SSH KEY FILE FORMATS

Author: Lily Li
Conferred with: Jeff Ondich

Sources: https://datatracker.ietf.org/doc/html/rfc4251
https://datatracker.ietf.org/doc/html/rfc8017#appendix-A.1
https://datatracker.ietf.org/doc/html/rfc4716
https://letsencrypt.org/docs/a-warm-welcome-to-asn1-and-der/

Id_rsa_homework content:

-----BEGIN RSA PRIVATE KEY-----
MIIG4wIBAAKCAYEAtcgOmbeFckxidPOcBhfNb8CgqDvE+lXXoJLeWH+aDBosmKWCaEj7m+xmGEYE+3Zc2G/DvxuggYyefAZH4k
kszPBWKwrmbaSNrusJ8jXTTHGwwg8THEUVh0kN8gNKxWFbhF8FaWcDXev6vifJPBnCHhWiouijIbKJlRtO6+8olIdS0lQNsgSJ7hh
HLoTKlzqtdAkoUYGCd5VBKkWNSYwPmylL3v2InMhxF1dYd9PuDGBEIeyPifIMyHjmAdIQporW1owoMIh1WPls7LiKGuMLd7tQ7LFk
gX5F38gH3DyvPMgkfXyIILDkSmx6kSaZ2DJPUf3mCfgylmDQ1yFOvVYrzeEJ86WS8hbCvu+47kzCLgIpUIPTRCiJbO5YF50iMESN3
f634E7DFwasC7pjtlhrRTm89X4+4dhZMPsnsv6Nmq1jnSh/nezlrmD9WIMjCeXQoD9y9WOXE9vQl0Lh78zWZg+yC+4t67vBdeNU2X
yJnkTwokMLkwbHGaX64nmWT617AgMBAAECggGBALUoSmSmoDboKLakkDkCP5m/PlxqNEPGiL1doHhRshOdghQd3XhQQ2+A
h75pCU4RBzYrK8ion6x7lvXCzKCWjC8w1K7QjhN2ijxnv+HCAnGMW7sJiuIBdxL7mwo0fa929t1UVPBV2OUg5yHaRJHAmdOhfBMe
BHTwmGwgauZ10Ns0gTVazss5+rttNoMFk2Bi+blJ5wRTLNl/zFF0Pvmsu2DGJvZzdFHK7+lzApdofTOYPE4SknnG3zZqRc5Kqo1Ojl
4F3c6oAoTuQqzs/luSnFGBRlcVB4k71b+N4BAeiD7M7rXoEXih+5MRsbvCAlFidAuD9JVfzt1HVzHN2Uj6UPPjOlBmJ5wCXoHurv//Ad
9Agkcxd57QXvg2QgeU/n26WYXo6aFgVClVt0r7qw/Znja9l0zMk2FLmXL8w8q2tqscFVZdfim87/ytH1WbHG59Uzv4w2y6GP521QWe
kbPYRHe3lVpHZMMQoAnumNmpDN+xf5qoPqdPk8cjqS82sr5EAQKBwQDZjvaUGLbJr/ly4rnwp4nj38+gvERomyNA/aVxzwwAaLJT
zMkCvZAW8HS3mc31Hm4YM6G3xy4Yo/vvzzlx+QlhTXBlCzmLw20LCxBQK8aLRs3N1aQAKVWDU1CGmYOod8N0fIm+H+JhenvH
GV01Y7BMuXhtWbV9Ox+q8veJN2vo5/dUBWSktpHRt/o4YCQtb3wmrvTL7anwXztSeXcMW11eTs0BlZF3RcMbDVR5bb7fMP3nyfxq
rFBl58vqt9ccGFECgcEA1ebCJmKh6UdqFm8rETSjrZRrQE+QnRNt5URtljVZnqA9WKILwE3vRh8PtLjUVxScb6FgcXf+W0h6ny3qjyT
XPJVs8/KOkrAsa3gYsYPwWpncq/6WzlA9CQFGyn7kUiUAWHDPZP9hnKyL9CemQThIA44ioBWcJyh4nqeEzFrhl9v/l5TqBydtGSDP
8J75Tf/rJBm21Y79WRus/RcXiBeSDaoD84fmywxe1LH+FcvpsWmeZIYBLI/qYj9vvY7wygILAoHAOdTL7RAovrl9f/GO9c7DU8Sxv4ts
L9ZVZ7hFG1yrVwx9otMD9+uF+uMOU6PZ4LenEl3fNigHM9ubY+LU5VwKx0e7gdCbCnD3coENZBJO48T9fiK1Rqm2JoOmGzGBku
3LEhLs1pJJLhpN/t3aLhAC1FJVFdRL7y8hOuL9NAd8ctv+neIVH8e8OWBLbmL4Gu/Qs8O8JN1XWCModML/B1TeG+yciDhHA8qSk
4VWmDS5EUFHrS6/H2gQsxYL2afm0XwRAoHAZ7pIql6oT7z++DQeuXBCc3mG7R/obBOw9j5RukbS1/ay6RlkA3SHtQR5FFWRO0
1Wc1fZsgzTfybNuAWHOrVN3yZDOKWGeCIQ2Gjb6LGny/fTKGEEdrY4cvZ4gtg4FpG08i56lGq3xkdycwsLN5N4cKjrp5/zIpYgsCdiYc
ssnPzlnHdtady3rjjYPcKJ+UZCnDYQEM4Pk7e1EuKw2aE8hX3N/roHtK2iCxjps144TH2AfxTw2tBLhCtpH46CvCNzAoHAa74X8HW3q
XB+QD+Esgt++hx3HvjbSOc4OU2WdlkJIsuicnkCmcU9XnBTzMDVunM0dYmYDmbWYR0UZKQX12cY0yiTrPpbmXoI393tivKKBnj1
Ko8trpBwYci/nBT2acSEyylC1xwwCuST78+1eDnHmR/6QMt8Y/gGgu8ZlGS17UdqqeJfb6afNla76rlWgQl1Gx+TozElHzkW6XWAdPeI
f3habkxp2nSq8rJpRVKg2GJs4RS+4tlN6J5hvqmY7Iuo
-----END RSA PRIVATE KEY-----

Id_rsa_homework.pub content:

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQC1yA6Zt4VyTGJ085wGF81vwKCoO8T6Vdegkt5Yf5oMGiyYpYJoSPub7GYYRgT7dlzYb
8O/G6CBjJ58BkfiSSzM8FYrCuZtpI2u6wnyNdNMcbDCDxMcRRWHSQ3yA0rFYVuEXwVpZwNd6/q+J8k8GcIeFaKi6KMhsomVG07r
7yiUh1LSVA2yBInuGEcuhMqXOq10CShRgYJ3lUEqRY1JjA+bKUve/YicyHEXV1h30+4MYEQh7I+J8gzIeOYB0hCmitbWjCgwiHVY+
WzsuIoa4wt3u1DssWSBfkXfyAfcPK88yCR9fIggsORKbHqRJpnYMk9R/eYJ+DKWYNDXIU69VivN4QnzpZLyFsK+77juTMIuAilQg9N
EKIls7lgXnSIwRI3d/rfgTsMXBqwLumO2WGtFObz1fj7h2Fkw+yey/o2arWOdKH+d7OWuYP1YgyMJ5dCgP3L1Y5cT29CXQuHvzNZ
mD7IL7i3ru8F141TZfImeRPCiQwuTBscZpfrieZZPrXs= lilyli@LilydeMacBook-Pro.local

=== Private Key ===

A. Items contained in *id_rsa_homework*:

1: PEM private key headers and footers, specifying the type of data the begin/end of data:

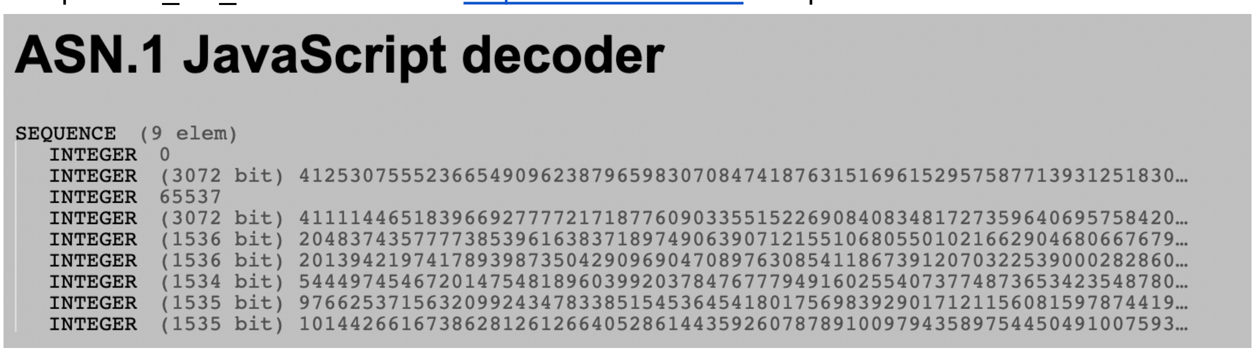
-----BEGIN RSA PRIVATE KEY----- -----END RSA PRIVATE KEY-----

2: A base-64 encoded version of the DER encoded ASN.1 type *RSAPrivateKey* in the following structure:

```
RSAPrivateKey ::= SEQUENCE {
   version           Version,
   modulus           INTEGER,  -- n
   publicExponent    INTEGER,  -- e
   privateExponent   INTEGER,  -- d
   prime1            INTEGER,  -- p
   prime2            INTEGER,  -- q
   exponent1         INTEGER,  -- d mod (p-1)
   exponent2         INTEGER,  -- d mod (q-1)
   coefficient       INTEGER,  -- (inverse of q) mod p
   otherPrimeInfos   OtherPrimeInfos OPTIONAL
}
```

B. Decode Private Key File:

1: Upload *id_rsa_homework* to the [Lapo ASN.1 decoder] and press button ***decode***



To get the HEX version of the decoded *INTEGERS*, we can use the [holtstrom decoder], copy paste in the base-64 data without the header/footer from *id_rsa_homework* and press button ***decode***

Input

MIIG4wIBAAKCAYEAtcgOmbeFckxidPOcBhfNb8CgqDvE+lXXoJLeWH+aDBosmKWC
aEj7m+xmGEYE+3Zc2G/DvxuggYyefAZH4kkszPBWKwrmbaSNrusJ8jXTTHGwwg8T
HEUVh0kN8gNKxWFbhF8FaWcDXev6vifJPBnCHhWiouijIbKJlRtO6+8olIdS0lQN
sgSJ7hhHLoTKlzqtdAkoUYGCd5VBKkWNSYwPmylL3v2InMhxF1dYd9PuDGBEIeyP
ifIMyHjmAdIQporW1owoMIh1WPls7LiKGuMLd7tQ7LFkgX5F38gH3DyvPMgkfXyI
ILDkSmx6kSaZ2DJPUf3mCfgylmDQ1yFOvVYrzeEJ86WS8hbCvu+47kzCLgIpUIPT

Convert | BASE64/PEM to ASN.1

Output

SEQUENCE {
   INTEGER 0x00 (0 decimal)
   INTEGER
0x00b5c80e99b785724c6274f39c0617cd6fc0a0a83bc4fa55d7a092de587f9a0
c1a2c98a5826848fb9bec66184604fb765cd86fc3bf1ba0818c9e7c0647e2492c
ccf0562b0ae66da48daeeb09f235d34c71b0c20f131c451587490df2034ac5615
b845f056967035debfabe27c93c19c21e15a2a2e8a321b289951b4eebef289487
52d2540db20489ee18472e84ca973aad74092851818277954l2a458d498c0f9b2
94bdefd889cc87117575877d3ee0c604421ec8f89f20cc878e601d210a68ad6d6
8c2830887558f96cecb88a1ae30b77bb50ecb164817e45dfc807dc3caf3cc8247
d7c8820b0e44a6c7a912699d8324f51fde609f8329660d0d7214ebd562bcde109
f3a592f216c2beefb8ee4cc22e02295083d34428896cee58179d2230448dddfeb

C. Decoded File Explained:

1: What are the integers? What are the corresponding values (in HEX)

We got 9 decoded integers, corresponding to 9 components of the *RSAPrivateKey* structure:

Name: Version

version is the version number, according to rfc8017, unless multi-prime, it’s 0.

Offset: 4 bytes

Length: 2 bytes(Object description) + 1 bytes(content)

Value: 0x00

Name: modulus

RSA modulus n, the product of two distinct prime numbers p and q

Offset: 7 bytes

Length: 4 bytes(Object description) + 385 bytes(content)

Value:

0x00b5c80e99b785724c6274f39c0617cd6fc0a0a83bc4fa55d7a092de587f9a0c1a2c98a5826848fb9bec6
6184604fb765cd86fc3bf1ba0818c9e7c0647e2492cccf0562b0ae66da48daeeb09f235d34c71b0c20f131c4
51587490df2034ac5615b845f056967035debfabe27c93c19c21e15a2a2e8a321b289951b4eebef2894875
2d2540db20489ee18472e84ca973aad74092851818277954l2a458d498c0f9b294bdefd889cc871175758
77d3ee0c604421ec8f89f20cc878e601d210a68ad6d68c2830887558f96cecb88a1ae30b77bb50ecb16481
7e45dfc807dc3caf3cc8247d7c8820b0e44a6c7a912699d8324f51fde609f8329660d0d7214ebd562bcde10
9f3a592f216c2beefb8ee4cc22e02295083d34428896cee58179d2230448dddfeb7e04ec31706ac0bba63b
6586b4539bcf57e3ee1d85930fb27b2fe8d9aad639d287f9dece5ae60fd58832309e5d0a03f72f5639713db
d09742e1efccd6660fb20bee2debbbc175e354d97c899e44f0a2430b9306c719a5fae279964fad7b

Name: publicExponent

RSA public exponent e where gcd(e,lcm(p-1,q-1)) = 1
Offset: 396 bytes
Length: 2 bytes(Object description) + 3 bytes(content)
Value:
0x010001

Name: privateExponent
RSA private exponent d where d = e^(-1)mod lcm(p-1,q-1) = 1.
Offset: 401 bytes
Length: 4 bytes(Object description) + 385 bytes(content)
Value:
0x00b5284a64a6a036e828b6a49039023f99bf3e5c6a3443c688bd5da07851b2139d82141ddd7850436f8
087be69094e1107362b2bc8a89fac7b96f5c2cca0968c2f30d4aed08e13768a3c67bfe1c202718c5bbb098a
e2017712fb9b0a347daf76f6dd5454f055d8e520e721da4491c099d3a17c131e0474f0986c206ae675d0db
3481355acecb39fabb6d368305936062f9b949e704532cd23fcc51743ef9acbb60c626f6737451caefe9730
297687d33983c4e129279c6df366a45ce4aaa8d4e8e5e05ddcea80284ee42acecfe5b929c518146571507
893bd5bf8de0101e883ecceeb5e81178a1fb9311b1bbc2025162740b83f4955fcedd475731cdd948fa50f3e
33a5066279c025e81eeaeffff01df40824731779ed05ef836420794fe7dba5985e8e9a160542955b74afbab0
fd99e36bd974ccc93614b9972fcc3cab6b6ab1c15565d7e29bceffcad1f559b1c6e7d533bf8c36cba18fe76d
5059e91b3d84477b7955a4764c310a009ee98d9a90cdfb17f9aa83ea74f93c723a92f36b2be4401

Name: prime1
prime factor p of n, where pq=n
Offset: 790 bytes
Length: 3 bytes(Object description) + 193 bytes(content)
Value:
0x00d98ef69418b6c9aff972e2b9f0a789e3dfcfa0bc44689b2340fda571cf0c0068b253ccc902bd9016f074b
799cdf51e6e1833a1b7c72e18a3fbefcf3971f909614d70650b398bc36d0b0b10502bc68b46cdcdd5a40029
55835350869983a877c3747c89be1fe2617a7bc7195d3563b04cb9786d59b57d3b1faaf2f789376be8e7f75
40564a4b691d1b7fa3860242d6f7c26aef4cbeda9f05f3b5279770c5b5d5e4ecd0195917745c31b0d54796d
bedf30fde7c9fc6aac5065e7cbeab7d71c1851

Name: prime2
prime factor q of n, where pq=n
Offset: 986 bytes
Length: 3 bytes(Object description) + 193 bytes(content)
Value:
0x00d5e6c22662a1e9476a166f2b1134a3ad946b404f909d136de5446d9635599ea03d58a20bc04def461f
0fb4b8d457149c6fa1607177fe5b487a9f2dea8f24d73c956cf3f28e92b02c6b7818b183f05a99dcabfe96ce5
03d090146ca7ee45225005870cf64ff619cac8bf427a6413848038e22a0159c2728789ea784cc5ae197dbff
9794ea07276d1920cff09ef94dffeb2419b6d58efd591bacfd17178817920daa03f387e6cb0c5ed4b1fe15cbe
9b1699e6486012c8fea623f6fbd8ef0ca020b

Name: exponent1
d mod (p - 1)
Offset: 1182 bytes

Length: 3 bytes(Object description) + 192 bytes(content)
Value:
0x39d4cbed1028beb23d7ff18ef5cec353c4b1bf8b6c2fd65567b8451b5cab570c7da2d303f7eb85fae30e53a3d9e0b7a7125ddf36280733db9b63e2d4e55c0ac747bb81d09b0a70f772810d64124ee3c4fd7e22b546a9b62683a61b318192edcb1212ecd692492e1a4dfeddda2e1002d4525515d44bef2f213ae2fd34077c72dbfe9de2151fc7bc39604b6e62f81aefd0b3c3bc24dd5758232874c2ff0754de1bec9c88384703ca929385569834b9114147ad2ebf1f6810b3160bd9a7e6d17c11

Name: exponent2
d mod (q - 1)
Offset: 1377 bytes
Length: 3 bytes(Object description) + 192 bytes(content)
Value:
0x67ba48aa5ea84fbcfef8341eb97042737986ed1fe86c13b0f63e51ba46d2d7f6b2e91964037487b504791455913b4d567357d9b20cd37f26cdb805873ab54ddf264338a586782210d868dbe8b1a7cbf7d328610476b63872f67882d8381691b4f22e7a946ab7c64772730b0b37937870a8eba79ff3229620b0276261cb2c9cfce59c776d69dcb7ae38d83dc289f946429c361010ce0f93b7b512e2b0d9a13c857dcdfeba07b4ada20b18e9b35e384c7d807f14f0dad04b842b691f8e82bc2373

Name: coefficient
q^(-1) mod p
Offset: 1572 bytes
Length: 3 bytes(Object description) + 192 bytes(content)
Value:
0x6bbe17f075b7a9707e403f84b20b7efa1c771ef8db48e738394d9676590922cba272790299c53d5e7053ccc0d5ba73347589980e66d6611d1464a417d76718d32893acfa5b997a08dfdded8af28a0678f52a8f2dae907061c8bf9c14f669c484cb2942d71c300ae493efcfb57839c7991ffa40cb7c63f80682ef199464b5ed476aa9e25f6fa69f3656bbeab9568109751b1f93a331081f3916e9758074f7887f785a6e4c69da74aaf2b2694552a0d8626ce114bee2d94de89e61bea998ec8ba8

2: Interpretation of bytes from the decoded base64

Although our decoder has automatically decoded the DER encoding for us, we can still find the HEX represented DER encoding on the side of the Lapo ASN.1 decoder when we decode.

```
30 82 06 E3 02 01 00 02  82 01 81 00 B5 C8 0E 99
B7 85 72 4C 62 74 F3 9C  06 17 CD 6F C0 A0 A8 3B
C4 FA 55 D7 A0 92 DE 58  7F 9A 0C 1A 2C 98 A5 82
68 48 FB 9B EC 66 18 46  04 FB 76 5C D8 6F C3 BF
1B A0 81 8C 9E 7C 06 47  E2 49 2C CC F0 56 2B 0A
… skipping 288 bytes …
2D EB BB C1 75 E3 54 D9  7C 89 9E 44 F0 A2 43 0B
93 06 C7 19 A5 FA E2 79  96 4F AD 7B 02 03 01 00
01 02 82 01 81 00 B5 28  4A 64 A6 A0 36 E8 28 B6
A4 90 39 02 3F 99 BF 3E  5C 6A 34 43 C6 88 BD 5D
A0 78 51 B2 13 9D 82 14  1D DD 78 50 43 6F 80 87
BE 69 09 4E 11 07 36 2B  2B C8 A8 9F AC 7B 96 F5
C2 CC A0 96 8C 2F 30 D4  AE D0 8E 13 76 8A 3C 67
… skipping 288 bytes …
98 D9 A9 0C DF B1 7F 9A  A8 3E A7 4F 93 C7 23 A9
2F 36 B2 BE 44 01 02 81  C1 00 D9 8E F6 94 18 B6
C9 AF F9 72 E2 B9 F0 A7  89 E3 DF CF A0 BC 44 68
9B 23 40 FD A5 71 CF 0C  00 68 B2 53 CC C9 02 BD
90 16 F0 74 B7 99 CD F5  1E 6E 18 33 A1 B7 C7 2E
18 A3 FB EF CF 39 71 F9  09 61 4D 70 65 0B 39 8B
… skipping 96 bytes …
45 C3 1B 0D 54 79 6D BE  DF 30 FD E7 C9 FC 6A AC
```

DER is a ***type-length-value*** encoding. For every piece of data, you will first encounter bytes which indicate the data type (identifier: ASN.1 tag and type number), then bytes indicating length of the data value, and then bytes representing value. In the decoder, the identifier bytes are nicely marked in **blue**, and the length in **green**. *To understand them better, we will transform HEX into binary data.*

The identifier encoding encodes the ASN.1 tag's class number and type number. It also encodes whether the contents octets represent a constructed or primitive value.

The length encoding of DER must take the definite form. There is a short form and a long form. The short form consists of a single octet in which bit 8 is 0, and bits 1–7 encode the length. The long form consists of 1 initial octet followed by 1 or more subsequent octets, containing the length.

Examples:

Take the first data piece for instance, the offset is 4 bytes:

`30 82 06 E3`

Binary: **0011 0000** **1000 0010 0000 0110 1110 0011**

Now we will analyze this encoding bit by bit:

First Part:

0011 0000:

From each bit of **0011 0000** we can know:

00: indicates data is a Universal class

1: indicates content is constructed, containing 0, 1, or more encodings

10000 (Decimal 16): indicates type of data is a SEQUENCE. From the Tag type table we will find it is a SEQUENCE Tag

Second Part:

1000 0010 0000 0110 1110 0011

From each bit of **1000 0010** 0000 0110 1110 0011 we can know:

This is a **Long form** length encoding.

1: indicates that this is a long form encoding

000 0010 (Decimal 3): the number of length bytes that follows, in this case there are 2

0000 0110 0001 0011: In big-endian, the number of bytes in the data

An INTEGER example which follows the previous bytes:

`02 01 00`

Binary: 0000 0010 0000 0001 0000 0000

First Part:
0000 0010
From each bit of **0000 0010** we can know:
00: indicates data is a Universal class
0: indicates data is primitive class
0 0010 (Decimal 2): is the tag of INTEGER, so data type is INTEGER

Second Part:
0000 0001
From each bit of **0000 0001** we can know:
This is a **Short form** length encoding.
0: indicates that this is a short form encoding
0000001(Decimal 1): the actual data is only 1 byte long

Summary: All of the decoded Hex data can be interpreted/ understood in format similar to the examples above.

=== Public Key ===

A. Items contained in *id_rsa_homework.pub*:

The public key saved by ssh-keygen is written in the so-called SSH-format, with the structure: *Algorithm, Key, Comment*.

1: *Algorithm*
Public key type/format. Our *id_rsa_homework.pub* starts with string ‘ssh-rsa’ (because we are using the RSA algorithm)

2: *Key*
A base-64 encoded version of the *key* *(more explained later)*

3: Comment
In the format user@host.

B. Decode Private Key File:

After getting rid of the comment (end) and the algorithm (start), I used a base-64 to hex doctor from https://cryptii.com/pipes/base64-to-hex to decode the base-64 ***Key*** into Hex, resulting in the following:

< color of the hex numbers corresponds to the later explanation >

00 00 00 07 73 73 68 2d 72 73 61 00 00 00 03 01 00 01 00 00 01 01 00 b2 f5 fd 3f 9f 09 17 11 2c e4 2f 8b f8 7e d6 76 e1 52 58 be 44 3f 36 de af b0 b6 9b de 24 96 b4 95 ea ad 1b 01 ca d8 42 71 b0 14 e9 6f 79 38 6c 63 6d 34 85 16 da 74 a6 8a 8c 70 fb a8 82 87 0c 47 b4 21 8d 8f 49 18 6d df 72 72 7b 9d 80 c2 19 11 c3 e3 37 c6 e4 07 ff b4 7c 2f 27 67 b0 d1 64 d8 a1 e9 af 95 f6 48 1b f8 d9 ed fb 2e 39 04 b2 52 92 68 c4 60 25 6f af d0 a6 77 d2 98 98 f1 0b 1d 15 12 8a 69 58 39 fc 08 ed d5 84 e8 33 56 15 b1 d1 d7 27 7b e6 5c 53 2d ca 92 dd c7 05 03 74 86 8b 11 7e a9 15 49 14 ef 92 92 b8 44 3f 13 69 6e 4f ad 50 de d6 bd 90 e5 a6 f7 ed 33 be 2e ce 31 c6 dd 7a 42 53 ee 6c dc 56 78 7d dd 1d 5c d7 76 61 40 22 db 87 d0 3b b2 2f 23 28 5b 5a 31 67 af 8d ac ab be a4 00 04 47 13 37 d3 78 1e 8c 5c ca 0e a5 e2 77 99 b5 10 e4 ef 93 8c 61 ca a6 0d

C. Decoded File Explained:

1: What do the Hex numbers (Key) represent?

The "ssh-rsa" key format has the following specific encoding:

STRING "ssh-rsa"
MPINT e
MPINT n

A **STRING** contains**:**
4 bytes indicating length, the number of following bytes which contains the string.

A **MPINT** contains:
Multiple precision integers in two's complement format,stored as a **STRING** (therefore, also has the structure: 4 bytes for length + content bytes)

Now we can look at what all the hex numbers mean.

A.

First of all, We have the **STRING:**
00 00 00 07 73 73 68 2d 72 73 61

4 bytes indicating the string is 7 bytes long +
hex representation of the ASCII characters ‘ssh-rsa’

B.
Then it is followed by the **MPINT e**, the RSA exponent, also in the structure of a **STRING**
00 00 00 03 01 00 01

4 bytes indicating the number is 3 bytes long +
Hex number 010001 (Decimal 65537), very common RSA exponent

C.

Then it is followed by the **MPINT n**, the large product of two distinct prime numbers p and q, also in the structure of a **STRING**

00 00 01 01 00 b2 f5 fd 3f 9f 09 17 11 2c e4 2f 8b f8 7e d6 76 e1 52 58 be 44 3f 36 de af b0 b6 9b de 24 96 b4 95 ea ad 1b 01 ca d8 42 71 b0 14 e9 6f 79 38 6c 63 6d 34 85 16 da 74 a6 8a 8c 70 fb a8 82 87 0c 47 b4 21 8d 8f 49 18 6d df 72 72 7b 9d 80 c2 19 11 c3 e3 37 c6 e4 07 ff b4 7c 2f 27 67 b0 d1 64 d8 a1 e9 af 95 f6 48 1b f8 d9 ed fb 2e 39 04 b2 52 92 68 c4 60 25 6f af d0 a6 77 d2 98 98 f1 0b 1d 15 12 8a 69 58 39 fc 08 ed d5 84 e8 33 56 15 b1 d1 d7 27 7b e6 5c 53 2d ca 92 dd c7 05 03 74 86 8b 11 7e a9 15 49 14 ef 92 92 b8 44 3f 13 69 6e 4f ad 50 de d6 bd 90 e5 a6 f7 ed 33 be 2e ce 31 c6 dd 7a 42 53 ee 6c dc 56 78 7d dd 1d 5c d7 76 61 40 22 db 87 d0 3b b2 2f 23 28 5b 5a 31 67 af 8d ac ab be a4 00 04 47 13 37 d3 78 1e 8c 5c ca 0e a5 e2 77 99 b5 10 e4 ef 93 8c 61 ca a6 0d

4 bytes indicating the public key is 257 bytes long +
The long hex version of the public key

=== Sanity Check ===

Expected Relationships Check (Python code at the bottom):

A. e*d mod lambda(n) = 1

From MPINT e in *id_rsa_homework.pub* we know e= 65537, from decoded *id_rsa_homework* we know p, q, and d. Now we plug them into a python code to find lambda(n) check if calculations work (converted to decimals)

And it worked, e*d mod lambda(n) = 1!

```
(e*d)%lambda_n is:  1
```

B. n = p*q

Now we check if the **modulus**(n) from *RSAPrivateKey* equals to **prime1** times **prime2**

And it worked,n = p*q

```
p*q equals n
```

C. gcd(e,(p − 1)(q − 1)) = 1

We will use e from public key file and p,q from private key file to check:

And it worked, gcd(e,(p − 1)(q − 1)) = 1

```
gcd(e,(q-1)(p-1)) equals 1
```

D. Check 1 < e < λ(n), gcd(e, λ(nA)) = 1

And it worked!

```
1<e<lambda_n and math.gcd(e, lambda_n) ==1
```

Python code :

```python
import math

n =
4125307555236654909623879659830708474187631516961529575877139312518309222239049736608215889547586072773978857840904064062951739184647958075557305682892540361581959917165213656824303657817706460621569263804428873270175291989960526333496846322929272132546712182209303641705683721579454396735919235151217631210414024900924988249779201495217385751324403975919692578411911108106114840919011578186103297496761836400497105249843700575511127178902126227918112808544420243468749502088511383092793433966021929596223289257637877900923771721608723602630525286193130726698464489321350481643254243452175459327636046421743620025210490127234521886688055570477229813251450599421650943218151716432081230249856969852301535056854190006430491443448488395058941599569302505573459222725146013403324233825542783145124732780066409991592126096951641045156683182791644192389719022839536560955216388906990743441965064518391229705267629610832896478653819

d =
4111144651839669277772171877609033551522690840834817273596406957584201472799743906455220656699756497627509760338512996201832613479831659029720596743230199674926239622043951269580678403030197359600163751126765927201759138661340950850562980103501176763063473549971070379283177674043629179846748509791359475252369818489238336093650597495394020144200977653467002787451954442416140081024497370866697873965981187100252787556308524527942730645413364484150751235968280126891877086645570721218132734871257605676218779054283327645942848715460686224529093844342923711889526171100321919870990732677258988776353978484913029455906632905522524074452334387440904675389281675335159058914994189375630268888964816669817672941281149379123744438653265194194850487088595944627023374667345450854134123883210743611846696778309816147063870665868516491654168187488954813253480940343822309337276552445405829272038342921655394030477540585372603782153217

p=204837435777738539616383718974906390712155106805501021662904680667679445331881937792085072472927712958455507781328783602139486432379285479035760664828736840741112106211868854393996571902352368866037146804255216394849805310602429723391495418401942738076653604459316623167044073821002134213440463582350070015216080943726893564286467753184983965700990031780242190306636379364781480887600637606241160579782398703843175343300370409822389835403049533561912006033985339 3

q=201394219741789398735042909690470897630854118673912070322539000282860366590640786430439338910456697618545892030335993317111001933085209031583928173515877923054914215801231322674029662578067908247499395557270004009845352406781669019909741225746269716251556150896327804553782331246028036310844495594905336215820743107416357090644762020180663399176124036151860762618749053743270675276499625666101414642507639879226825296096187719485805654607875002278054851206250548 3

e= 65537

lambda_n = (q-1)*(p-1)//math.gcd(q-1, p-1)
print("(e*d)%lambda_n is: ",(e*d)%lambda_n)

if p*q == n:
    print("p*q equals n\n")

if (math.gcd(e,(q-1)*(p-1))) == 1:
    print("gcd(e,(q-1)(p-1)) equals 1\n")

if 1<e<lambda_n and math.gcd(e, lambda_n) ==1:
    print("1<e<lambda_n and math.gcd(e, lambda_n) ==1\n")
```

```
(e*d)%lambda_n is:  1
p*q equals n

gcd(e,(q-1)(p-1)) equals 1

1<e<lambda_n and math.gcd(e, lambda_n) ==1
```