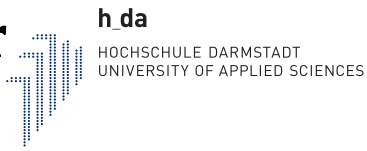
Cryptography – Summer 2023



by Prof. Dr. Alexander Wiesmaier

Praktikum 03

Date:	21.05.2023		
Supervisor:	Prof. Wiesmaier		
Students:	Réault	Corentin	Matr.Nr.1123639
	Bazan	Pablo	Matr.Nr.1120012
	Amoussou	Géraldo	Matr.Nr.1123370

Task 1 (RSA in openssl)

a) Generate an RSA key pair encrypted using 256bit AES

```
$ openssl genrsa -aes256 -out keyfile.pem 2048
```

b) Describe which encodings are used by openssl to store data structures

OpenSSL uses various encodings to store data structures like RSA key pairs. The two primary encodings used by OpenSSL for storing RSA key pairs are:

- **PEM (Privacy-Enhanced Mail):** PEM is a commonly used format in OpenSSL. It is a base64-encoded format that represents data structures in ASCII text. PEM files typically have extensions like .pem, .crt, .key, or .cer. PEM format is human-readable and can store not only RSA key pairs but also other cryptographic objects like X.509 certificates and private keys.
- **DER (Distinguished Encoding Rules):** DER is a binary format that represents data structures. It is a more compact and machine-readable encoding compared to PEM. DER files typically have extensions like .der or .crt. DER format is used for encoding cryptographic objects such as RSA keys, X.509 certificates, and certificate signing requests (CSRs).

The relationship between these encodings is that they can be used to represent the same data structure. For example, an RSA key pair can be stored in both PEM and DER formats. PEM format uses base64 encoding and includes additional header and footer lines to indicate the type of data (e.g., "BEGIN RSA PRIVATE KEYänd ËND

RSA PRIVATE KEY"). DER format, on the other hand, is a binary representation of the same data structure without any additional header or footer lines.
OpenSSL provides commands and functions to convert between these encodings. For example, the openssl rsa command can be used to convert an RSA key from PEM to DER format and vice versa.
Overall, these encodings provide flexibility in how data structures like RSA key pairs can be stored and used in different scenarios, allowing interoperability across various systems and applications.

c) Explain, which encoding is used for the file keyfile.pem

The content of the file is **Base64 ASCII encoded**. This encoding method allows the representation of various type of data, including cryptographic keys and certificates in a textual format using a set of 64 different characters (A-Z; a-z; 0-9 for the first 62 values as well as "+ and /"for the last two). In our case PEM files put base64 in action to illustrate binary data, making it easily readable and transferable across different systems.
The binary digit are first of all split into a set of 6 bits then each of those are convert into base10-format and finally each base10-number will match an index of the base64 alphabet.(The equal sign (=) is used for padding if necessary while splitting)

d) Convert the private keyfile keyfile.pem to DER and compare the length of the two files (explain the reason)

```
$ openssl rsa -outform der -in keyfile.pem -out keyfile.der
$ ls -la keyfile.*
-rw------- 1 cocopops cocopops 1216 18 mai   14:07 keyfile.der
-rw------- 1 cocopops cocopops 1874 18 mai   13:46 keyfile.pem
```

The .pem file is **heavier** than the .der file This can be explained on the one hand by the header and footer present in the .pem format, but also by the .pem and .der encodings themselves.
The base64 encoding (PEM) expands the original binary data, as each three bytes of binary data are represented by four ASCII characters in the base64 alphabet: These aspect putting together contribute to the larger file size of PEM files compared to the original binary representation.
The .DER encoding is faithful to the real size of the stored data because it is a binary format without any additional encoding overhead.

e) Print all components of both the public and the private key

```
$ openssl rsa -in keyfile.pem -text -noout


Private-Key: (2048 bit, 2 primes)
modulus:
    00:ba:3d:49:3d:33:20:49:95:0e:73:39:f4:f5:9a:
    aa:0f:fc:6d:d7:8d:b5:ea:dc:24:12:d9:54:27:4b:
    b6:7d:33:5d:e3:ad:5f:f8:38:1e:05:75:9d:83:6f:
    bb:1f:93:f0:b4:81:f9:b4:e2:ae:47:0d:43:ec:8f:
    bf:26:d5:1d:68:06:05:85:7d:1a:30:9f:02:f3:6b:
    f1:c8:45:61:f1:55:8f:57:1a:7e:97:48:2d:3b:d1:
    f8:29:aa:69:66:c3:b3:78:61:01:97:b8:b9:08:ce:
    4b:ec:1c:49:04:87:49:82:6e:c4:7f:84:7a:7c:c2:
    da:90:4c:00:60:63:d3:40:8e:94:02:c6:93:88:19:
    c4:c0:4b:30:e6:5d:2a:ef:5e:0c:d7:17:5f:cd:85:
    04:b5:be:1c:85:d0:9b:00:d7:eb:b8:a9:0d:e3:27:
    d5:64:fe:6f:68:26:4a:ab:07:60:47:fd:1c:ec:5d:
    0c:ee:4e:ff:f1:90:b3:0e:7b:bb:b0:06:73:1e:86:
    90:e1:8b:97:92:73:11:ae:cf:a8:3b:bd:db:4a:bb:
    c3:03:3b:ab:55:de:f7:68:03:00:23:d5:fe:53:13:
    97:93:00:73:81:18:c6:8c:5f:74:c4:0b:e1:7b:4c:
    56:36:6e:77:83:a1:a6:7c:cf:10:0f:d1:ca:8a:ef:
    15:51
publicExponent: 65537 (0x10001)
privateExponent:
    0a:ef:0c:cb:03:12:b3:e1:34:6e:c9:3d:f7:72:99:
    e5:fb:fd:58:a2:a3:25:cf:80:6e:6b:1c:0e:bd:ca:
    20:1f:f5:34:d7:fd:96:b3:78:37:3c:b6:47:74:76:
    7f:5a:4a:8f:15:63:a3:26:0d:02:62:bd:ec:70:2c:
    c2:2f:50:35:76:87:89:d4:8c:36:63:2a:7b:9f:28:
    6b:e8:1f:61:b9:52:c3:d3:cc:4f:6e:43:4a:14:17:
    da:3d:dd:07:e5:88:c6:82:26:f3:58:ea:e3:90:85:
    7a:6e:57:6e:d4:b0:b9:0f:ae:d9:77:ee:0f:3f:ab:
    10:5e:46:e6:21:cc:31:e9:a7:3e:c1:35:1b:03:da:
    c4:a3:c1:80:03:7e:3e:05:be:4d:93:2c:a6:93:72:
    9f:97:3a:ae:ad:58:43:a5:a9:96:7d:91:9f:54:98:
    45:3f:80:89:46:47:6d:ef:73:b9:f2:ea:d7:bf:fa:
    d9:de:f5:67:f4:6a:6a:df:8d:34:4e:97:7e:73:4b:
    48:69:c3:62:f4:f0:2c:ea:8b:61:77:32:06:53:8c:
    d3:76:03:75:4f:0d:d0:c4:87:c3:2a:0c:6c:9f:e7:
    71:7f:e5:aa:36:d7:87:d3:87:c3:4a:22:65:97:c1:
```

66:bc:62:71:86:aa:1d:76:c7:b6:e6:fa:bf:fb:70:
61
prime1:
00:c8:fb:35:3b:00:03:39:2e:**fc**:5c:do:0e:28:6f:
a7:0b:3c:09:74:60:fd:08:07:b2:ae:c7:53:d4:b3:
45:f1:c5:e1:2f:d3:02:38:df:41:f8:**fc**:b7:24:dd:
52:1b:96:a7:e3:11:00:f2:0b:ab:06:45:1d:23:d6:
fe:7a:2a:c4:dc:3e:0e:05:da:56:db:82:72:0e:95:
76:49:2e:3c:be:e2:ae:4d:2f:dd:08:4e:fe:ca:85:
4d:ec:4c:4a:33:fe:0d:9b:0d:03:11:4b:ec:2f:3b:
f7:91:60:03:74:b2:7e:6f:48:0e:d6:0b:bc:6d:eo:
9b:30:c9:ad:ao:do:30:22:31
prime2:
00:ed:38:f7:1e:01:cb:ff:84:do:73:8d:0b:cb:co:
62:61:62:22:77:e1:71:c3:6a:42:40:a4:bo:c8:do:
b5:60:e3:bc:c7:0f:98:95:1a:06:a5:c4:2d:06:db:
c8:ba:41:5a:29:db:3d:0f:36:**cd**:db:90:4f:74:aa:
9c:46:f4:ed:d5:91:36:41:00:75:37:81:36:a8:fe:
98:70:96:ao:03:64:4a:e1:0f:3f:**fc**:64:51:ec:15:
5a:f9:4a:25:3d:96:e4:81:93:7c:80:60:dc:d9:34:
d9:02:6f:**fc**:af:85:6a:14:06:b8:13:f2:e7:7e:a7:
c3:08:51:88:f9:a3:a5:3d:21
exponent1:
3c:b2:e9:cb:ac:eo:4c:57:ed:6d:d6:84:40:20:20:
72:1b:2a:bb:d4:42:f7:36:e3:f7:37:4a:11:36:27:
8c:b2:77:a5:2a:f4:43:da:e1:ao:e2:2a:29:df:11:
da:35:30:f5:3a:00:70:19:a8:08:57:bd:4a:42:b7:
4e:8c:36:32:52:27:88:0a:fe:2a:83:86:10:f3:80:
ea:ef:24:f7:7d:9c:a8:c1:28:df:46:84:5b:03:d1:
99:e9:1c:8c:01:51:78:9f:80:6a:ab:e4:64:0b:97:
64:77:7b:f3:b5:a8:cb:d1:16:71:**fc**:cf:66:db:eb:
e2:db:36:2e:18:52:41:31
exponent2:
39:61:0a:04:ec:12:57:df:dc:3b:d6:e5:ff:86:ad:
45:38:e3:75:73:c6:7c:a9:fb:14:7f:c1:73:11:68:
8d:e3:08:0a:6e:2f:4e:01:59:92:46:fd:4d:27:64:
4a:08:**fc**:b2:1b:21:8d:c8:87:ca:90:01:68:0a:cc:
7a:2d:4c:49:d8:31:f3:4f:15:0b:33:e9:fo:be:84:
48:d7:66:24:eb:e1:60:c6:bb:87:65:0c:9b:ba:1e:
a3:25:d8:14:29:a7:63:eo:34:c1:28:ac:c2:ad:11:
co:91:2d:e1:96:b7:ad:d8:62:36:11:0f:05:87:b6:

```
    99:10:7f:bd:42:63:f9:81
coefficient:
    5a:4b:e1:e4:e9:ff:7b:46:a6:ec:bf:97:8d:1e:35:
    cf:3f:83:dd:87:c7:63:76:69:6d:2d:56:ee:78:3a:
    73:e4:b1:6e:eb:a3:7e:c4:7e:6a:cb:c8:06:db:2e:
    35:4f:ae:fc:30:c5:43:3e:82:bd:9d:f8:3d:cb:d7:
    b6:b5:90:2c:46:0e:c6:ca:40:38:66:2b:6a:66:32:
    e5:51:b5:75:e7:ee:87:ec:77:58:2c:8d:28:0a:34:
    ae:e2:0d:af:df:a7:7c:b9:aa:15:95:c0:b0:04:47:
    6c:e1:19:73:89:c2:86:0e:bc:59:95:8e:e3:67:9f:
    0d:9f:72:45:f9:86:0c:76
```

- **Modulus (n):** is a large number that is part of the mathematical representation of the private key. It is a product of two prime numbers and forms the core of the key's mathematical structure.
- **Public Exponent (e):** is a relatively small prime number used in conjunction with the modulus for encryption and certain cryptographic operations. It is also part of the public key.
- **Private Exponent (d):** is the secret part of the private key. It is an integer that is inversely related to the public exponent. The private exponent is used for decryption, digital signing, and other operations that require the private key.
- **Prime 1 and 2 (p and q):** p and q, are large prime numbers that are multiplied together to obtain the modulus. Knowledge of these prime factors is crucial for certain cryptographic operations, such as key generation and decryption.
- **Exponent1 (d mod (p-1)):** This component is derived from the private exponent. It is calculated as the private exponent modulo (p-1).
- **Exponent2 (d mod (q-1)):** Similar to exponent1.It is calculated as the private exponent modulo (q-1).
- **Coefficient :** The coefficient is another component used in CRT optimizations (Chinese Remainder Theorem) for RSA. It is calculated as the modular inverse of q mod p.

$$q^{-1} \mod p$$

f) Export the public key from file keyfile.pem to the file keyfile_pub.pem and show (and explain) the different components saved in keyfile_pub.pem

```
$ openssl rsa -in keyfile_pub.pem -pubin -text -noout

Public-Key: (2048 bit)
Modulus:
    00:ba:3d:49:3d:33:20:49:95:0e:73:39:f4:f5:9a:
    aa:0f:fc:6d:d7:8d:b5:ea:dc:24:12:d9:54:27:4b:
    b6:7d:33:5d:e3:ad:5f:f8:38:1e:05:75:9d:83:6f:
    bb:1f:93:f0:b4:81:f9:b4:e2:ae:47:0d:43:ec:8f:
    bf:26:d5:1d:68:06:05:85:7d:1a:30:9f:02:f3:6b:
    f1:c8:45:61:f1:55:8f:57:1a:7e:97:48:2d:3b:d1:
    f8:29:aa:69:66:c3:b3:78:61:01:97:b8:b9:08:ce:
    4b:ec:1c:49:04:87:49:82:6e:c4:7f:84:7a:7c:c2:
    da:90:4c:00:60:63:d3:40:8e:94:02:c6:93:88:19:
    c4:c0:4b:30:e6:5d:2a:ef:5e:0c:d7:17:5f:cd:85:
    04:b5:be:1c:85:d0:9b:00:d7:eb:b8:a9:0d:e3:27:
    d5:64:fe:6f:68:26:4a:ab:07:60:47:fd:1c:ec:5d:
    0c:ee:4e:ff:f1:90:b3:0e:7b:bb:b0:06:73:1e:86:
    90:e1:8b:97:92:73:11:ae:cf:a8:3b:bd:db:4a:bb:
    c3:03:3b:ab:55:de:f7:68:03:00:23:d5:fe:53:13:
    97:93:00:73:81:18:c6:8c:5f:74:c4:0b:e1:7b:4c:
    56:36:6e:77:83:a1:a6:7c:cf:10:0f:d1:ca:8a:ef:
    15:51
Exponent: 65537 (0x10001)
```

The public key contains only the modulus and the public exponent (most of the time 65537). Moreover, the modulus corresponds to the product of two prime numbers, p and q, kept secret in the private key. The Exponent is therefore what is called public exponent in the private key

g) Which of the key parts are actually encoded within the file keyfile.der and which are computed on the fly by the openssl tool

```
$ openssl asn1parse -inform DER -in keyfile.der

    0:d=0  hl=4 l=1212 cons: SEQUENCE
    4:d=1  hl=2 l=   1 prim: INTEGER           :00
    7:d=1  hl=2 l=  13 cons: SEQUENCE
    9:d=2  hl=2 l=   9 prim: OBJECT            :rsaEncryption
   20:d=2  hl=2 l=   0 prim: NULL
   22:d=1  hl=4 l=1190 prim: OCTET STRING      [HEX DUMP]:308204A
20201000282010100BA3D493D332049950E7339F4F59AAA0FFC6DD78DB5EAD
C2412D954274BB67D335DE3AD5FF8381E05759D836FBB1F93F0B481F9B4E2A
```

E470D43EC8FBF26D51D680605857D1A309F02F36BF1C84561F1558F571A7E9
7482D3BD1F829AA6966C3B378610197B8B908CE4BEC1C49048749826EC47F8
47A7CC2DA904C006063D3408E9402C6938819C4C04B30E65D2AEF5E0CD7175
FCD8504B5BE1C85D09B00D7EBB8A90DE327D564FE6F68264AAB076047FD1CE
C5D0CEE4EFFF190B30E7BBBB006731E8690E18B97927311AECFA83BBDDB4AB
BC3033BAB55DEF768030023D5FE5313979300738118C68C5F74C40BE17B4C5
6366E7783A1A67CCF100FD1CA8AEF15510203010001028201000AEF0CCB031
2B3E1346EC93DF77299E5FBFD58A2A325CF806E6B1C0EBDCA201FF534D7FD9
6B378373CB64774767F5A4A8F1563A3260D0262BDEC702CC22F5035768789D
48C36632A7B9F286BE81F61B952C3D3CC4F6E434A1417DA3DDD07E588C6822
6F358EAE390857A6E576ED4B0B90FAED977EE0F3FAB105E46E621CC31E9A73
EC1351B03DAC4A3C180037E3E05BE4D932CA693729F973AAEAD5843A5A9967
D919F5498453F808946476DEF73B9F2EAD7BFFAD9DEF567F46A6ADF8D344E9
77E734B4869C362F4F02CEA8B61773206538CD37603754F0DD0C487C32A0C6
C9FE7717FE5AA36D787D387C34A226597C166BC627186AA1D76C7B6E6FABFF
B706102818100C8FB353B0003392EFC5CD00E286FA70B3C097460FD0807B2A
EC753D4B345F1C5E12FD30238DF41F8FCB724DD521B96A7E31100F20BAB064
51D23D6FE7A2AC4DC3E0E05DA56DB82720E9576492E3CBEE2AE4D2FDD084EF
ECA854DEC4C4A33FE0D9B0D03114BEC2F3BF791600374B27E6F480ED60BBC6
DE09B30C9ADA0D0302231028181 00ED38F71E01CBFF84D0738D0BCBC062616
22277E171C36A4240A4B0C8D0B560E3BCC70F98951A06A5C42D06DBC8BA415
A29DB3D0F36CDDB904F74AA9C46F4EDD59136410075378136A8FE987096A00
3644AE10F3FFC6451EC155AF94A253D96E481937C8060DCD934D9026FFCAF8
56A1406B813F2E77EA7C3085188F9A3A53D210281803CB2E9CBACE04C57ED6
DD684402020721B2ABBD442F736E3F7374A1136278CB277A52AF443DAE1A0E
22A29DF11DA3530F53A007019A80857BD4A42B74E8C36325227880AFE2A838
610F380EAEF24F77D9CA8C128DF46845B03D199E91C8C0151789F806AABE46
40B9764777BF3B5A8CBD11671FCCF66DBEBE2DB362E18524131028180396 10
A04EC1257DFDC3BD6E5FF86AD4538E37573C67CA9FB147FC17311688DE3080
A6E2F4E01599246FD4D27644A08FCB21B218DC887CA9001680ACC7A2D4C49D
831F34F150B33E9F0BE8448D76624EBE160C6BB87650C9BBA1EA325D81429A
763E034C128ACC2AD11C0912DE196B7ADD86236110F0587B699107FBD4263F
98102818 05A4BE1E4E9FF7B46A6ECBF978D1E35CF3F83DD87C76376696D2D5
6EE783A73E4B16EEBA37EC47E6ACBC806DB2E354FAEFC30C5433E82BD9DF83
DCBD7B6B5902C460EC6CA4038662B6A6632E551B575E7EE87EC77582C8D280
A34AEE20DAFDFA77CB9AA1595C0B004476CE1197389C2860EBC59958EE3679
F0D9F7245F9860C76

Explanation of the component:

- Sequence (Offset: 0, Length: 1212): This is the top-level structure of the DER data, indicating that it contains a sequence of components.

- Integer (Offset: 4, Length: 1): This is a primitive integer component with a length of 1 byte. The content of this integer is represented as "ooïn hexadecimal.
- Sequence (Offset: 7, Length: 13): This is a nested sequence that contains more components.
- Object Identifier (Offset: 9, Length: 9): This is a primitive component representing an object identifier. The content is "rsaEncryption,ïndicating the encryption algorithm used.
- Null (Offset: 20, Length: 0): This is a primitive component representing a null value. It doesn't contain any content.
- Octet String (Offset: 22, Length: 1190): This is a primitive component representing an octet string, which holds the encoded data. The content is displayed as a hexadecimal dump.

We stored all the components of the keypair found with the command "$ openssl rsa -in keyfile.pem -text -noout"(question e) in separate files which we then tried to find in the hexdump just above.

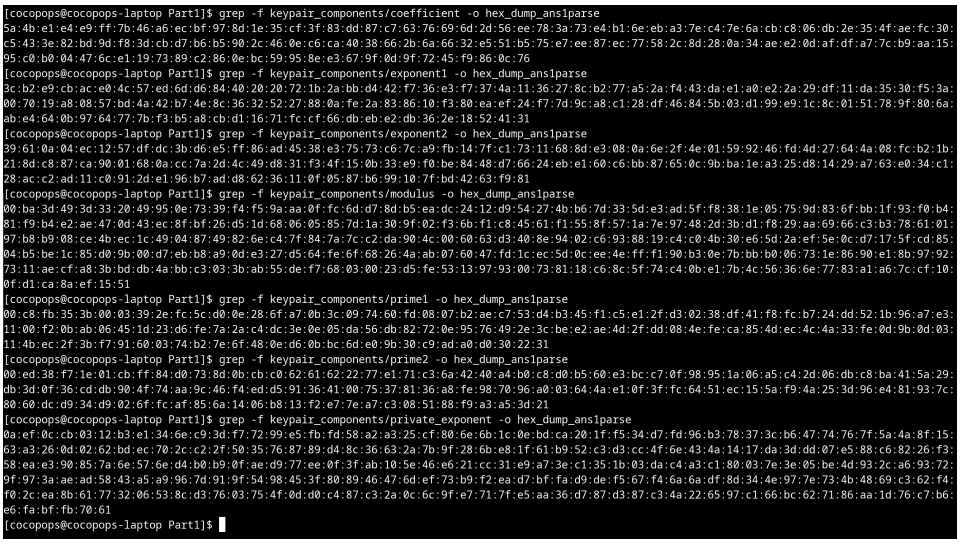


Abbildung 1: Finding key components in in the hexdump

All these tests show that all the components seem to be present in the DER format.

h) Sign the key file keyfile.der using SHA-512 as hash algorithm and your private RSA key. What is the length of the signature? Do you see a relation between the signature length and your RSA parameters?

Sign keyfile.der with keyfile.pem and SHA512 :

```
$ openssl dgst -sha512 -sign keyfile.pem -out
signed_keyfile_der.sha512 keyfile.der
```

Verify signature with public key :

```
$ openssl dgst -sha512 -verify keyfile_pub.pem -signature
signed_keyfile_der.sha512 keyfile.der

   Verified OK
```

Task 2 (Breaking RSA)

You know that Bob's public key is (2281437973, 3). You intercept the ciphertext 232770554. Please compute the private key and the underlying plaintext. Use whatever means you see fit, e.g. CrypTool.

For this part we used a software named RsaCtfTool, from a github repository

```
[cocopops@cocopops-laptop RsaCtfTool]$ ./RsaCtfTool.py -n 2281437973 -e 3 --uncipher 232770554 --private

[*] Testing key /tmp/tmpurgq4qhn.
attack initialized...
attack initialized...
[*] Performing factordb attack on /tmp/tmpurgq4qhn.
[*] Attack success with factordb method !

Results for /tmp/tmpurgq4qhn:

Private key :
-----BEGIN RSA PRIVATE KEY-----
MCoCAQACBQCH+/sVAgEDAgRapvmzAgMAjCcCAwD4YwICXW8CAwC1lwICfys=
-----END RSA PRIVATE KEY-----

Unciphered data :
HEX : 0x499602d2
INT (big endian) : 1234567890
INT (little endian) : 3523384905
utf-16 : 陒颡
STR : b'I\x96\x02\xd2'
[cocopops@cocopops-laptop RsaCtfTool]$
```

Abbildung 2: Breaking RSA with RsaCtfTool

The attack was very fast (almost instantaneous), and allowed us to decrypt the cyphertext and find the private key in PEM format. We then copied the private key into a key.pem file in order to use another function of the program and find the characteristics of the key.

```
[cocopops@cocopops-laptop RsaCtfTool]$ ./RsaCtfTool.py --dumpkey --key key.pem
private argument is not set, the private key will not be displayed, even if recovered.
n: 2281437973
e: 3
d: 1520892339
p: 35879
q: 63587
[cocopops@cocopops-laptop RsaCtfTool]$
```

Abbildung 3: Private key details

Literatur

[1] https://mbed-tls.readthedocs.io/en/latest/kb/cryptography/asn1-key-structures-in-der-and-pem/, last accessed: 19.05.2023

[2] https://medium.com/@bn121rajesh/rsa-sign-and-verify-using-openssl-behind-the-scen
cessed: 19.05.2023

[3] https://www.rfc-editor.org/rfc/rfc3447#page-42, last accessed: 19.05.2023

[4] https://github.com/RsaCtfTool/RsaCtfTool, last accessed: 19.05.2023