

Security: Szyfrowanie Asymetryczne

Do realizacji zadania wykorzystano WSL1 na systemie Windows 2019 server.

Dystrybucja uruchomiona pod systemem WSL to Ubuntu 22.04.4 LTS

Do włączenia legacy providera w OpenSSL wykorzystano poniższy tutorial:

<https://lindevs.com/enable-openssl-legacy-provider-on-ubuntu>

Zadanie 1

Korzystając z polecenia openssl genrsa zapisz składnię polecenia, które wygeneruje parę kluczy (publiczny i prywatny) RSA o długości 2048 bitów i zapisze je w pliku o nazwie rsa.key. Po wygenerowaniu kluczy sprawdź zawartość pliku rsa.key za pomocą następującego polecenia: openssl rsa -in rsa.key -text

Realizacja

```
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-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCcCwggSjAgEAAoIBAQCXuq3EuzR3Ui1M
9JY2203vDFLX4wiU32ByBAgz8m9bYHfrQ8iXKrv45HsPIwZCKY4ewB+A0G/Krhx
vfXipWE75e2spLJuVI2VNN1AvkvmBRvHHtgqjLkZ+rxz+CMT1iiHUhFW/gXWJ7wo
KVct+W96507S02fT6rLCbRvLuksauRIWPwp94Y0JEp07A+WqHZhQRxkFAc1/4B/4
gUqxzpnpdLGBnsrrpkPmtP70HEgVxL0CaKXnD7YvgmnSeJZDo3K9KEQk8rfFgotx
d+UY0Nz5gn990HDY0T+mZjxiWrtRnMZdfDjObXd9zIC02Hd5RouPilf4Sg6J00Lx
Dw+DKSh9AgMBAAECggEABJ6Ac3NM0G+42lVojKNQ3udYu01y24lvfPx6uoz+NB/1
KrQUMd8cAxsptenCjdyW34pzbisypZSCPY1v1i6usSjnAgeym5emKMhnr3S0Q8
lsDHZp6ziGAguVPKBJGf0F5Lqqiwdoe082CY8TAEeqN92l2iTs6wEmIZZjhIsnr
P2DjRGots9JrWwM8af3uIfo0rDePbu71ZMVsLXCmVY08BBfl+2DqWmMhQoE52yXV
67Nj1CUSOMmn6aD9TQi/MD46w0Lwn6jNoL2xaq67IrF96PLw8AZpfsGnUJSAKH6
v/kUUX0pzg+414uzMsjyi2AwDCycf0Gi0DRngR59YQKBgQDHBWc2R9uwvGLhpGzH
sqLbMmEWIkZjIEY8APGB2/1G4+6dXBUBRSoQUuGPGu6CyphwWxt3je1nfBLdnuYg
uNBYPgRsJDPd40qRyDI/tiR65EJ06srWzoZrqa9m7QGKSX++7kILD0kkVIQssQp
EsFgfH+Nm0Zfq8BNB11UdPbkEQKBgQDDKywLuwBw1bH/iYMnIg48j9vKlr4ctDFX
CB2oxnxDJkFhoBBGFGrUTXa/i8nKzkf9aY4foX4LFAr40TAC9VZWRiF4cJXwuCo7
tr8HTLfqkmZ6cdB04E+4/4l5Dr4SJPSEdqlY3wQkbizos2/RcLSXULq0nY98qfh
roPwnzV5rQKBgFaApbS1F0rDZuMw0Wwu4hQN2J7rNNA+32gTL0U5fe0DDGJ+9Hm0
3VKNA5bUqNhWQF2Na+nR2V8CUZzcBMAyM/QeYWuzbIvCx5GnhxekDkUP00bICnPE
/i3rgevlsiB/YBFszaact7G0rDGv0Yzcpwp0p15AIB1TJFkWm10rHWhRAoGBAJOK
p5mWnRav9ghAZK3IyQpvlrDoLIA0mN46fHGC253H5dVvs93GwZwNk5/qhXqTx0iD
67Vt2GPG0mS8lTcLsUc3bUYnlrKMBYfTXayfCsqCmoa0U/PND3fDdqHVy0rc0gd9
0FGRc9Ck0cVTZkca7dUq09RZlpnuP4wU6hmVwk9hAoGAFLh9+WWQamU3bkEE2DTG
85EP1tHIpS9wKLC70EbZ092c6XHg0GQALuYj/37kVPwzRbpXwjYZNNEdR/L8y/dC
D/wVymhIHmVwvfqpMoE/eUXdX1xVw02Q5ptugKZHJxLh4YCAW1V1rBm9D2nk1T+5
8HzY34z/uiH34bUm36apgJo=
-----END PRIVATE KEY-----
root@WSL:lab6_pliki>
```

Pytania

Jakie jest przeznaczenie parametrów: modulus, publicExponent, privateExponent, prime ?

- modulus (n):

Iloczyn dwóch dużych liczb pierwszych (p i q). Wspólny składnik klucza publicznego i prywatnego. Używany w szyfrowaniu i deszyfrowaniu.

- publicExponent (e):

Wykładnik publiczny. Najczęściej ma wartość 65537. Służy do szyfrowania wiadomości lub weryfikacji podpisu cyfrowego.

- privateExponent (d):

Wykładnik prywatny. Obliczany na podstawie e, p, i q. Służy do odszyfrowywania wiadomości lub generowania podpisu cyfrowego.

- prime1 (p) i prime2 (q):

Dwie liczby pierwsze, na podstawie których tworzony jest modulus. Służą do wewnętrznych obliczeń klucza prywatnego.

Które z parametrów mogą być publicznie dostępne (które są częścią klucza publicznego a które powinny być utrzymywane w tajemnicy (są częścią klucza prywatnego) ?

- **Publiczne (mogą być jawne, wchodzi w skład klucza publicznego):**
 - modulus (n)
 - publicExponent (e)
- **Prywatne (muszą być utrzymywane w tajemnicy):**
 - privateExponent (d)
 - prime1 (p)
 - prime2 (q)
 - oraz parametry pochodne, np. $d \bmod (p-1)$, $d \bmod (q-1)$, $q^{-1} \bmod p$

Zadanie 2

Zapisz i wykonaj polecenie dokonujące ekstrakcji publicznego klucza RSA z pliku rsa.key i zapisującego go w pliku rsa.pubkey oraz wyświetlenie jego zawartości.

Realizacja

```
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-----
MIIBIjANBgkqhkiG9w0BAQEFAAAOCAQ8AMIIBCgKCAQEAl7qtxLs0d1ItTPSWNttN
7wxVy1+MILN9gcgQIM/JvW2B360PIlyq7+OR7DyMGQpG0HsAfgDhvyq4cb314qVh
0+XtrKSyblSNlTTdQL5L5gUbxx7YKoy5Gfq8c/gjE9Yoh1IRVv4F1ie8KCLXLflv
euTu0tNn0+qywm0by7pLGrkSFj8KfeGDiRKd0wPlqh2YUEcZBQHNf+Af+IFKsc6Z
6XSxgZ7K66ZD5rT+9BxIFcSzgmil5w+2L4Jp0niWQ6NyvShEJPK3xYKLCxflGDjc
+YJ/fThw2NE/pmY8Ylq7UZzGXXw4zm13fcyAtNh3eUaLj4pX+Eo0iTtC8Q8Pgyko
fQIDAQAB
-----END PUBLIC KEY-----
root@WSL:lab6_pliki> cat rsa.pubkey
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAAOCAQ8AMIIBCgKCAQEAl7qtxLs0d1ItTPSWNttN
7wxVy1+MILN9gcgQIM/JvW2B360PIlyq7+OR7DyMGQpG0HsAfgDhvyq4cb314qVh
0+XtrKSyblSNlTTdQL5L5gUbxx7YKoy5Gfq8c/gjE9Yoh1IRVv4F1ie8KCLXLflv
euTu0tNn0+qywm0by7pLGrkSFj8KfeGDiRKd0wPlqh2YUEcZBQHNf+Af+IFKsc6Z
6XSxgZ7K66ZD5rT+9BxIFcSzgmil5w+2L4Jp0niWQ6NyvShEJPK3xYKLCxflGDjc
+YJ/fThw2NE/pmY8Ylq7UZzGXXw4zm13fcyAtNh3eUaLj4pX+Eo0iTtC8Q8Pgyko
fQIDAQAB
-----END PUBLIC KEY-----
root@WSL:lab6_pliki>
```

Zadanie 3

1. Przygotuj plik tekstowy z wiadomością np. "This is a confidential message" (Podpowiedź: należy użyć komendy pkeyutl i nazwać plik "plain")
2. Napisz składnię polecenia openssl do zaszyfrowania wiadomości zawartej w pliku "plain".

Szyfrogram zapisz w pliku o nazwie "encRSA". Podpowiedź: należy użyć komendy "pkeyutl".

3. Napisz składnię komendy openssl używanej do odszyfrowania szyfrogramu z kroku nr 2

Realizacja

```
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.....0/.|
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>
```

Zadanie 4

Napisz składnię polecenia openssl do stworzenia sygnatury (podpisu) pliku 'plain' i zapisaniu jej w pliku o nazwie 'sig'. Podpowiedź: użyj polecenia "openssl pk yeutl". Napisz składnię polecenia openssl do weryfikacji prawdziwości sygnatury zawartej w pliku 'sig'. Podpowiedź: po poprawnym wykonaniu na ekranie powinna pojawić się informacja: "Signautre Verified Successfully". Podpowiedź: użyj polecenia "openssl pkeyutl".

Realizacja

```
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>
```

Pytania

Jakiego klucza należy użyć do poszczególnych operacji wykonanych w poleceniu nr 4 ?

- Podpisywanie (tworzenie sygnatury):

Należy użyć klucza prywatnego (rsa.key) – tylko właściciel klucza może podpisać dane.

- Weryfikacja podpisu:

Należy użyć klucza publicznego (rsa.pubkey) – każdy może sprawdzić autentyczność podpisu.

Zadanie 5

Użytkownik maszyny A chce przesłać do użytkownika maszyny B wiadomość w sposób bezpieczny. W tym celu chce zaszyfrować wiadomość szyfrem symetrycznym, jednak użytkownik maszyny B nie dysponuje tajnym kluczem do odszyfrowania wiadomości. Zapisz składnię poleceń i prawidłową sekwencję działań które powinny zostać wykonane przez użytkowników maszyn A i B. Następnie wykonaj te działania i sprawdź czy rzeczywiście przesłana wiadomość udało się odszyfrować użytkownikowi maszyny B. Podpowiedź: użyj poleceń "openssl enc" oraz "openssl pkeyutl".

Realizacja

```
#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___.50/..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>
```

Zadanie 6

Użytkownik maszyny A chce przesłać do użytkownika maszyny B wiadomość z zachowaniem jej integralności. Zapisz składnię poleceń oraz prawidłową sekwencję działań, które powinny zostać wykonane przez użytkowników maszyn A i B. Następnie wykonaj te działania i sprawdź czy rzeczywiście użytkownikowi maszyny B udało się potwierdzić integralność odebranej wiadomości. Polecenie wykonaj w dwóch wariantach: 1) używając poleceń “openssl pkeyutl”, 2) używając poleceń “openssl dgst”.

Realizacja wariant 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
```

Realizacja wariant 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
```

Pytania

Co się stanie jeżeli atakujący zmodyfikuje dane wysłane przez użytkownika maszyny A do użytkownika maszyny B (rozważ dwie sytuacje: a) zmodyfikowane zostaną dane ale nie sygnatura, b) zmodyfikowana zostanie sygnatura ale nie dane) ? Odpowiedź uzasadnij.

- **Zmodyfikowane dane, ale nie sygnatura:**

Jeśli atakujący zmodyfikuje dane (np. wiadomość w pliku plain), ale nie zmodyfikuje sygnatury, wówczas weryfikacja podpisu przez użytkownika maszyny B zakończy się niepowodzeniem, ponieważ podpis został wygenerowany na oryginalnej treści, a zmodyfikowana treść nie zgadza się z podpisem. Otrzyma komunikat o błędzie.

- **Zmodyfikowana sygnatura, ale nie dane:**

Jeśli atakujący zmodyfikuje tylko sygnaturę, ale nie zmieni danych, wówczas weryfikacja podpisu zakończy się niepowodzeniem, ponieważ zmieniony podpis nie będzie pasował do oryginalnych danych. Użytkownik maszyny B nie będzie w stanie zweryfikować poprawności podpisu.

Co się stanie gdy atakujący podmieni klucz publiczny użytkownika maszyny B na swój ? Czy w tej sytuacji atakujący może zamienić oryginalne dane i sygnaturę ? Jaki będzie wynik weryfikacji poprawności sygnatury ? Odpowiedź uzasadnij.

- Jeśli atakujący podmieni klucz publiczny użytkownika maszyny B na swój własny klucz publiczny, wówczas, pomimo że użytkownik maszyny A może podpisać wiadomość poprawnie, użytkownik maszyny B nie będzie w stanie zweryfikować podpisu, ponieważ użyje klucza publicznego atakującego, a nie właściwego klucza publicznego użytkownika maszyny B.
- Weryfikacja zawsze zakończy się niepowodzeniem, ponieważ atakujący nie zna klucza prywatnego użytkownika maszyny B, więc nie będzie mógł wygenerować prawidłowej sygnatury pasującej do oryginalnych danych.
- W wyniku weryfikacji podpisu przez atakującego, który będzie używał swojego klucza publicznego, zawsze będzie otrzymywał komunikat o błędzie — podpis nie będzie pasować do danych.

Zadanie 7

1. Użytkownik maszyny A wysyła swój klucz publiczny użytkownikowi maszyny B
2. Użytkownik maszyny B generuje losowy ciąg znaków i zapisuje go w pliku o nazwie "random". Następnie szyfruje go kluczem publicznym użytkownika maszyny A (otrzymany w kroku nr 1). Wynik zapisuje w pliku o nazwie "challenge". Zapisz składnię polecenia koniecznego do wykonania tego kroku (szyfrowania). Wykorzystaj komendy openssl: rand i pkeyutl.
3. Użytkownik maszyny B wysyła plik "challenge" do użytkownika maszyny A.
4. Użytkownik maszyny A odszyfrowuje wiadomość zapisaną w pliku "challenge" a następnie tekst

jawny odsyła użytkownikowi maszyny B (napisz składnię polecenia do odszyfrowania wiadomości). Odszyfrowaną wiadomość z pliku challenge zapisz w pliku o nazwie "response".

5. Użytkownik maszyny B porównuje wiadomość oryginalną oraz odpowiedź (response) na challenge odebraną od użytkownika maszyny A. Do porównania można użyć polecenia "diff b"

Realizacja

```
root@WSL:lab6_pliki> #polecenia sa udawane gdyz cwiczenie jest robione na
jednej maszynie ale nie zmienia to wnioskow oraz przebiegu 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>
```

Pytania

Zakładając poprawność (autentyczność) kluczy podaj jakie widzisz jakie słabości w protokole "challenge response" opisanym powyżej.

1. Brak uwierzytelniania klucza publicznego:

Klucz publiczny użytkownika maszyny A jest przekazywany bez żadnej weryfikacji. Jeśli atakujący przechwyci ten klucz, może go podmienić na swój klucz publiczny. W takim przypadku użytkownik maszyny B zaszyfruje swoje dane za pomocą klucza publicznego atakującego, co pozwala mu na odszyfrowanie wiadomości przez atakującego, a nie przez prawdziwego użytkownika maszyny A.

1. Brak użycia certyfikatów:

Klucz publiczny użytkownika A nie jest weryfikowany przez zaufaną stronę trzecią (np. w systemie opartym na certyfikatach). Może to prowadzić do ataku typu „man-in-the-middle”, gdzie atakujący może przechwycić komunikację i podmienić klucz publiczny, uniemożliwiając prawidłową weryfikację.

1. Brak mechanizmu sprawdzania integralności:

Protokół nie zapewnia żadnej formy sprawdzania integralności przesyłanych danych. Jeśli wiadomości zostaną zmodyfikowane po wysłaniu przez użytkownika maszyny A, użytkownik maszyny B nie będzie w stanie wykryć, że dane zostały zmienione, ponieważ nie ma żadnej formy hashowania lub podpisywania wiadomości.

Jakie są możliwe rozwiązania ?

1. Weryfikacja klucza publicznego:

Zamiast przesyłać klucz publiczny bezpośrednio, użytkownik maszyny A może wysłać swój klucz publiczny przez zaufaną stronę trzecią (np. serwer certyfikatów). Dzięki temu użytkownik maszyny B może upewnić się, że klucz publiczny pochodzi rzeczywiście od użytkownika maszyny A. Można również użyć protokołów takich jak TLS, aby zapewnić, że komunikacja między maszynami jest szyfrowana i chroniona przed podsłuchiwaniem.

1. Podpisy cyfrowe dla kluczy publicznych:

Można użyć podpisów cyfrowych, aby upewnić się, że klucz publiczny użytkownika maszyny A jest autentyczny i pochodzi z wiarygodnego źródła. Użytkownik maszyny B może wtedy zweryfikować podpis, aby upewnić się, że klucz publiczny jest autentyczny.

1. Wykorzystanie certyfikatów:

Używanie certyfikatów X.509 do podpisania kluczy publicznych oraz weryfikacji to dobra praktyka w przypadku takich protokołów. Certyfikat można zweryfikować przy użyciu wbudowanych zaufanych urzędów certyfikacji (CA), co daje pewność, że klucz publiczny należy do odpowiedniego użytkownika.

1. Zastosowanie funkcji skrótu i podpisu:

Aby zapewnić integralność wiadomości, warto zastosować funkcję skrótu (np. SHA-256) i podpisać wiadomość przed jej wysłaniem. Użytkownik maszyny B mógłby wtedy porównać podpisaną wiadomość z jej wersją po odszyfrowaniu. To pozwala upewnić się, że wiadomość nie została zmodyfikowana w trakcie transmisji.

1. Użycie bezpiecznego kanału komunikacji:

Warto rozważyć użycie bezpiecznego kanału komunikacji, jak np. TLS, w celu wymiany kluczy publicznych oraz zapewnienia prywatności i integralności danych.