# Security: Asymmetric encryption

WSL1 on Windows 2019 server was used to complete the task.
The distribution running under WSL is Ubuntu 22.04.4 LTS
The following tutorial was used to enable the legacy provider in OpenSSL:
https://lindevs.com/enable-openssl-legacy-provider-on-ubuntu

# Task 1

Using the openssl genrsa command, write down the syntax of the command that will generate a pair of RSA keys (public and private) of length 2048 bits and store them in a file named rsa.key. Once the keys have been generated, check the contents of the rsa.key file using the following command: openssl rsa in rsa.key text

## Implementation

```
root@WSL:lab6_pliki> ls
'Lab6 - szyfry asymetryczne - zaoczne.pdf'  'Lab6 - szyfry asymetryczne
v2.pdf'
root@WSL:lab6_pliki> openssl genrsa -out rsa.key 2048
root@WSL:lab6_pliki> openssl rsa -in rsa.key -text
Private-Key: (2048 bit, 2 primes)
modulus:
00:97:ba:ad:c4:bb:34:77:52:2d:4c:f4:96:36:db:
4d:ef:0c:55:cb:5f:8c:22:53:7d:81:c8:10:20:cf:
c9:bd:6d:81:df:ad:0f:22:5c:aa:ef:e3:91:ec:3c:
8c:19:0a:46:38:7b:00:7e:00:e1:bf:2a:b8:71:bd:
f5:e2:a5:61:3b:e5:ed:ac:a4:b2:6e:54:8d:95:34:
dd:40:be:4b:e6:05:1b:c7:1e:d8:2a:8c:b9:19:fa:
bc:73:f8:23:13:d6:28:87:52:11:56:fe:05:d6:27:
bc:28:29:57:2d:f9:6f:7a:e4:ee:d2:d3:67:d3:ea:
b2:c2:6d:1b:cb:ba:4b:1a:b9:12:16:3f:0a:7d:e1:
83:89:12:9d:3b:03:e5:aa:1d:98:50:47:19:05:01:
cd:7f:e0:1f:f8:81:4a:b1:ce:99:e9:74:b1:81:9e:
ca:eb:a6:43:e6:b4:fe:f4:1c:48:15:c4:b3:82:68:
a5:e7:0f:b6:2f:82:69:d2:78:96:43:a3:72:bd:28:
44:24:f2:b7:c5:82:8b:71:77:e5:18:38:dc:f9:82:
7f:7d:38:70:d8:d1:3f:a6:66:3c:62:5a:bb:51:9c:
c6:5d:7c:38:ce:6d:77:7d:cc:80:b4:d8:77:79:46:
8b:8f:8a:57:f8:4a:0e:89:3b:42:f1:0f:0f:83:29:
28:7d
publicExponent: 65537 (0x10001)
privateExponent:
04:9e:80:73:73:4c:38:6f:b8:da:55:68:8c:a3:50:
de:e7:58:bb:4d:72:db:89:55:7c:fc:7a:ba:8c:fe:
```

```
34:1f:f5:2a:ba:94:31:df:1c:03:1b:ec:a6:d7:a7:
0a:37:72:5b:7e:29:cd:d6:e2:b3:2a:59:48:23:d8:
d6:fd:62:ea:eb:12:8e:70:20:7b:29:b9:7a:62:8c:
1e:7a:f7:48:e4:3c:96:c0:c7:cd:9a:7a:ce:21:80:
82:e5:4f:28:12:46:7f:41:79:2e:aa:a2:c1:da:1e:
d3:cd:82:63:c4:c0:11:ea:8d:f7:69:76:89:3b:3a:
c0:49:88:65:98:e1:22:c9:eb:3f:60:e3:44:6a:2d:
b3:d2:6b:5b:03:3c:69:fd:ee:21:fa:34:ac:37:8f:
6e:ee:f5:64:c5:6c:2d:70:a6:55:8d:3c:04:17:e5:
fb:60:ea:58:c9:87:42:81:39:db:25:d5:eb:b3:63:
d4:25:12:38:c9:a7:e9:a0:fd:4d:08:bf:30:3e:3a:
c0:e2:f0:9f:a8:cd:a0:bd:b1:6a:ae:bb:22:b1:7d:
e8:f2:f0:f0:06:69:7e:c1:a7:50:94:80:00:a1:fa:
bf:f9:14:51:73:a9:ce:0f:b8:d7:8b:b3:32:c8:f2:
8b:60:30:0c:2c:9c:7f:41:a2:d0:34:67:81:1e:7d:
61
prime1:
00:c7:05:67:36:47:db:b0:bc:62:e1:a4:6c:c7:b2:
a2:db:32:61:16:22:46:63:20:46:3c:00:f1:81:db:
fd:46:e3:ee:9d:5c:15:1b:45:2a:10:52:e1:8f:1a:
ee:82:ca:98:70:5b:1b:77:8d:ed:67:7c:12:dd:9e:
e6:20:b8:d0:58:06:04:6c:24:33:dd:e0:ea:91:c8:
32:3f:b6:24:7a:e4:42:74:ea:ca:d6:ce:86:6b:a9:
e6:bd:9b:b4:06:29:25:fe:fb:b9:08:2c:33:a4:91:
52:10:b2:c4:29:12:c1:60:7e:1f:8d:9b:46:5f:ab:
c0:4d:07:5d:54:74:f6:e4:11
prime2:
00:c3:2b:2c:0b:bb:00:70:d5:b1:ff:89:83:27:22:
0e:3c:8f:db:ca:96:be:1c:b4:31:57:08:1d:a8:c6:
7c:43:26:41:61:a0:10:46:14:6a:d4:4d:76:bf:8b:
c9:ca:ce:47:fd:69:8e:1f:a1:7e:0b:7c:0a:f8:39:
30:02:f5:56:56:46:21:78:70:95:f0:b8:2a:3b:b6:
bf:07:4c:b7:ea:92:66:7a:71:d0:4e:e0:4f:b8:ff:
89:79:0e:be:12:24:fb:04:76:ad:72:63:7c:10:29:
b8:b3:a2:cd:bf:45:c2:d2:5d:42:ea:d2:76:3d:f2:
a7:e1:ae:83:f0:9f:35:79:ad
exponent1:
56:80:a5:b4:b5:14:ea:c3:66:e3:16:39:65:ae:e2:
14:0d:d8:9e:eb:34:d6:be:df:68:13:2c:e5:39:7d:
e3:83:0c:62:7e:f4:79:8e:dd:52:8d:03:96:d4:a8:
d8:56:40:5d:8d:6b:e9:d1:d9:5f:02:51:9c:dc:04:
c0:32:33:f4:1e:61:6b:b3:6c:8b:c2:c7:91:a7:87:
17:a4:0e:45:0f:d3:46:c8:0a:73:c4:fe:2d:eb:81:
eb:e5:b2:20:7f:60:11:6c:cd:a6:9c:b7:b1:8e:ac:
31:af:d1:8c:dc:a7:0a:4e:a7:5e:40:20:1d:53:24:
59:16:9b:5d:2b:1d:68:51
exponent2:
00:93:8a:a7:99:96:9d:16:af:f6:08:40:64:ad:c8:
c9:0a:6f:96:b0:e8:94:80:0e:98:de:3a:7c:71:82:
db:9d:c7:e5:d5:6f:b3:dd:c6:c1:9c:0d:93:9f:ea:
85:7a:93:c7:48:83:eb:b5:6d:d8:63:c6:3a:64:bc:
```

```
96:d7:25:b1:47:37:6d:46:27:96:b2:8c:07:21:6d:
5d:ac:9f:0a:ca:82:32:86:8e:53:f3:cd:0f:77:c3:
76:a1:d5:cb:4a:dc:3a:07:7d:d0:51:91:73:d0:a4:
39:c5:53:66:47:1a:ed:d5:2a:3b:d4:59:96:99:ee:
3f:8c:14:ea:19:95:c2:4f:61
coefficient:
14:b8:7d:f9:65:90:6a:65:37:6e:41:04:d8:34:c6:
f3:91:0f:d6:d1:c8:a5:2f:70:28:b0:bb:38:46:d9:
3b:dd:9c:e9:71:e0:38:64:00:2e:e6:23:ff:7e:e4:
54:fc:33:45:ba:57:c2:36:19:34:d1:1d:47:f2:fc:
cb:f7:42:0f:fc:15:ca:68:48:1e:65:70:c1:fa:a9:
32:81:3f:79:45:dd:5f:5c:55:c0:ed:90:e6:9b:6e:
80:a6:47:27:12:e1:e1:80:80:5b:55:75:ac:19:bd:
0f:69:e4:95:3f:b9:f0:7c:d8:df:8c:ff:ba:21:f7:
e1:b5:26:df:a6:a9:80:9a
writing RSA key
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCXuq3EuzR3Ui1M
9JY2203vDFXLX4wiU32ByBAgz8m9bYHfrQ8iXKrv45HsPIwZCkY4ewB+AOG/Krhx
vfXipWE75e2spLJuVI2VNN1AvkvmBRvHHtgqjLkZ+rxz+CMT1iiHUhFW/gXWJ7wo
KVct+W965O7S02fT6rLCbRvLuksauRIWPwp94YOJEp07A+WqHZhQRxkFAc1/4B/4
gUqxzpnpdLGBnsrrpkPmtP70HEgVxLOCaKXnD7YvgmnSeJZDo3K9KEQk8rfFgotx
d+UYONz5gn99OHDY0T+mZjxiWrtRnMZdfDjObXd9zIC02Hd5RouPilf4Sg6JO0Lx
Dw+DKSh9AgMBAAECggEABJ6Ac3NMOG+42lVojKNQ3udYu01y24lVfPx6uoz+NB/1
KrqUMd8cAxvsptenCjdyW34pzdbisypZSCPY1v1i6usSjnAgeym5emKMHnr3SOQ8
lsDHzZp6ziGAguVPKBJGf0F5Lqqiwdoe082CY8TAEeqN92l2iTs6wEmIZZjhIsnr
P2DjRGots9JrWwM8af3uIfo0rDePbu71ZMVsLXCmVY08BBfl+2DqWMmHQoE52yXV
67Nj1CUSOMmn6aD9TQi/MD46wOLwn6jNoL2xaq67IrF96PLw8AZpfsGnUJSAAKH6
v/kUUXOpzg+414uzMsjyi2AwDCycf0Gi0DRngR59YQKBgQDHBWc2R9uwvGLhpGzH
sqLbMmEWIkZjIEY8APGB2/1G4+6dXBUbRSoQUuGPGu6CyphwWxt3je1nfBLdnuYg
uNBYBgRsJDPd4OqRyDI/tiR65EJ06srWzoZrqea9m7QGKSX++7kILDOkkVIQssQp
EsFgfh+Nm0Zfq8BNB11UdPbkEQKBgQDDKywLuwBw1bH/iYMnIg48j9vKlr4ctDFX
CB2oxnxDJkFhoBBGFGrUTXa/i8nKzkf9aY4foX4LfAr4OTAC9VZWRiF4cJXwuCo7
tr8HTLfqkmZ6cdBO4E+4/4l5Dr4SJPsEdq1yY3wQKbizos2/RcLSXULq0nY98qfh
roPwnzV5rQKBgFaApbS1FOrDZuMWOWWu4hQN2J7rNNa+32gTLOU5feODDGJ+9HmO
3VKNA5bUqNhWQF2Na+nR2V8CUZzcBMAyM/QeYWuzbIvCx5GnhxekDkUP00bICnPE
/i3rgevlsiB/YBFszaact7GOrDGv0YzcpwpOp15AIB1TJFkWm10rHWhRAoGBAJOK
p5mWnRav9ghAZK3IyQpvlrDolIAOmN46fHGC253H5dVvs93GwZwNk5/qhXqTx0iD
67Vt2GPGOmS8ltclsUc3bUYnlrKMByFtXayfCsqCMoaOU/PND3fDdqHVy0rcOgd9
0FGRc9CkOcVTZkca7dUqO9RZlpnuP4wU6hmVwk9hAoGAFLh9+WWQamU3bkEE2DTG
85EP1tHIpS9wKLC7OEbZO92c6XHgOGQALuYj/37kVPwzRbpXwjYZNNEdR/L8y/dC
D/wVymhIHmVwwfqpMoE/eUXdX1xVwO2Q5ptugKZHJxLh4YCAW1V1rBm9D2nklT+5
8HzY34z/uiH34bUm36apgJo=
-----END PRIVATE KEY-----
root@WSL:lab6_pliki>
```

# Questions

### What is the purpose of the parameters: modulus, publicExponent, privateExponent, prime ?

- modulus (n):

The product of two large prime numbers (p and q). A common component of the public and private key.
Used in encryption and decryption.

- publicExponent (e):

The public exponent. Most commonly has a value of 65537.
Used for message encryption or digital signature verification.

- privateExponent (d):

The private exponent. Calculated from e, p, and q.
Used to decrypt a message or generate a digital signature.

- prime1 (p) and prime2 (q):

Two prime numbers from which the modulus is formed.
They are used for internal private key calculations.

### Which parameters can be publicly available (which are part of the public key and which should be kept secret (are part of the private key) ?

- **Public (can be public, they are part of the public key):**
  - modulus (n)
  - publicExponent (e)
- **Private (must be kept secret):**
  - privateExponent (d)
  - prime1 (p)
  - prime2 (q)
  - and derived parameters, e.g. $d \mod (p-1)$, $d \mod (q-1)$, $q^{-1} $d \mod p$

# Task 2

Write and execute a command that extracts the RSA public key from the rsa.key file and stores it in the rsa.pubkey file and displays its contents.

## Implementation

```
root@WSL:lab6_pliki> openssl rsa -in rsa.key -pubout -out rsa.pubkey
writing RSA key
root@WSL:lab6_pliki> openssl rsa -pubin -in rsa.pubkey -text
```

```
Public-Key: (2048 bit)
Modulus:
00:97:ba:ad:c4:bb:34:77:52:2d:4c:f4:96:36:db:
4d:ef:0c:55:cb:5f:8c:22:53:7d:81:c8:10:20:cf:
c9:bd:6d:81:df:ad:0f:22:5c:aa:ef:e3:91:ec:3c:
8c:19:0a:46:38:7b:00:7e:00:e1:bf:2a:b8:71:bd:
f5:e2:a5:61:3b:e5:ed:ac:a4:b2:6e:54:8d:95:34:
dd:40:be:4b:e6:05:1b:c7:1e:d8:2a:8c:b9:19:fa:
bc:73:f8:23:13:d6:28:87:52:11:56:fe:05:d6:27:
bc:28:29:57:2d:f9:6f:7a:e4:ee:d2:d3:67:d3:ea:
b2:c2:6d:1b:cb:ba:4b:1a:b9:12:16:3f:0a:7d:e1:
83:89:12:9d:3b:03:e5:aa:1d:98:50:47:19:05:01:
cd:7f:e0:1f:f8:81:4a:b1:ce:99:e9:74:b1:81:9e:
ca:eb:a6:43:e6:b4:fe:f4:1c:48:15:c4:b3:82:68:
a5:e7:0f:b6:2f:82:69:d2:78:96:43:a3:72:bd:28:
44:24:f2:b7:c5:82:8b:71:77:e5:18:38:dc:f9:82:
7f:7d:38:70:d8:d1:3f:a6:66:3c:62:5a:bb:51:9c:
c6:5d:7c:38:ce:6d:77:7d:cc:80:b4:d8:77:79:46:
8b:8f:8a:57:f8:4a:0e:89:3b:42:f1:0f:0f:83:29:
28:7d
Exponent: 65537 (0x10001)
writing RSA key
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAl7qtxLs0d1ItTPSWNttN
7wxVy1+MIlN9gcgQIM/JvW2B360PIlyq7+OR7DyMGQpGOHsAfgDhvyq4cb314qVh
O+XtrKSyblSNlTTdQL5L5gUbxx7YKoy5Gfq8c/gjE9Yoh1IRVv4F1ie8KClXLflv
euTu0tNn0+qywm0by7pLGrkSFj8KfeGDiRKdOwPlqh2YUEcZBQHNf+Af+IFKsc6Z
6XSxgZ7K66ZD5rT+9BxIFcSzgmil5w+2L4Jp0niWQ6NyvShEJPK3xYKLcXflGDjc
+YJ/fThw2NE/pmY8Ylq7UZzGXXw4zm13fcyAtNh3eUaLj4pX+EoOiTtC8Q8Pgyko
fQIDAQAB
-----END PUBLIC KEY-----
root@WSL:lab6_pliki> cat rsa.pubkey
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAl7qtxLs0d1ItTPSWNttN
7wxVy1+MIlN9gcgQIM/JvW2B360PIlyq7+OR7DyMGQpGOHsAfgDhvyq4cb314qVh
O+XtrKSyblSNlTTdQL5L5gUbxx7YKoy5Gfq8c/gjE9Yoh1IRVv4F1ie8KClXLflv
euTu0tNn0+qywm0by7pLGrkSFj8KfeGDiRKdOwPlqh2YUEcZBQHNf+Af+IFKsc6Z
6XSxgZ7K66ZD5rT+9BxIFcSzgmil5w+2L4Jp0niWQ6NyvShEJPK3xYKLcXflGDjc
+YJ/fThw2NE/pmY8Ylq7UZzGXXw4zm13fcyAtNh3eUaLj4pX+EoOiTtC8Q8Pgyko
fQIDAQAB
-----END PUBLIC KEY-----
root@WSL:lab6_pliki>
```

# Task 3

1. Prepare a text file with a message, e.g. 'This is a confidential message' (Hint: use the pkeyutl command and name the file 'plain')
2 Write the syntax of the openssl command to encrypt the message contained in the „plain” file. Save

the ciphertext in a file named „encRSA”. Hint: use the „pkeyutl” command.
3. write the syntax of the openssl command used to decrypt the ciphertext from step #2

# Implementation

```
root@WSL:lab6_pliki> echo "This is a confidential message" > plain
root@WSL:lab6_pliki> openssl pkeyutl -encrypt -pubin -inkey rsa.pubkey -in
plain -out encRSA
root@WSL:lab6_pliki> openssl pkeyutl -decrypt -inkey rsa.key -in encRSA -out
decrypted.txt
root@WSL:lab6_pliki> hexdump -C encRSA
00000000  53 11 ca 10 6a b4 b3 b9  e3 51 c7 8c da 8c c0 6c
|S...j....Q.....l|
00000010  c2 df 0e f6 7c 28 c2 62  4e 5c 02 08 76 d8 27 56
|....|(.bN\..v.`V|
00000020  64 ec b2 b5 83 8e 7c 58  b3 f2 15 6e c8 8e 44 fd
|d.....|X...n..D.|
00000030  a4 40 6a 2c b4 b1 03 35  c8 b1 5c 4e f5 d8 3e d1
|.@j,...5..\N..>.|
00000040  68 47 a7 2a b5 bf a2 45  e0 7e de ca 5c e4 3f 99
|hG.*...E.~..\.?.|
00000050  82 9e 32 da 29 a1 e1 db  07 71 d9 7d da 06 e0 f1
|..2.)....q.}....|
00000060  6a 51 36 64 86 a8 ae 7d  9f 40 53 a3 61 5d fd 74
|jQ6d...}.@S.a].t|
00000070  f9 3f f6 e8 56 05 7b f8  fd 2a d6 9d ba e6 27 0a
|.?..V.{..*....`.|
00000080  88 a7 c7 1f 44 f3 37 50  04 b6 15 8b d4 4f 2f d4
|....D.7P.....O/.|
00000090  df d4 9b ea 11 2a 0d 2b  18 13 fb 2a 2c 0c 2c e7
|.....*.+...*,.,.|
000000a0  a5 25 d1 e7 f4 49 7b a0  92 9e c2 b2 73 ea c6 f8
|.%...I{.....s...|
000000b0  cc 79 a7 73 39 34 fd ab  19 5b 28 e2 3a 1f 9e 8b
|.y.s94...[(.:...|
000000c0  d3 df 35 f0 4c bd 09 82  89 ba 16 c7 8c 8f 09 12
|..5.L...........|
000000d0  98 53 02 ca 88 a5 be 1b  cb 43 fa db f6 68 fb 28
|.S.......C...h.(|
000000e0  41 b7 34 ac 0e ad fd bc  1a 78 a8 bb 3c 5f 51 c3
|A.4......x..<_Q.|
000000f0  5d 65 ce 7a 60 e9 7c 00  22 37 a5 a2 64 64 e0 8f
|]e.z`.|.``7..dd..|
00000100
root@WSL:lab6_pliki> hexdump -C decrypted.txt
00000000  54 68 69 73 20 69 73 20  61 20 63 6f 6e 66 69 64  |This is a
confid|
00000010  65 6e 74 69 61 6c 20 6d  65 73 73 61 67 65 0a     |ential
message.|
```

```
0000001f
root@WSL:lab6_pliki>
```

# Task 4

Write the syntax for the openssl command to create a signature (signature) of a 'plain' file AND save it in a file named 'sig'. Hint: use the 'openssl pk yeutl' command. Write the syntax of the openssl command to verify the veracity of the signature contained in the file 'sig'. Hint: when executed correctly, the screen should say „Signautre Verified Successfully”. Hint: use the 'openssl pkyeutl' command.

## Implementation

```
root@WSL:lab6_pliki> openssl pkeyutl -sign -inkey rsa.key -in plain -out sig
root@WSL:lab6_pliki> openssl pkeyutl -verify -pubin -inkey rsa.pubkey -in
plain -sigfile sig
Signature Verified Successfully
root@WSL:lab6_pliki>
```

## Questions

### What key should be used for the various operations performed in command 4 ?

- Signing (signature creation):

Use the private key (rsa.key) - only the owner of the key can sign the data.

- Signature verification:

Use a public key (rsa.pubkey) - anyone can verify the authenticity of the signature.

# Task 5

Machine user A wants to send a message to machine user B in a secure way. To do so, he wants to encrypt the message with a symmetric cipher, but machine user B does not have the secret key to decrypt the message. Write down the syntax of the commands and the correct sequence of actions that should be performed by the users of machines A and B. Then, carry out these actions and check whether the transmitted message was indeed decrypted by the user of machine B. Hint: use „opens sl enc” and „openssl pkeyutl”.

# Implementation

```
#Generowanie klucza symetrycznego:
root@WSL:lab6_pliki> openssl rand -out secret.key 32
#Szyfrowanie wiadomości przy użyciu klucza symetrycznego:
root@WSL:lab6_pliki> openssl enc -aes-256-cbc -salt -in plain -out
encMessage -pass file:./secret.key
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
#Szyfrowanie klucza symetrycznego za pomocą klucza publicznego użytkownika
maszyny B:
root@WSL:lab6_pliki> openssl pkeyutl -encrypt -pubin -inkey rsa.pubkey -in
secret.key -out encSecret.key
#Odszyfrowanie klucza symetrycznego przy użyciu klucza prywatnego:
root@WSL:lab6_pliki> openssl pkeyutl -decrypt -inkey rsa.key -in
encSecret.key -out secret.key
#Odszyfrowanie wiadomości przy użyciu odszyfrowanego klucza symetrycznego:
root@WSL:lab6_pliki> openssl enc -d -aes-256-cbc -in encMessage -out
decryptedMessage -pass file:./secret.key
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
root@WSL:lab6_pliki> cat plain
This is a confidential message
root@WSL:lab6_pliki> cat decryptedMessage
This is a confidential message
root@WSL:lab6_pliki> hexdump -C encMessage
00000000  53 61 6c 74 65 64 5f 5f  c6 35 4f 2f aa 19 44 ff
|Salted__.5O/..D.|
00000010  f8 e1 cf a5 eb 6d 65 d9  64 e8 5e e0 5f 3e 3e 34
|.....me.d.^._>>4|
00000020  ad f2 d0 0f c8 57 91 af  b3 89 5f 6f 13 0b db 65
|.....W...._o...e|
00000030
root@WSL:lab6_pliki> hexdump -C encSecret.key
00000000  1e 99 ed 85 4a c0 c4 55  d4 34 8d 52 5d 7b 0d ba
|....J..U.4.R]{..|
00000010  c9 a9 f4 2b 4a 87 f2 11  e7 a8 26 ea ad d9 b4 a6
|...+J.....&.....|
00000020  5a 61 21 22 6f dc a3 be  14 b4 68 c0 a8 f9 54 16
|Za!``o.....h...T.|
00000030  98 8f a1 46 6d 39 8f c1  58 8d 85 27 ef fc b5 e7
|...Fm9..X..`....|
00000040  bc 24 80 62 df d1 0d 3a  43 45 6d 41 9f 90 b6 58
|.$.b...:CEmA...X|
00000050  39 b5 7a 39 2f de 88 a7  89 8a e7 4c af 8f 7a 2d
|9.z9/......L..z-|
00000060  ea 45 a7 f2 88 9b 96 78  a1 d0 a2 a6 36 31 15 d3
|.E.....x....61..|
00000070  99 7f 44 a5 a1 db 83 56  d5 c7 90 4e 12 18 75 fc
|..D....V...N..u.|
```

```
00000080  53 cc e8 45 c3 94 f5 4c  72 4e ff 96 1f 99 40 a4
|S..E...LrN....@.|
00000090  86 fa 17 71 cf df 62 1e  e5 e7 c6 c4 a9 01 ed 33
|...q..b........3|
000000a0  60 67 73 46 86 d2 a3 73  91 c1 3a 9e 6d 30 60 f6
|`gsF...s..:.m0`.|
000000b0  2a 0c ea a8 28 d2 76 14  c5 d4 25 a0 b7 85 8d 27
|*...(.v...%....`|
000000c0  31 bf 83 7d 0b 1e be ab  2e c9 88 91 68 e1 11 73
|1..}........h..s|
000000d0  b5 f8 2b 1f 89 fa 88 85  8b 8f 95 9d 0d 9c dc 55
|..+............U|
000000e0  77 2b e5 8f 4e c7 7b 4b  38 ff d8 47 b1 08 60 a8
|w+..N.{K8..G..`.|
000000f0  a2 44 09 db ef e3 68 f8  04 63 73 d0 db 2e 92 c8
|.D....h..cs.....|
00000100
root@WSL:lab6_pliki>
```

# Task 6

Machine user A wants to send a message with integrity to machine user B. Write down the command syntax and the correct sequence of actions that should be performed by machine users A and B. Then perform these actions and check that machine user B has indeed succeeded in confirming the integrity of the received message. Run the command in two variations: 1) using the commands „openssl pkeyutl", 2) using the commands „openssl dgst".

## Implementation variant 1

```
root@WSL:lab6_pliki> openssl pkeyutl -sign -inkey rsa.key -in plain -out sig
root@WSL:lab6_pliki> openssl pkeyutl -verify -inkey rsa.pubkey -in plain -
sigfile sig
Signature Verified Successfully
```

## Implementation variant 2

```
root@WSL:lab6_pliki> openssl dgst -sha256 -sign rsa.key -out sig plain
root@WSL:lab6_pliki> openssl dgst -sha256 -verify rsa.pubkey -signature sig
plain
Verified OK
```

# Questions

### What happens if an attacker modifies the data sent by a user of machine A to a user of machine B (consider two situations: a) the data is modified but not the signature, b) the signature is modified but not the data) ? Justify your answer.

- **Modified data but not signature:**

If the attacker modifies the data (e.g. the message in the plain file) but does not modify the signature, then the signature verification by machine user B will fail because the signature was generated on the original content and the modified content does not match the signature. It will receive the error message.

- **Modified signature, but not the data:**

If the attacker only modifies the signature but does not change the data, then the signature verification will fail because the modified signature will not match the original data. The user of machine B will not be able to verify the correctness of the signature.

### What happens if the attacker substitutes the public key of machine user B with his own ? In this situation, can the attacker swap the original data and signature ? What will be the result of signature validation ? Justify your answer.

- If the attacker swaps the public key of machine user B with his own public key, then, although machine user A can sign the message correctly, machine user B will not be able to verify the signature because he will use the attacker's public key and not the correct public key of machine user B.
- The verification will always fail because the attacker does not know the private key of machine user B, so he will not be able to generate a valid signature that matches the original data.
- The signature verification by the attacker using his public key will always result in an error message - the signature will not match the data.

# Task 7

1. the user of machine A sends his public key to the user of machine B
The user of machine B generates a random string of characters and stores it in a file named 'random'.
The user of machine B generates a random string of characters and saves it in a file named 'random'.
The result is saved in a file named „challenge”. Write down the syntax of the command needed to perform this step (encryption). Use the openssl commands: rand and pkeyutl.
3. the user of machine B sends the „challenge” file to the user of machine A.
Machine user A decrypts the message stored in the challenge file and then sends the plain text back to machine user B (write the command syntax for decrypting the message). Save the decrypted message from the challenge file in a file named „response”.

5. machine user B compares the original message and the response received from machine user A. The command „diff b” can be used for comparison

# Implementation

```
root@WSL:lab6_pliki> #polecenia sa udawane gdyz cwiczenie jest robione na
jednej maszynie ale nie zmienia to wnioskow oraz przbiegu cwiczenia
root@WSL:lab6_pliki> #scp rsa.pubkey userB@machineB:/home/userB/
root@WSL:lab6_pliki> openssl rand -out random 32
root@WSL:lab6_pliki> openssl pkeyutl -encrypt -pubin -inkey rsa.pubkey -in
random -out challenge
root@WSL:lab6_pliki> #scp challenge userA@machineA:/home/userA/
root@WSL:lab6_pliki> openssl pkeyutl -decrypt -inkey rsa.key -in challenge -
out response
root@WSL:lab6_pliki> diff random response
root@WSL:lab6_pliki>
```

# Questions

## Assuming the correctness (authenticity) of the keys, state what weaknesses you see in the challenge response protocol described above.

1. **Lack of public key authentication:**

The public key of the user of machine A is transmitted without any verification. If an attacker intercepts this key, he can replace it with his public key. In this case, the user of machine B will encrypt his data with the attacker's public key, allowing the attacker to decrypt the message rather than the real user of machine A.

1. **No use of certificates:**

User A's public key is not verified by a trusted third party (e.g. in a certificate-based system). This can lead to a man-in-the-middle attack, where an attacker can intercept the communication and substitute the public key, preventing proper verification.„

1. **Lack of integrity checking mechanism:**

The protocol does not provide any form of integrity checking for transmitted data. If the messages are modified after being sent by the user of machine A, the user of machine B will not be able to detect that the data has been altered, as there is no form of hashing or signing of the messages.

## What are the possible solutions ?

1. **Public key verification:**

Instead of sending the public key directly, the user of machine A can send their public key via a

trusted third party (e.g. a certificate server). This allows the user of machine B to ensure that the public key is indeed from the user of machine A. Protocols such as TLS can also be used to ensure that communication between machines is encrypted and protected from eavesdropping.

1. **Digital signatures for public keys:**

Digital signatures can be used to ensure that the public key of machine user A is authentic and comes from a reliable source. Machine user B can then verify the signature to ensure that the public key is authentic.

1. **Use of certificates:**

Using X.509 certificates for public key signing and verification is good practice for such protocols. The certificate can be verified using built-in trusted Certificate Authorities (CAs), which provides assurance that the public key belongs to the correct user.

1. **Use of hash and signature functions:**

To ensure the integrity of the message, it is useful to use a hash function (e.g. SHA-256) and sign the message before sending it. The user of machine B could then compare the signed message with its version after decryption. This helps to ensure that the message has not been modified during transmission.

1. **Using a secure communication channel:**

It is worth considering using a secure communication channel, such as TLS, to exchange public keys and ensure data privacy and integrity.