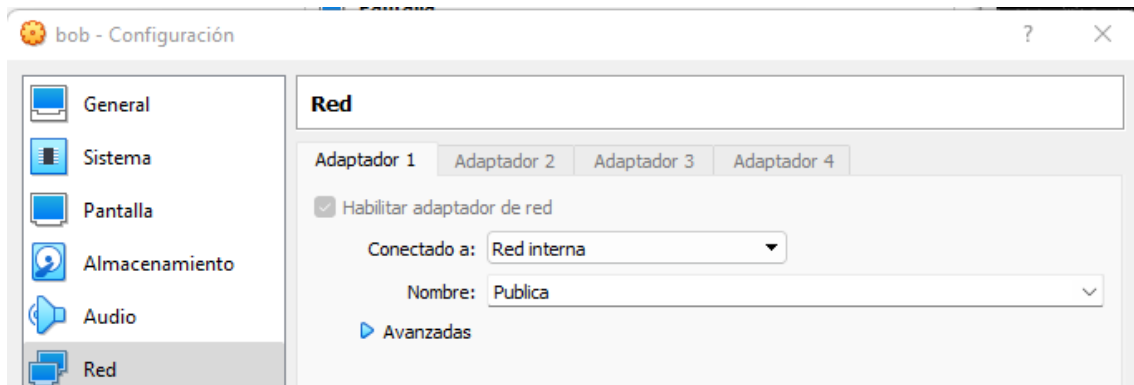
```

root@mallet:/home/mallet# echo 1 > /proc/sys/net/ipv4/ip_forward
root@mallet:/home/mallet# cat /proc/sys/net/ipv4/ip_forward
1
root@mallet:/home/mallet#
  
```

Realizamos un cat para comprobar que el contenido del archivo ha sido cambiado correctamente.

3. Configura la máquina bob para que esté dentro de la red interna "Pública" y tenga como dirección IP la 1.0.0.2/8, y como puerta de enlace la 1.0.0.1/8.

Como en Mallet, el primer paso es configurar el adaptador de red en el menú de configuración de red de la máquina virtual. Bob se encuentra en la red interna "Pública"



Ahora encendemos la máquina y configuramos la dirección ip y la pasarela mediante los comandos ifconfig "nombre interfaz" "dirección ipv4" y route add default gw "dirección ipv4 gateway" "nombre interfaz" respectivamente. Después mediante el comando ip a observamos si se han producido los cambios.

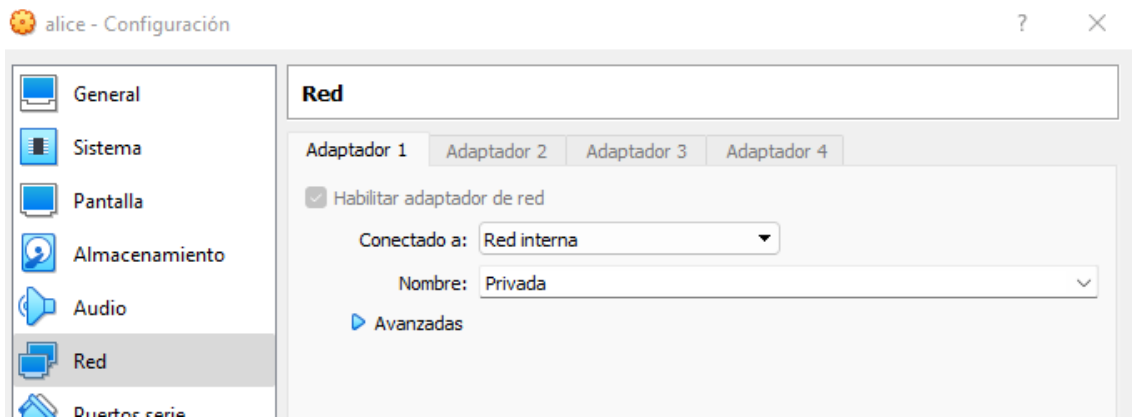
Para ver si tambien se ha configurado correctamente la pasarela podemos utilizar el comando "route -n"

```
bob:/home/bob# ifconfig eth0 1.0.0.2
bob:/home/bob# route add default gw 1.0.0.1 eth0
SIOCADDRT: File exists
bob:/home/bob# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:e2:73:8b brd ff:ff:ff:ff:ff:ff
    inet 1.0.0.2/8 brd 1.255.255.255 scope global eth0
    inet6 fe80::a00:27ff:fee2:738b/64 scope link
        valid_lft forever preferred_lft forever
3: sit0: <NOARP> mtu 1480 qdisc noop state DOWN
    link/sit 0.0.0.0 brd 0.0.0.0
bob:/home/bob# route -n
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
1.0.0.0        0.0.0.0         255.0.0.0       U        0      0        0 eth0
0.0.0.0        1.0.0.1         0.0.0.0         UG       0      0        0 eth0
bob:/home/bob# _
```

Vemos como todos los cambios se han realizado correctamente.

4. Configura la máquina Alice para que esté dentro de la red interna "Privada" y tenga como dirección IP la 192.168.1.2, y como puerta de enlace la 192.168.1.3/24.

Primero configuramos el adaptador de red conectando la máquina a la red "Privada".



Después, al igual que con Mallet, modificamos el archivo de interfaces para agregar la configuración pedida en el enunciado. Establecemos una dirección ip estática con la dirección, máscara y pasarela pedidas.

```

root@alice: /home/alice
File Edit View Terminal Help
GNU nano 2.2.2 File: /etc/network/interfaces

auto lo
iface lo inet loopback

auto eth3
iface eth3 inet static
    address 192.168.1.2
    netmask 255.255.255.0
    gateway 192.168.1.3

```

Para verificar si se han producido correctamente los cambios utilizamos los comandos “ip a” y “route -n” como ya hemos hecho con bob anteriormente.

```

root@alice: /home/alice
File Edit View Terminal Help
root@alice:/home/alice# sudo nano /etc/network/interfaces
root@alice:/home/alice# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:11:19:2f brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.2/24 brd 192.168.1.255 scope global eth3
        inet6 fe80::a00:27ff:fe11:192f/64 scope link
            valid_lft forever preferred_lft forever
root@alice:/home/alice# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth3
169.254.0.0 0.0.0.0 255.255.0.0 U 1000 0 0 eth3
0.0.0.0 192.168.1.3 0.0.0.0 UG 100 0 0 eth3
root@alice:/home/alice#

```

**5. Realiza pruebas a nivel de red para comprobar que las máquinas se comunican.
¿Qué ocurre con el valor del ttl cuando un paquete atraviesa un router?**

Para ver si se puede establecer conexión a través de la red haremos ping desde Alice a su Gateway, luego a la Gateway de Bob y después a Bob.

```
root@alice:/home/alice# ping 192.168.1.3 -c3
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
64 bytes from 192.168.1.3: icmp_seq=1 ttl=64 time=0.350 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=0.324 ms
64 bytes from 192.168.1.3: icmp_seq=3 ttl=64 time=0.320 ms

--- 192.168.1.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.320/0.331/0.350/0.019 ms
root@alice:/home/alice# ping 1.0.0.1 -c3
PING 1.0.0.1 (1.0.0.1) 56(84) bytes of data.
64 bytes from 1.0.0.1: icmp_seq=1 ttl=64 time=0.485 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=0.633 ms
64 bytes from 1.0.0.1: icmp_seq=3 ttl=64 time=0.566 ms

--- 1.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.485/0.561/0.633/0.063 ms
root@alice:/home/alice# ping 1.0.0.2 -c3
PING 1.0.0.2 (1.0.0.2) 56(84) bytes of data.
64 bytes from 1.0.0.2: icmp_seq=1 ttl=63 time=2.97 ms
64 bytes from 1.0.0.2: icmp_seq=2 ttl=63 time=2.40 ms
64 bytes from 1.0.0.2: icmp_seq=3 ttl=63 time=1.92 ms

--- 1.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 1.921/2.434/2.978/0.433 ms
```

Como se ve, hay comunicación desde Alice con todas las direcciones IP de las redes que hemos montados. Ahora veremos si con Bob pasa lo mismo.

```
bob:/home/bob# ping 1.0.0.1 -c2
PING 1.0.0.1 (1.0.0.1) 56(84) bytes of data.
64 bytes from 1.0.0.1: icmp_seq=1 ttl=64 time=0.474 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=1.74 ms

--- 1.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.474/1.109/1.745/0.636 ms
bob:/home/bob# ping 192.168.1.3 -c2
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
64 bytes from 192.168.1.3: icmp_seq=1 ttl=64 time=0.581 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=1.86 ms

--- 192.168.1.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.581/1.223/1.866/0.643 ms
bob:/home/bob# ping 192.168.1.2 -c2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=63 time=0.704 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=2.99 ms

--- 192.168.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.704/1.849/2.994/1.145 ms
bob:/home/bob#
```

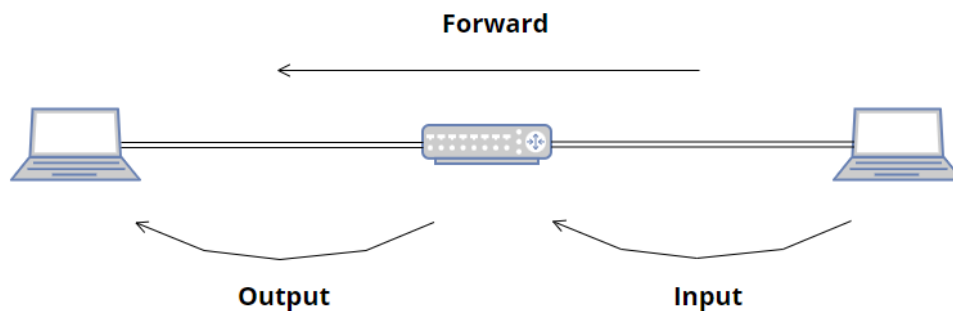
Como vemos desde Bob también es posible comunicarse con todas las demás direcciones (Las dos de Mallet y la de Alice).

En cuanto al ttl, vemos que cada vez que atraviesa el router se decrementa en una unidad. Esto sirve para que los paquetes que realicen muchos saltos tengan un ttl bajo o incluso 0 pudiendo así, descartarlos y evitar que entorpezcan el tráfico en la red.

6. Indica las diferencias entre las reglas de tipo INPUT, OUTPUT y FORWARD en iptables.

- » **INPUT:** es para filtrar paquetes que vienen hacia nuestra máquina.
- » **OUTPUT:** es para filtrar paquetes generados por nuestra máquina.
- » **FORWARD:** es para filtrar aquellos paquetes que llegan a nuestra máquina, pero no son para nosotros, es decir, que llegan para que nuestra máquina los reencamine.

Desde el punto de vista desde la máquina de Mallet se vería algo como esto (las flechas indican el paso de paquetes):



7. Configura una regla en el firewall del router para que sólo se puedan realizar peticiones de tipo ICMP(8) desde la red interna.

Para configurar las reglas del firewall nos valdremos del comando iptables. Primero añadiremos al comando el argumento -L para ver el listado de las reglas de una cadena actuales.

```
root@mallet:/home/mallet# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@mallet:/home/mallet#
```

Vemos como no hay ninguna regla en el firewall. Ahora para añadir una nueva hay que ejecutar el comando que vemos en la siguiente imagen. El argumento “-A” indica que queremos añadir una nueva regla en una determinada cadena. FORWARD sirve, como hemos dicho antes, para filtrar los paquetes que van de una interfaz de red a otra. “-s ipv4” indica la dirección IP de origen de los paquetes, “-d ipv4” la de destino. “-p protocolo” indica el protocolo de red que queremos seguir.”—icmp-type 8” se utiliza para restringir las peticiones de tipo ICMP(8). Para acabar tenemos el argumento “-j DROP”, que indica que queremos desechar este tipo de paquetes.

Con “iptables -L” vemos si se ha introducido correctamente la regla.

```
root@mallet:/home/mallet# iptables -A FORWARD -s 1.0.0.0/8 -d 192.168.1.0/24 -p
icmp --icmp-type 8 -j DROP
root@mallet:/home/mallet# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                               destination
DROP       icmp -- 1.0.0.0/8                          192.168.1.0/24      icmp echo-request

Chain OUTPUT (policy ACCEPT)
target     prot opt source                               destination
```

Ahora comprobamos si la regla funciona enviando paquetes desde bob a Alice.

```
bob:/home/bob# ping 192.168.1.2 -c3
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.

--- 192.168.1.2 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2007ms
```

La regla funciona correctamente ya que esta denegando el paso de paquetes como podemos ver en los resultados de los ping.

8. Configura una regla en el firewall para que sólo la máquina con IP 192.168.1.3 sea quién pueda establecer una conexión por ssh a la máquina 192.168.1.2.

Para conseguir esto añadiremos una regla al firewall para que rechace todas las conexiones ssh. Estas conexiones se hacen mediante el puerto 22 tcp por lo que rechazaremos cualquier conexión de este tipo. Después de añadir la regla comprobamos si ha sido añadida correctamente.


```

root@mallet: /home/mallet
File Edit View Terminal Help
root@mallet:/home/mallet# iptables -A FORWARD -s 0.0.0.0/0 -p tcp --dport 22 -j DROP
root@mallet:/home/mallet# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination
DROP      icmp -- 1.0.0.0/8              192.168.1.0/24      icmp echo-request
DROP      tcp  -- anywhere              anywhere            tcp dpt:ssh

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
root@mallet:/home/mallet#

```

Para comprobar que funciona intentaremos establecer una conexión ssh desde Bob a Alice.

```
bob:/home/bob# ssh alice@192.168.1.2
```

Como se puede observar la conexión no se puede establecer, sin embargo, ahora vamos a ver si podemos establecer una conexión desde la red interna. Es decir, vamos a conectarnos desde Mallet a Alice.

```

root@mallet: /home/mallet
File Edit View Terminal Help
root@mallet:/home/mallet# ssh alice@192.168.1.2
The authenticity of host '192.168.1.2 (192.168.1.2)' can't be established.
RSA key fingerprint is e4:60:c7:8e:e8:2b:a5:7e:ec:ca:36:38:59:fe:40:95.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.2' (RSA) to the list of known hosts.
alice@192.168.1.2's password:
Linux alice 2.6.32-28-generic #55-Ubuntu SMP Mon Jan 10 21:21:01 UTC 2011 i686 GNU/Linux
Ubuntu 10.04.2 LTS

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/

New release 'precise' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Thu Sep 29 13:09:09 2022 from mallet.local
alice@alice:~$ exit
logout
Connection to 192.168.1.2 closed.
root@mallet:/home/mallet#

```

Vemos que en esta ocasión si que establecemos conexión de manera exitosa, es decir, la regla implementada funciona correctamente.