Practica 3

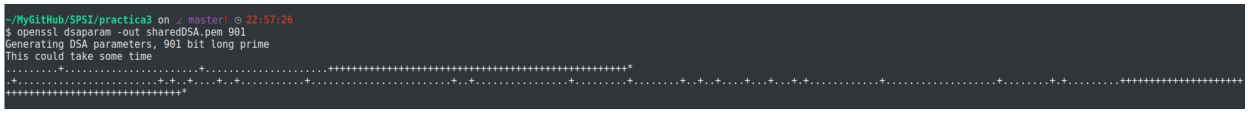
1.Generar un archivo sharedDSA.pem que contenga los parámetros. Mostrar los valores:

Utilizaremos el comando "dsaparam" que nos permite generar y manipular parámetros asociados al algoritmo DSA: primos ***p___ y q_ y el generador ___g***.

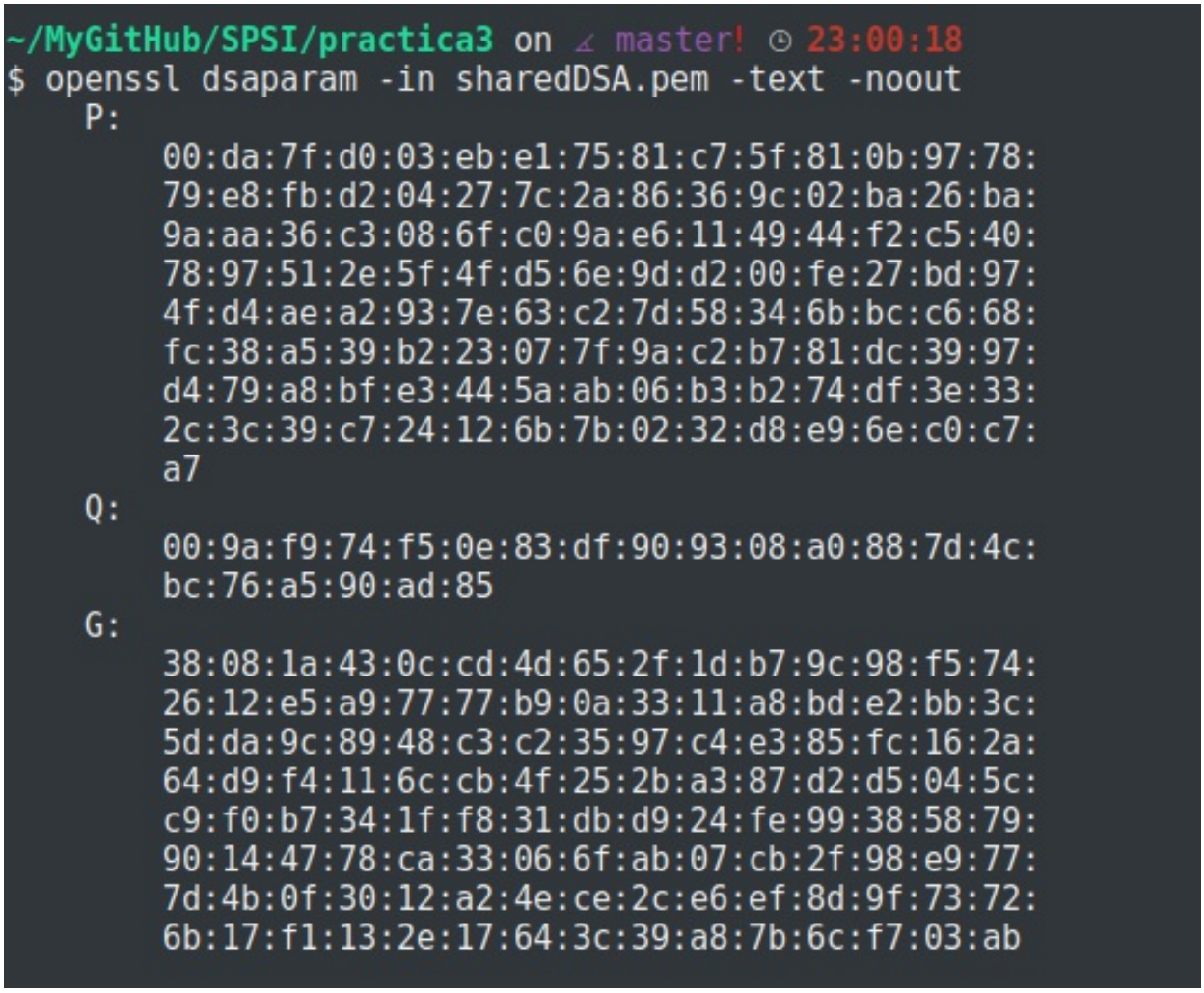
```
openssl dsaparam -out sharedDSA.pem 901
```

Así se generará el resultado con un tamaño de 901 bits.

La consola nos mostrará algo como esto:



Los valores obtenidos en hexadecimal tras generar el archivo son estos:



2.Generar dos parejas de claves para los parámetros anteriores. Las claves se almacenarán en los archivos DSAkey.pem y DSAkey.pem. No es necesario protegerlas con contraseña.

Para generar la pareja de claves necesitamos del archivo anterior (sharedDSA.pem) el cual actuará como parámetro. Esta vez utilizaremos el comando de openssl ***gedsa***.

En primer lugar generamos el par para Javier:

```
openssl gendsa -out JavierDSAkey.pem sharedDSA.pem
```

Por último, las generamos para Galera:

```
openssl gendsa -out GaleraDSAkey.pem sharedDSA.pem
```

```
~/MyGitHub/SPSI/practica3 on ⊿ master! ⏲ 23:17:31
$ openssl gendsa -out JavierDSAkey.pem sharedDSA.pem
Generating DSA key, 960 bits

~/MyGitHub/SPSI/practica3 on ⊿ master! ⏲ 23:17:47
$ openssl gendsa -out GaleraDSAkey.pem sharedDSA.pem
Generating DSA key, 960 bits
```

Ya tenemos generadas las dos parejas de claves.

3."Extraer" la clave privada contenida en el archivo DSAkey.pem a otro archivo que tenga por nombre DSApriv.pem. Este archivo deberá estar protegido por contraseña. Mostrad sus valores. Haced lo mismo para el archivo DSAkey.pem

Para realizarlo utilizaremos el comando de openssl ***dsa***. El cual nos permitirá sustraer la clave public y/o privada.Además utilizaremos como cifrado simétrico AES-128 y como contraseña: 0123456789

De esta forma extraeremos y cifraremos la clave privada de Javier:

```
openssl dsa -in JavierDSAkey.pem -out JavierDSApriv.pem -aes128
```

```
~/MyGitHub/SPSI/practica3 on ⊿ master! ⏲ 23:34:53
$ openssl dsa -in JavierDSAkey.pem -out JavierDSApriv.pem -aes128
read DSA key
writing DSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Del mismo modo para Galera:

```
~/MyGitHub/SPSI/practica3 on ⊿ master! ⏲ 23:35:05
$ openssl dsa -in GaleraDSAkey.pem -out GaleraDSApriv.pem -aes128
read DSA key
writing DSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

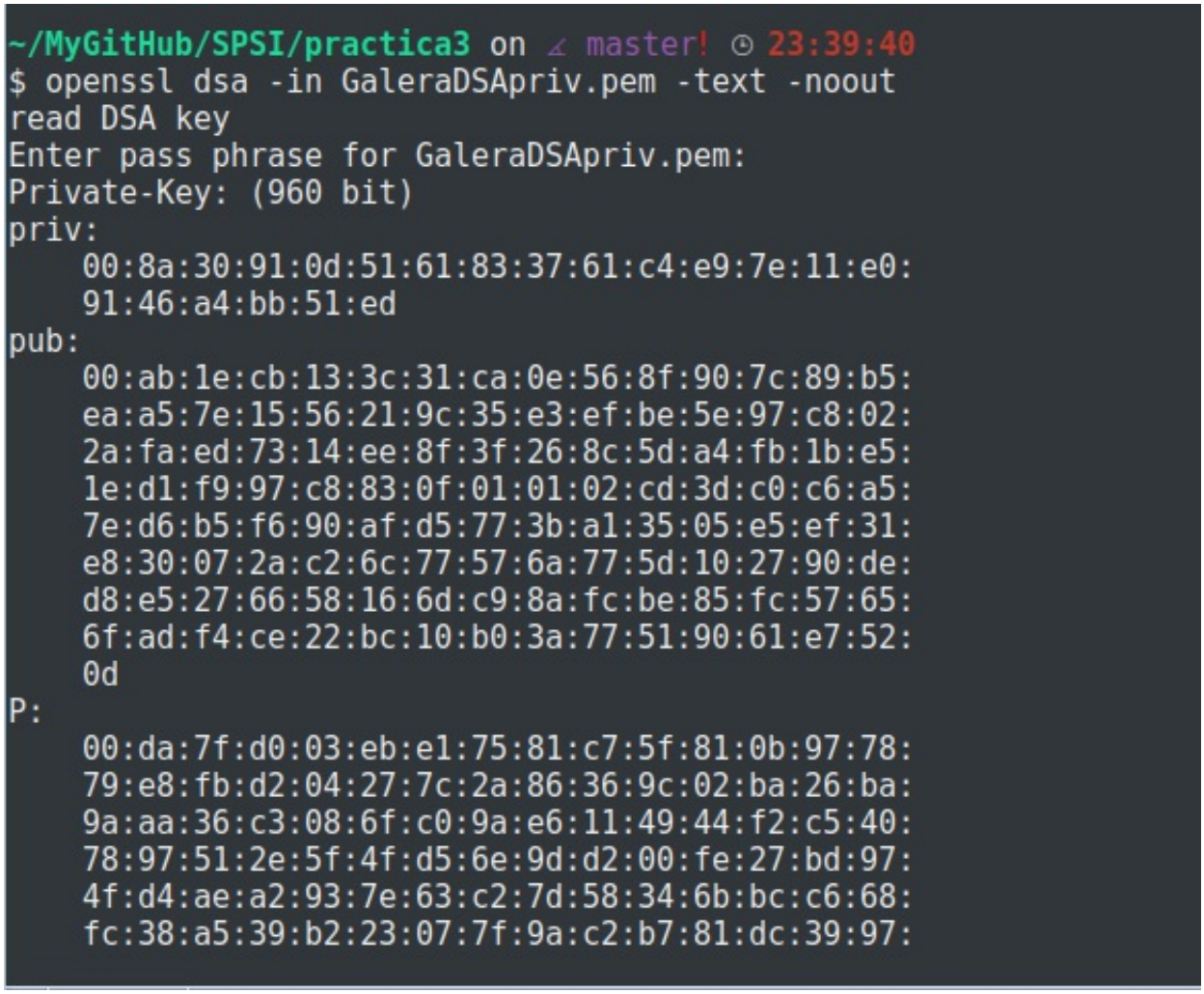
Por último mostraremos el resultado de dichas claves, a través del comando:

```
openssl dsa -in <nombre>DSApriv.pem -text -noout
```

De ***Javier***:

```
~/MyGitHub/SPSI/practica3 on ⊿ master! ⏲ 23:35:31
$ openssl dsa -in JavierDSApriv.pem -text -noout
read DSA key
Enter pass phrase for JavierDSApriv.pem:
Private-Key: (960 bit)
priv:
    48:5e:ba:e9:42:ce:4d:55:0c:82:e8:22:87:ed:54:
    32:3c:e7:8c:6b
pub:
    39:cf:34:a3:e7:58:49:da:c8:6e:31:52:69:08:07:
    59:1c:1a:7e:7d:ce:df:35:1c:0a:8c:f1:6b:ba:05:
    ba:1f:1d:6e:c4:09:36:cc:7a:e4:f1:00:f2:d9:d9:
    4b:69:88:23:31:b7:82:f2:39:4a:23:47:b6:ec:60:
    82:ed:53:f1:d5:11:fd:40:98:ed:85:fd:75:ab:35:
    a8:19:05:f4:4a:dd:33:c5:57:87:4a:e8:15:d1:f5:
    1a:c4:7b:9c:11:db:cb:e9:db:97:9a:49:61:a6:74:
    19:1c:ff:eb:d8:bd:57:18:3e:90:ed:97:52:58:af
P:
    00:da:7f:d0:03:eb:e1:75:81:c7:5f:81:0b:97:78:
    79:e8:fb:d2:04:27:7c:2a:86:36:9c:02:ba:26:ba:
    9a:aa:36:c3:08:6f:c0:9a:e6:11:49:44:f2:c5:40:
    78:97:51:2e:5f:4f:d5:6e:9d:d2:00:fe:27:bd:97:
    4f:d4:ae:a2:93:7e:63:c2:7d:58:34:6b:bc:c6:68:
    fc:38:a5:39:b2:23:07:7f:9a:c2:b7:81:dc:39:97:
    d4:79:a8:bf:e3:44:5a:ab:06:b3:b2:74:df:3e:33:
```

De ***Galera***:



4.Extraer en DSApub.pem la clave pública contenida en el archivo DSAkey.pem. De nuevo DSApub.pem no debe estar cifrado ni protegido. Mostrad sus valores. Lo mismo para el archivo DSAkey.pem

Al igual que en el apartado anterior, utilizaremos el comando de openssl ***dsa***.

Para la extracción de la clave pública utilizaremos el comando:

```
openssl dsa -in <nombre>DSAkey.pem -out <nombre>DSApub.pem -pubout
```

Tanto para Javier como para Galera:

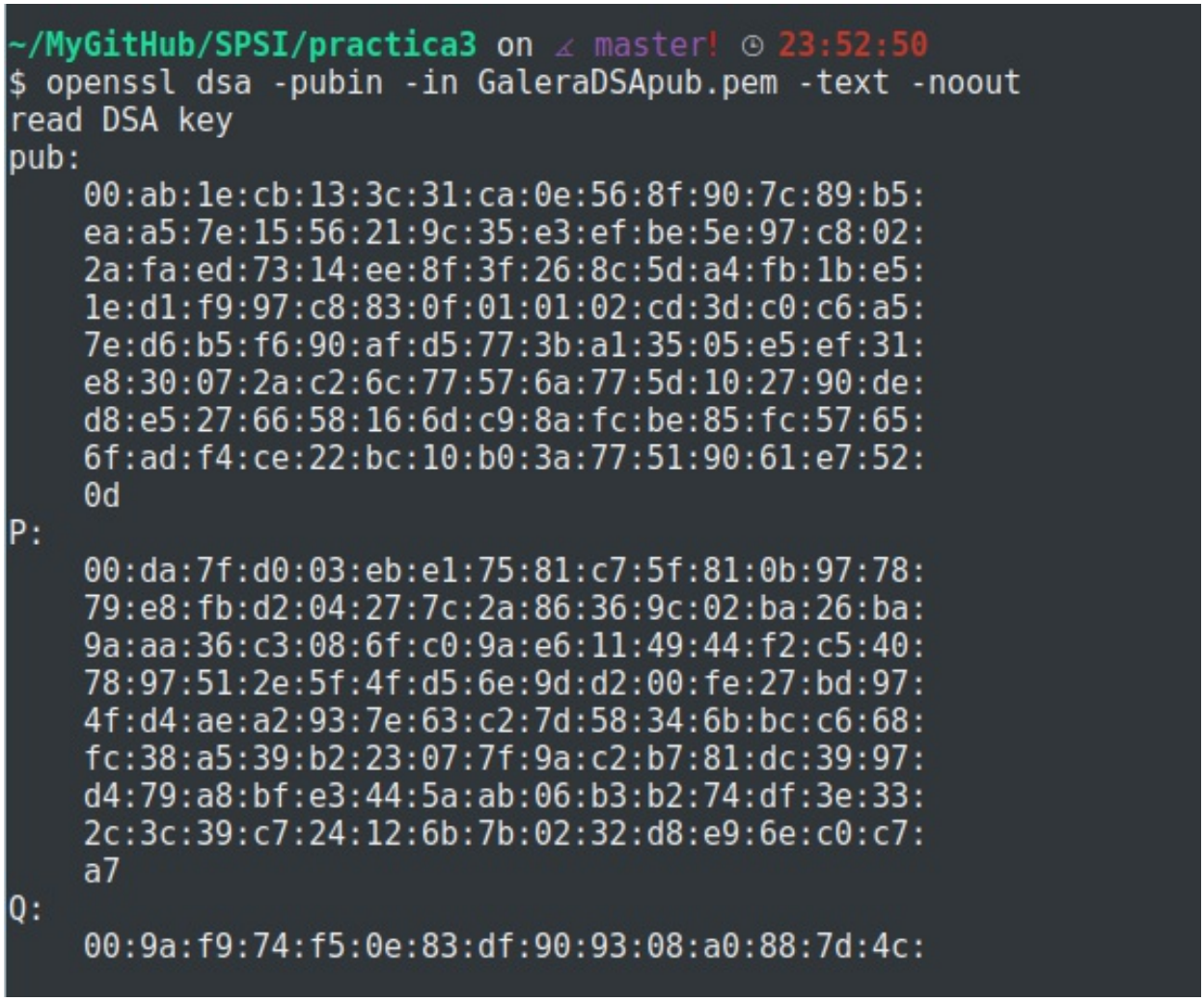
```
~/MyGitHub/SPSI/practica3 on  master! 23:48:55
$ openssl dsa -in JavierDSAkey.pem -out JavierDSApub.pem -pubout
read DSA key
writing DSA key

~/MyGitHub/SPSI/practica3 on  master! 23:49:19
$ openssl dsa -in GaleraDSAkey.pem -out GaleraDSApub.pem -pubout
read DSA key
writing DSA key
```

Por último mostraremos los valores de las claves:

Para ***Javier***:

```
~/MyGitHub/SPSI/practica3 on  master! 23:52:28
$ openssl dsa -pubin -in JavierDSApub.pem -text -noout
read DSA key
pub:
    39:cf:34:a3:e7:58:49:da:c8:6e:31:52:69:08:07:
    59:1c:1a:7e:7d:ce:df:35:1c:0a:8c:f1:6b:ba:05:
    ba:1f:1d:6e:c4:09:36:cc:7a:e4:f1:00:f2:d9:d9:
    4b:69:88:23:31:b7:82:f2:39:4a:23:47:b6:ec:60:
    82:ed:53:f1:d5:11:fd:40:98:ed:85:fd:75:ab:35:
    a8:19:05:f4:4a:dd:33:c5:57:87:4a:e8:15:d1:f5:
    1a:c4:7b:9c:11:db:cb:e9:db:97:9a:49:61:a6:74:
    19:1c:ff:eb:d8:bd:57:18:3e:90:ed:97:52:58:af
P:
    00:da:7f:d0:03:eb:e1:75:81:c7:5f:81:0b:97:78:
    79:e8:fb:d2:04:27:7c:2a:86:36:9c:02:ba:26:ba:
    9a:aa:36:c3:08:6f:c0:9a:e6:11:49:44:f2:c5:40:
    78:97:51:2e:5f:4f:d5:6e:9d:d2:00:fe:27:bd:97:
    4f:d4:ae:a2:93:7e:63:c2:7d:58:34:6b:bc:c6:68:
    fc:38:a5:39:b2:23:07:7f:9a:c2:b7:81:dc:39:97:
    d4:79:a8:bf:e3:44:5a:ab:06:b3:b2:74:df:3e:33:
    2c:3c:39:c7:24:12:6b:7b:02:32:d8:e9:6e:c0:c7:
    a7
Q:
    00:9a:f9:74:f5:0e:83:df:90:93:08:a0:88:7d:4c:
    bc:76:a5:90:ad:85
```

Para ***Galera***:



5. Coger un archivo que actuará como entrada, de al menos 128 bytes. En adelante, dicho archivo será denominado como "message".

Para crear el archivo utilizaremos:

```
dd if=/dev/zero of=message count=1024 bs=1
```

El parámetro count recibe el número de bits del archivo que se creará.
Hemos introducido 1024, pues 128 bytes * 8 bits/byte = 1024 bits.
El archivo será una retahíla de ceros, puesto que solo hemos creado el archivo, no lo hemos modificado nada.

6.Firmar directamente el archivo message

empleando el comando openssl pkeyutl sin calcular los valores hash, la firma deberá almacenarse en un archivo llamado message.sign. Mostrad el archivo con la firma.

Para firmar el archivo utilizaremos la clave privada de Javier con el comando:

```
openssl pkeyutl -sign -in message -inkey <nombre>DSApriv.pem -out message.bin
```

```
~/MyGitHub/SPSI/practica3 on  master! 11:07:56
$ openssl pkeyutl -sign -in message -inkey JavierDSApriv.pem -out message.sign
Enter pass phrase for JavierDSApriv.pem:
```

Ahora mostraremos el original y el resultado:

#####Original

```
~/MyGitHub/SPSI/practica3 on  master! 11:11:13
$ xxd message
00000000: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000130: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000160: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

#####Firmado

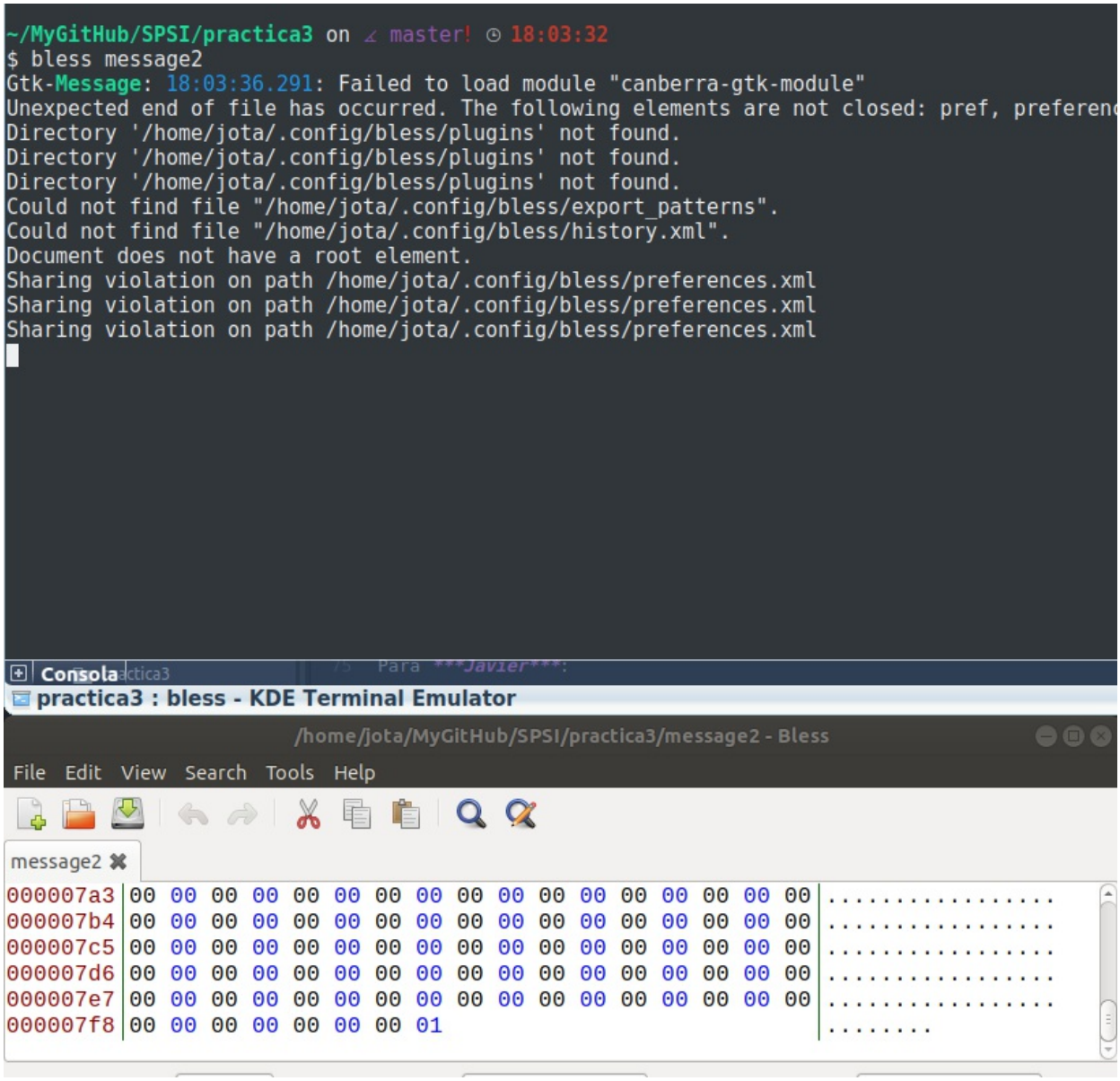
```
~/MyGitHub/SPSI/practica3 on  master! 11:11:19
$ xxd message.sign
00000000: 302c 0214 5269 89f3 4921 baa4 b2c3 d64e  0,..Ri..I!.....N
00000010: 6464 150e 34d8 815e 0214 7c3a 9d09 5af8  dd..4..^..|:..Z.
00000020: 475f 74fe 8b44 dd41 ae6a f4d4 c719       G_t..D.A.j....
```

7.Construir un archivo message2 diferente de message tal que la verificación de la firma message.sign sea correcta con respecto al archivo message2.

Para ello crearemos un archivo message2, en este caso será del doble de tamaño que message(2048 bits):

```
~/MyGitHub/SPSI/practica3 on  master! 18:02:00
$ dd if=/dev/zero of=message2 count=2048 bs=1
2048+0 registros leídos
2048+0 registros escritos
2048 bytes (2,0 kB, 2,0 KiB) copied, 0,00811345 s, 252 kB/s
```

Ahora introduciremos al final, en los últimos bits, un cambio para comprobar que aún asi verificará la firma:



Ahora utilizando este comando comprobaremos la verificación:

```
openssl pkeyutl -verify -in message2 -sigfile message.sign -pubin -inkey <nombre>DS
```

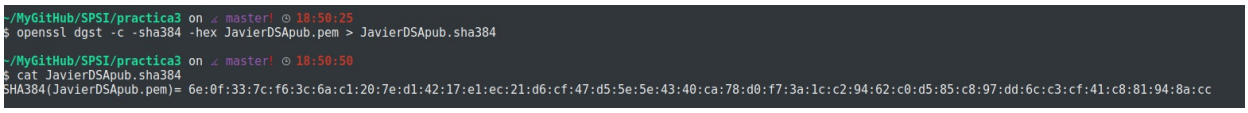
```
~/MyGitHub/SPSI/practica3 on master! 18:14:28
$ openssl pkeyutl -verify -in message2 -sigfile message.sign -pubin -inkey JavierDSApub.pem
Signature Verified Successfully
```

Esto tiene una explicación "simple", resulta que ***pkeyutl*** tiene un **pequeño *BUG***. Si firmamos un archivo con "message" de 128 bits(a 0 todos los bits o bien a un contenido "X") y creamos un archivo "message2" de 129 bits o más, donde los primermos 128 bits coinciden con los de "message", resulta que verifica la firma. Esto es debido a que trunca (algo así como "recortar") el archivo "message2" a partir de los 128 primeros bits.

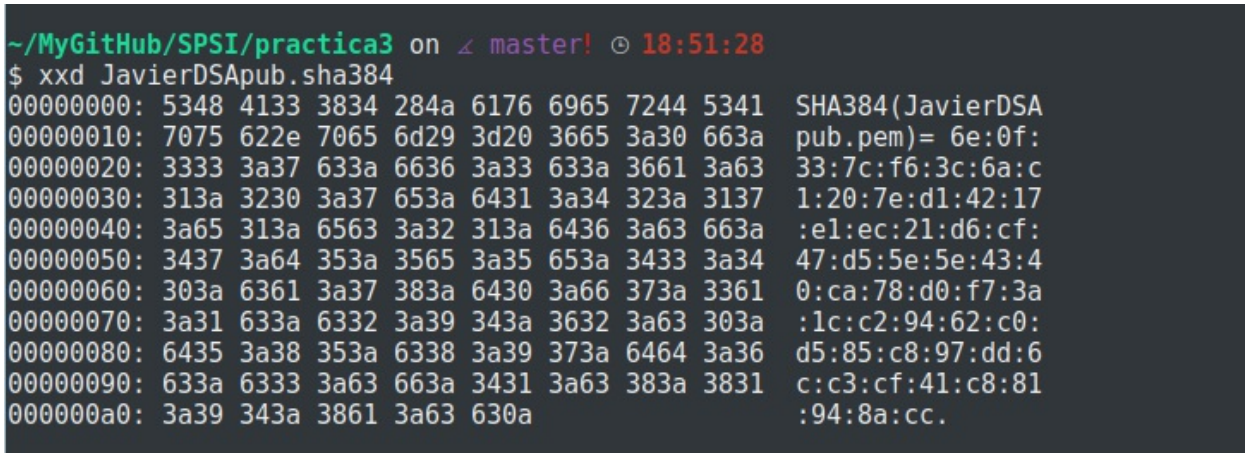
8.Calcular el valor hash del archivo con la clave pública DSApub.pem usando sha384 con salida hexadecimal con bloques de dos caracteres separados por dos puntos. Mostad los valores por salida estándar y guardarlos en DSApub.sha384.

Para ello utilizaremos el siguiente comando:

```
openssl dgst -c -sha384 -hex <nombre>DSApub.pem > <nombre>DSApub.sha384
```



Además mostraremos también el resultado:



Aquí podemos comprobar que se ha aplicado ***sha384***.

Además se puede mostrar por pantalla el resultado directamente con este comando:

```
openssl dgst -c -sha384 -hex <nombre>DSApub.pem && xdd <nombre>DSApub.pem
```

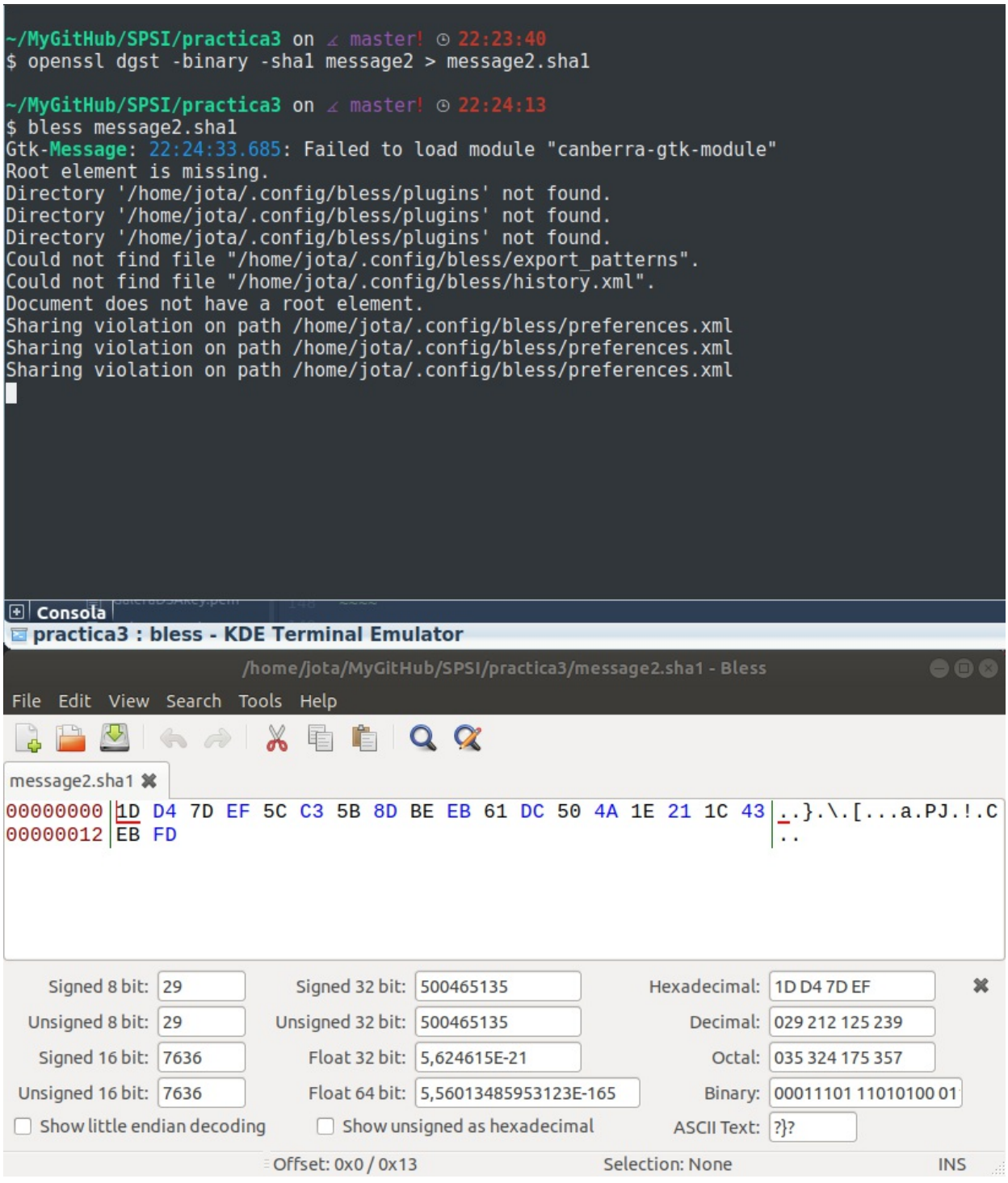
9. Calcular el valor hash de message2 usando una función hash de 160 bits con salida binaria. Guardad el hash en message2. y mostrar su

contenido.

En este caso utilizaremos SHA1, con ello el comando utilizado será:

```
openssl dgst -binary -sha1 message2 > message2.ripemd160
```

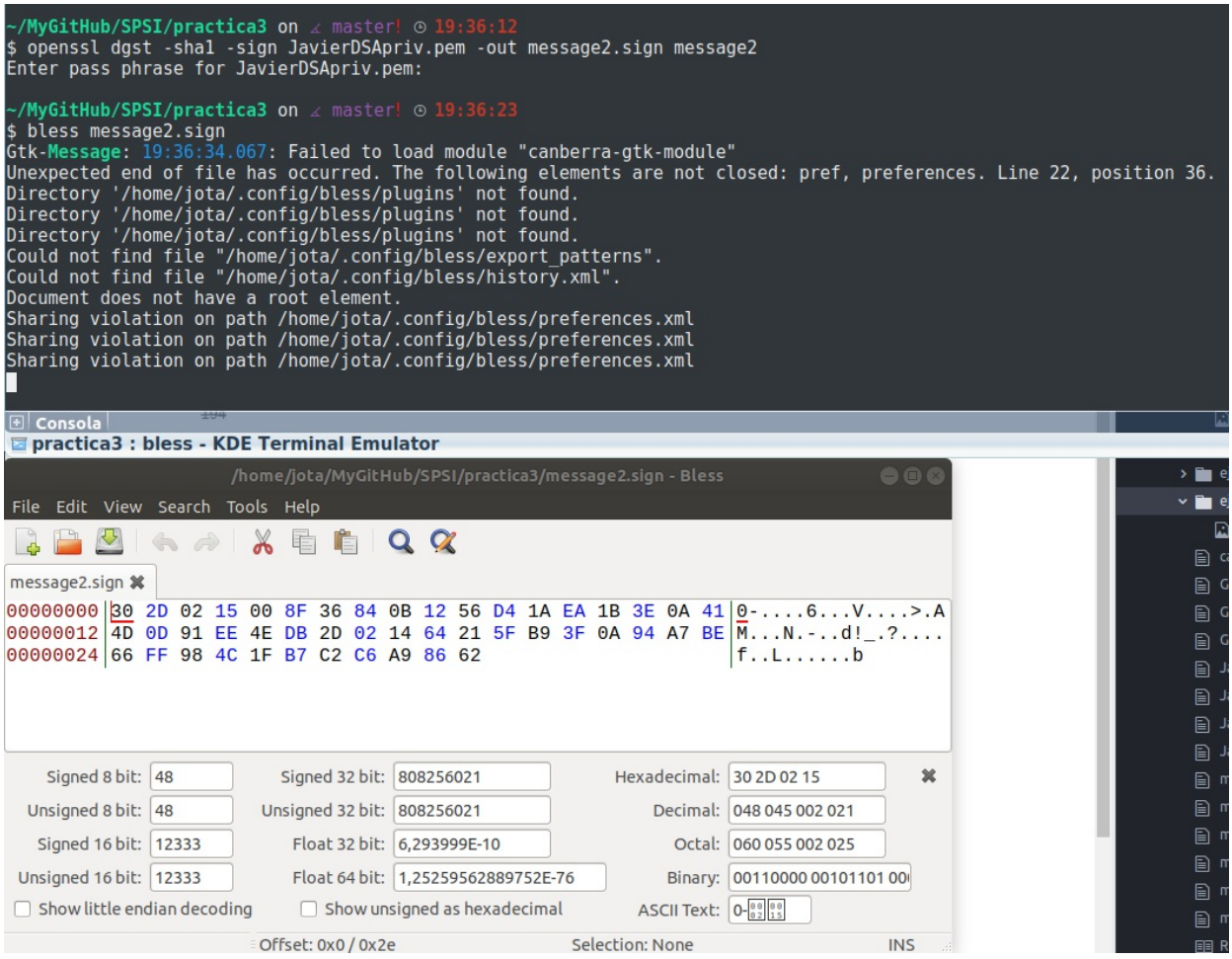
Aquí podemos observar el contenido del archivo y la creación de este:



10. Firmad el archivo message2 mediante el comando openssl dgst y la funcion hash del punto anterior. La firma deberá almacenarse en un archivo llamando message2.sign

Firmaremos el archivo utilizando el comando:

```
openssl dgst -sha1 -sign <nombre>DSApriv.pem -out message2.sign message2
```



11. Verificad la firma message2.sign con los archivos message y message2 empleando el comando openssl dgst.

Podemos verificar la firma con dgst de Openssl mediante:

```
openssl dgst -sha1 -verify JavierDSApub.pem -signature message2.sign message2
```

```
~/MyGitHub/SPSI/practica3 on  master! 19:37:18
$ openssl dgst -sha1 -verify JavierDSApub.pem -signature message2.sign message2
Verified OK
```

12. Verificad que message2.sign es una firma correcta para message2 pero empleando el comando openssl pkeyutl

Utilizando el siguiente comando podremos comprobar que se verifica:

```
openssl pkeyutl -verify -in message2.sha1 -sigfile message2.sign -pubin
-inkey <nombre>DSApub.key
```

```
~/MyGitHub/SPSI/practica3 on  master! 22:26:09
$ openssl pkeyutl -verify -in message2.sha1 -sigfile message2.sign -pubin -inkey JavierDSApub.pem
Signature Verified Successfully
```

13. Generar el valor HMAC del archivo sharedDSA.pem con clave '12345' mostrándolo por pantalla.

Para ello emplearemos el comando utilizado anteriormente ***dgst*** y como dice el enunciado le añadiremos la clave '12345', tal que:

```
openssl dgst -hmac '12345' sharedDSA.pem
```

```
~/MyGitHub/SPSI/practica3 on  master! 0:58:23
$ openssl dgst -hmac '12345' sharedDSA.pem
HMAC-SHA1(sharedDSA.pem)= fa1ed61c00b464b51ff00a20630bf93afee6410d
```

14. Simular una ejecución completa del protocolo Estación a Estación. Para ello emplearemos como

claves para firmar/verificación las generadas en esta práctica, y para el protocolo DH emplearemos las claves asociadas a curvas elípticas de la práctica anterior junto con las de otro usuario simulado que se deben generar nuevamente. El algoritmo simétrico a utilizar en el protocolo estación a estación será AES-128 en modo CFB8.

1. Recopilamos las claves que vamos a utilizar a lo largo del ejercicio. Los participantes serán: Javier y Galera.
   Por tanto las claves que utilizaremos para firmar y verificar serán:

```
-Javier: JavierDSApub.pem/JavierDSApriv.pem

-Galera: GaleraDSApub.pem/GaleraDSApriv.pem
```

Para el protocolo DH utilizaré las claves de la práctica anterior. Añadiendo una nueva pareja de claves para Galera.

2. Mostramos la clave privada y pública asociada a las curvas elípticas de Javier:
   Nota: la pass de las claves privadas es: 0123456789

Por parte de Javier tenemos:

PRIVADA

```
~/MyGitHub/SPSI/practica3 on ◿ master! ⏲ 1:23:24
$ openssl ec -in JavierECpriv.pem -text -noout
read EC key
Enter PEM pass phrase:
Private-Key: (256 bit)
priv:
    00:9f:22:21:5f:fe:a7:48:c9:23:ba:0b:55:09:50:
    9e:f6:29:c4:9b:ee:b9:cc:c3:3f:51:89:95:92:81:
    f3:a4:71
pub:
    04:c8:b7:6c:c7:5c:78:1e:f9:de:9e:97:d2:c7:f4:
    5e:e8:21:f8:c7:7b:02:c3:2f:c4:03:6b:7f:fc:ff:
    e7:08:29:55:55:4e:66:c1:61:b0:f8:fa:28:d5:2d:
    4c:1e:5b:f2:97:21:6b:f5:91:4c:aa:4c:be:19:60:
    16:d6:49:db:2f
ASN1 OID: prime256v1
```

PUBLICA

```
~/MyGitHub/SPSI/practica3 on ◿ master! ⏲ 1:27:56
$ openssl ec -pubin -in JavierECpub.pem -text -noout
read EC key
Private-Key: (256 bit)
pub:
    04:c8:b7:6c:c7:5c:78:1e:f9:de:9e:97:d2:c7:f4:
    5e:e8:21:f8:c7:7b:02:c3:2f:c4:03:6b:7f:fc:ff:
    e7:08:29:55:55:4e:66:c1:61:b0:f8:fa:28:d5:2d:
    4c:1e:5b:f2:97:21:6b:f5:91:4c:aa:4c:be:19:60:
    16:d6:49:db:2f
ASN1 OID: prime256v1
```

Por parte de Galera tenemos:

PRIVADA

```
~/MyGitHub/SPSI/practica3 on ◿ master! ⏲ 1:26:25
$ openssl ec -in GaleraECpriv.pem -text -noout
read EC key
Enter PEM pass phrase:
Private-Key: (256 bit)
priv:
    5d:04:f6:1a:30:84:bf:0d:44:18:00:0b:48:eb:7b:
    8a:f9:d6:44:33:3c:e1:53:d6:ad:df:23:11:f7:80:
    a4:aa
pub:
    04:0c:9c:5e:42:b1:cf:4f:4a:a3:45:f2:ac:be:3d:
    a0:33:66:17:c4:85:af:98:b5:b5:13:b0:9c:a1:d8:
    8e:89:db:a0:3a:90:37:0d:c5:c4:36:bf:6d:b5:a4:
    de:8b:9a:50:02:dc:9b:bf:47:bf:1c:16:20:6a:08:
    c0:97:5a:5e:50
ASN1 OID: prime256v1
```

PUBLICA

```
~/MyGitHub/SPSI/practica3 on ◿ master! ⏲ 1:27:20
$ openssl ec -pubin -in GaleraECpub.pem -text -noout
read EC key
Private-Key: (256 bit)
pub:
    04:0c:9c:5e:42:b1:cf:4f:4a:a3:45:f2:ac:be:3d:
    a0:33:66:17:c4:85:af:98:b5:b5:13:b0:9c:a1:d8:
    8e:89:db:a0:3a:90:37:0d:c5:c4:36:bf:6d:b5:a4:
    de:8b:9a:50:02:dc:9b:bf:47:bf:1c:16:20:6a:08:
    c0:97:5a:5e:50
ASN1 OID: prime256v1
```

Para generar la pareja de claves públicas y privadas se ha seguido el mismo proceso que en la práctica anterior.

Ya tenemos todo listo para empezar la simulación.

3. Comenzaremos utilizando una diapositiva del profesor:

Estación a estación

Mejora del protocolo de Diffie-Hellman buscando resolver sus debilidades. Fijamos $g \in \mathbb{Z}_p^*$ como en el protocolo de Diffie-Hellman. Asumimos que tenemos disponible un esquema de firma digital. Cada usuario tiene una pareja de claves (s, v) privada-pública para firma y verificación. Las partes públicas son auténticas y accesibles.

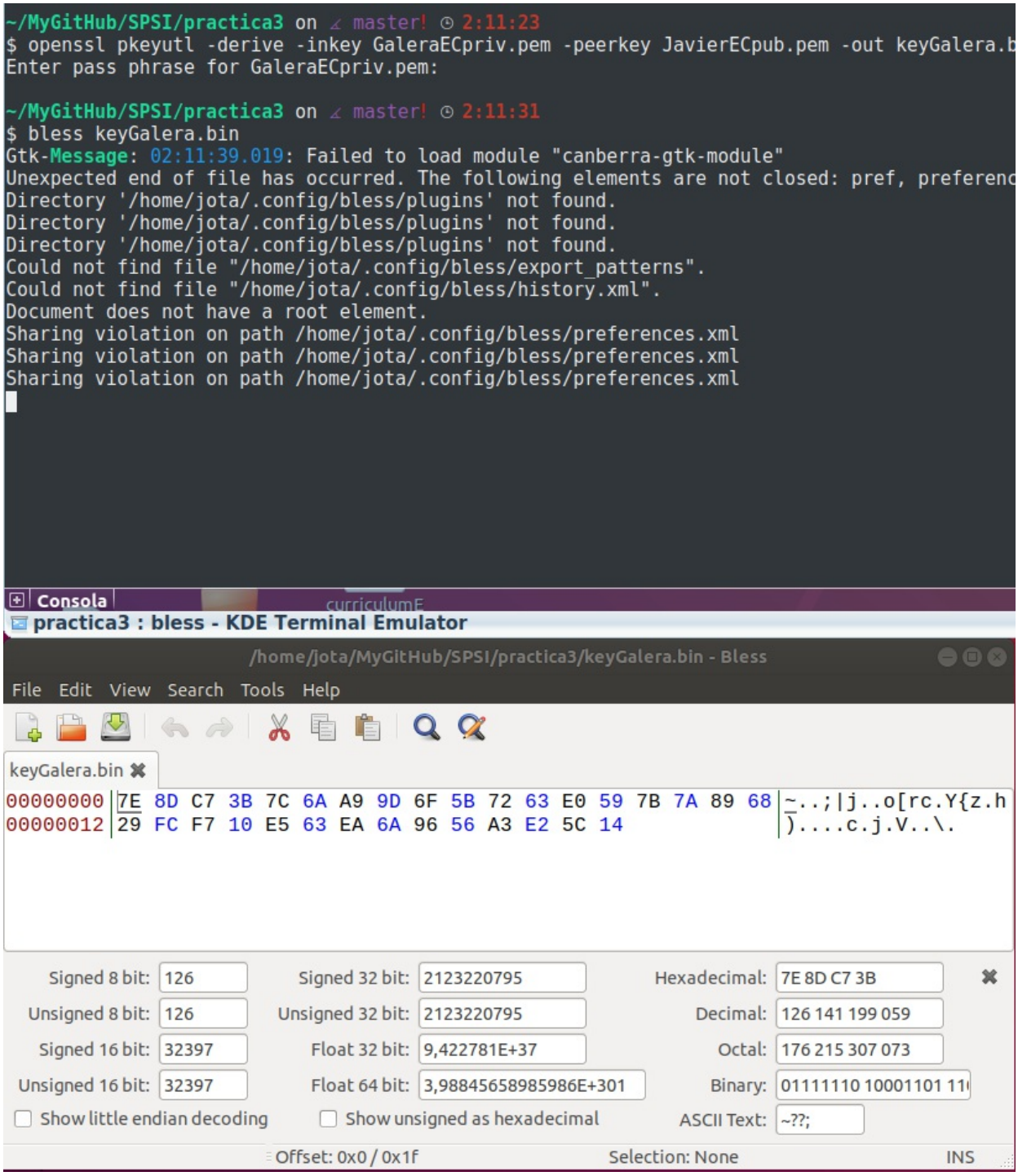
1. Alicia escoge aleatoriamente $1 \leq a \leq p-2$, calcula $c = g^a \bmod p$ y lo envía a Bernabé.
2. Bernabé escoge aleatoriamente $1 \leq b \leq p-2$, calcula $d = g^b \bmod p$ y $k = g^{ab} = (c)^b \bmod p$. Calcula $s = sgn_{s_B}(c||d)$. Envía $(d, e_k(s))$ a Alicia.
3. Alicia calcula $k = g^{ab} = d^a \bmod p$ y $s = d_k(e_k(s))$, verifica $vfy_{v_b}(c||d, s)$. Firma $t = sgn_{s_A}(d||c)$ y envía a Bernabé $e_k(t)$.
4. Bernabé calcula $t = d_k(e_k(t))$ y verifica $vfy_{v_A}(d||c, t)$.
5. A partir de este momento ambos pueden estar seguros de que la clave secreta k está compartida sólo por ellos dos.

4. Javier comparte su clave pública con Galera.

5. Galera genera su clave derivada usando el comando ***pkeyutl***, a partir de su clave privada y la pública de Javier tal que:

```
openssl pkeyutl -derive -inkey GaleraECpriv.pem -peerkey JavierECpub.pem -out keyGa
```

El resultado de esta operación es:



Una vez realizado esto, Galera concatena la clave pública de Javier con su clave pública, para ello utiliza:

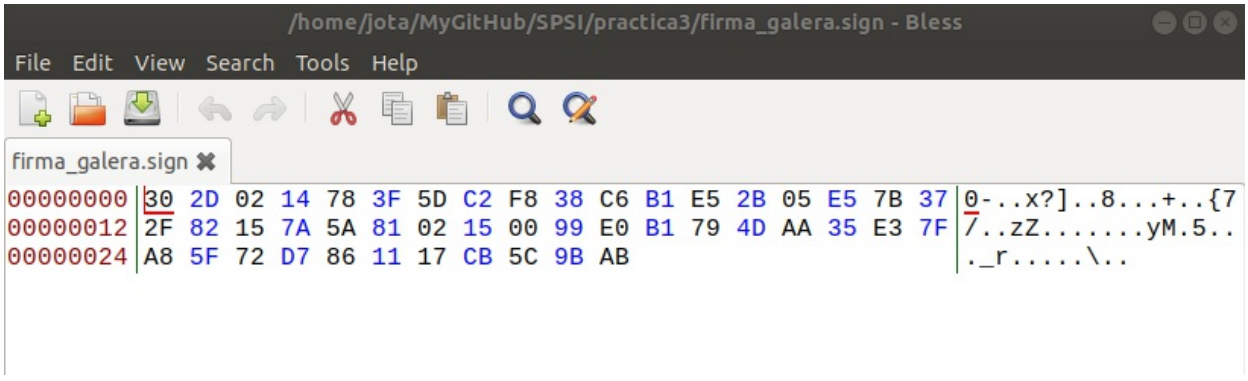
```
cat GaleraECpub.pem JavierECpub.pem > concatenacion.txt
```

```
~/MyGitHub/SPSI/practica3 on ⊿ master! ⏲ 2:12:37
$ cat GaleraECpub.pem JavierECpub.pem > concatenacion.txt

~/MyGitHub/SPSI/practica3 on ⊿ master! ⏲ 2:47:02
$ cat concatenacion
cat: concatenacion: No existe el archivo o el directorio

~/MyGitHub/SPSI/practica3 on ⊿ master! ⏲ 2:47:08
$ cat concatenacion.txt
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEDJxeQrHPT0qjRfKsvj2gM2YXxIWv
mLW1E7CcodiOidugOpA3DcXENr9ttaTei5pQAtybv0e/HBYgagjAl1peUA==
-----END PUBLIC KEY-----
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEyLdsx1x4HvnenpfSx/Re6CH4x3sC
wy/EA2t//P/nCClVVU5mwWGw+Poo1S1MHlvylyFr9ZFMqky+GWAW1knbLw==
-----END PUBLIC KEY-----
```

Galera ahora firmará este fichero(concatenacion.txt), cifrando el valor hash con su clave privada. El resultado será almacenado en un archivo llamado firma_galera.sign y utlizará el comando ***dgst*** tal que:

```
openssl dgst -sha256 -sign GaleraDSApriv.pem -out firma_galera.sign concatenacion.t
```

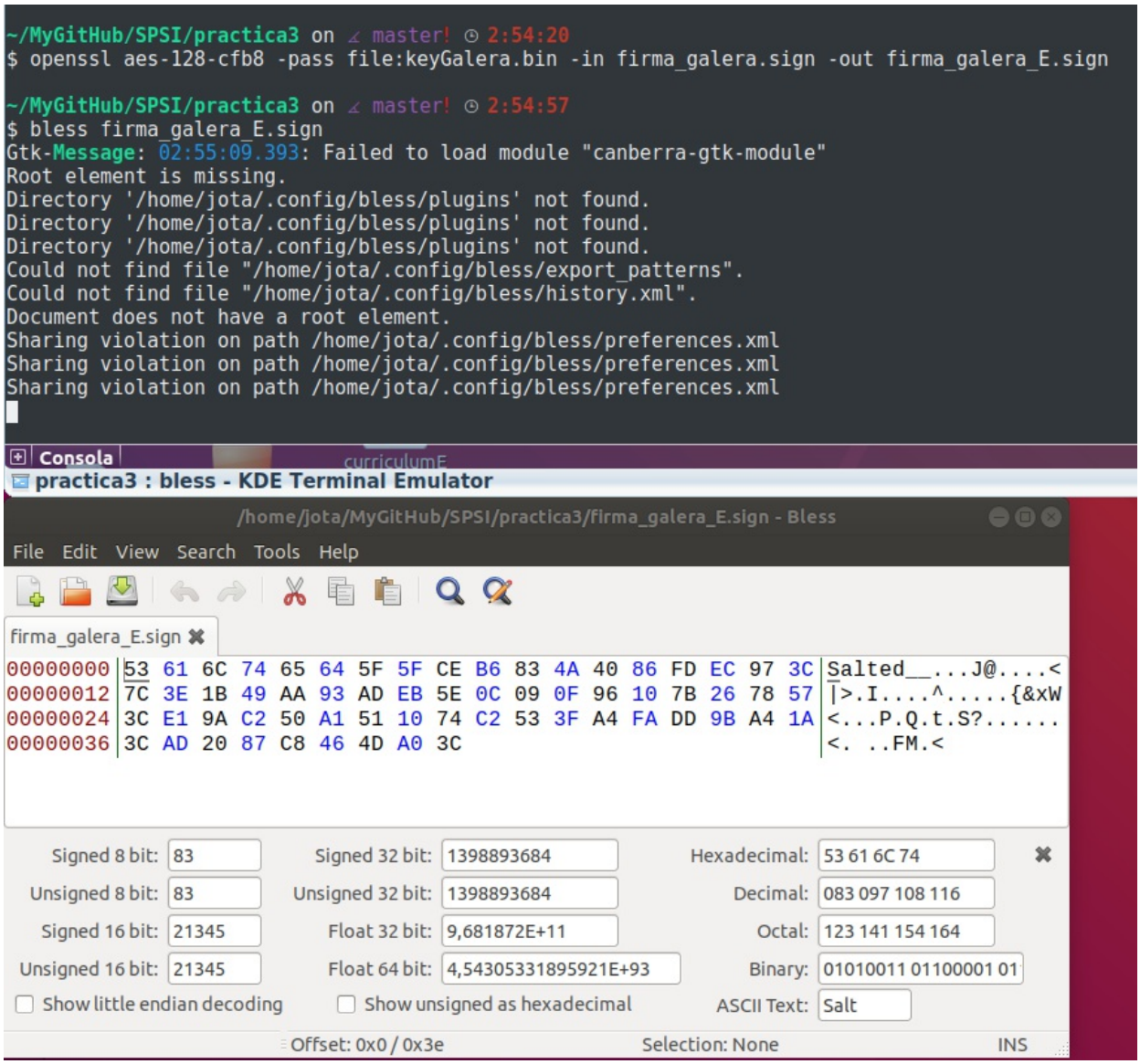
```
~/MyGitHub/SPSI/practica3 on ⊿ master! ⏲ 2:48:21
$ cat firma_galera.sign
0-x?]��8U�+�{7/�zZ����yM@5��_r◻\
```



Para cifrar el archivo se utilizará el cifrado simétrico indicado en el enunciado, AES-128 en modo CFB8:

```
openssl aes-128-cfb8 -pass file:keyGalera.bin -in firma_galera.sign -out firma_gale
```

Como resultado obtenemos:



Galera envía la firma encriptada y su clave pública a Javier.

6. Javier calculará su clave derivada como hizo Galera, simplemente ahora utilizarña su clave privada y la pública de Galera:

```
openssl pkeyutl -derive -inkey JavierECpriv.pem -peerkey GaleraECpub.pem -out keyJa
```