//ARTHUR VIEGAS EGUIA
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQCZshld8hL993i2PCDNXQkmNFe1quLC+FJyoCdWk
neD9Us8WwB4gP+twSCpAls2FBQxb6Mrg+8E9MeA3l2MN31xAKfqm47EoDWApCmYphoMs9
HeNQS9B15+Yr9WRW4seTTJXWffUZ2a559GhwuxiUvdWG8TPyVxDWK9v1PEdqmS49t3iPAo
weOD0cn6NnZhTXR5exJF6RJ12jJD5qEL4bBJDyCqKegWtg0a5zOFVDSPT6CumXg2eP5lQRb
DEQqWno89vPg6g+CwIBKpDA1SyMxs81G8+D6B0Xqfty4nTEHSi0NYSMDg5HrdZ0UwQJ0DH
VQtyfKMdbgeJN3VaOQyh7hTzEF8RgKnhwber9Umb4wC3lF7quuQ0AeV0gzB11VnIc8bEWW6
wAShN/7qtAGw6UO9VO193ecATaDHD/IBQOm9P5FhmFdCDmMdCatpedf0VSXi9agxSbHL0gs
aXkFgcAtUILFfnpPCj/SmgFc1T/kynAvqRQcBPrc9z2kROeXWC0c=
arthurviegaseguia@Arthurs-MacBook-Air-2.local

Private Key:

According to the RFC 8017, on the appendix C ASN.1 Module page 73, the Private key is a sequence containing:
Version: It might have values 0 or 1, if it is 0 the n has only two factors (p, q) and if it has value 1 it has more than 2 factors.
n, which is a large number that has two prime factors p and q. It is formally known as “modulus,” and it is used to both encrypt and decrypt the message.
e, the public exponent is also part of the public key file. As it is a public exponent everyone has access to it. e < (p-1)(q- 1). It is used to encrypt messages alongside the public key. (formally known as “publicExponent”).
d, which is the private exponent. It is used to encrypt/decrypt messages alongside the private key. A peculiarity is that $ed \equiv 1 mod(p-1)(q-1)$. (formally known as “privateExponent”)
p, which is the first prime to make up n (formally known as “prime1”)
q, which is the second prime to make up n (formally known as “prime2”)
Exponent1, which is d mod (p-1)
Exponent2, which is d mod(q-1)
Coefficient which is (inverse of q) mod p
OtherPrimeInfos, which is additional information and also an optional field.

Decoding the message:
The message starts with a header which says:
-----BEGIN RSA PRIVATE KEY-----
This is in the file because it is in PEM, which is a de facto file format for storing and sending cryptographic keys. Many cryptography standards, including this one use ASN.1 to define their data structures. By using PEM, the contents of the key are converted to base64. It also defines the line header consisting of -----BEGIN, our label which is PRIVATE KEY and -----. The header serves to define the type of message which is being encoded.

Then it is followed by 2396 characters, all of which are either lowercase letters, uppercase letters, numbers +s and /s. The last two characters are ==, which gives further evidence that it was encoded using base64. Let’s use an online decoder to see the hexadecimal bytes in it this section. I just copied the entire content of the private key file and pasted it to Lapo Luchini's

ASN.1 decoder. The decoder converted the Base64 into hex and highlighted the important bits, which I will analyze in detail in the following section.
The first four bytes are 0x30, 0x82, 0x06, 0xE4. The offset of this is 0, as they are the first four bytes of the file.
Well, 0x30 in binary is 00110000. In DER the first byte is the identifier octet, and its bits have special meaning based on their position. Let the rightmost bit be 0 and the leftmost bit be 8. The bits in positions 7 and 8, which are the “tag class” have value 00, which means that the following type is native to ASN.1 (Universal class). What is more, the bit in position 6 has value 1, which means that the following encoding will contain a constructed value. The following 5 bits are the “Tag type” which define the data type being encoded. They have value 0b10000 which in decimal is 16, this means that the following data is of type SEQUENCE. Data of type sequence is like a struct in C. It is important to outline that the data type SEQUENCE is by definition constructed, and that is why byte 6 has to be 1. This is all the information stored in the first byte. the most important part of all of this, for the rest of this assignment, is that the data type is SEQUENCE.
The next byte is 82, which is 10000010 in binary. This is the first length octet. The 8th bit in this octet is 1, which means that the data to be encoded has a definite size. The value of the next seven bits is 0000010 (2 in binary) which means that the 2 next octets determine the size of the SEQUENCE to be encoded. So we will use two octets to determine the size of the entire sequence. In other words, the next two bytes represent the number of bytes of the entire sequence.
The very next byte has the value 06, which is 00000110 in binary, and the following byte is E4 which is 11100100 in binary. Combining both in binary we get 0000011011100100. Converting that value to decimal we get that the size of our entire sequence is going to be 1764 bytes. This checks up, as the decoder says this is the size of the sequence.
So far we have A SEQUENCE, which has a definite size, of 1764 bytes (it took two bytes to describe this value).
Then the file has the encoding of the actual values stored in our private key. This starts at offset 4. Well, after E4 we have byte 02, which in DER means that we want to write an integer. The next octet is 01, which is the size of the block. As we have 02 01, we know that we have an integer that takes up one byte. The bit immediately following that is 00, which is the actual value being encoded. See that ASM.1 follows type->size->value. This means that the first value that is being stored is 00.
This bit will represent the field “value” in our private key. As we know, “value” can be either 0, or 1. If it is 0 it means that the module of our private key (n) is composed of only two large prime integers p, and q. This is what our private key is indicating.
Now let's move on to the next values being encoded in the file. Which happens at offset 7. The very next bit is 02, which indicates another integer. Then, we have the length octet, which is 82. As described above, this means that we will need two bytes to describe the size of the next integer in our private key. The next two bytes, which represent the number of bytes of the integer are 01 81. 01 is in binary 00000001 and 81 is in binary 10000001. So the next integer is of size 0b0000000110000001 or 385 bytes in decimal. The integer will be 0x0099b2195df212fdf778b63c20cd5d09263457b5aae2c2f85272a02756927783f54b3c5b007880ffadc120a9025b361414316fa32b83ef04f4c780de5d8c377d7100a7ea9b8ec4a03580a42998a6

1a0cb3d1de3504bd075e7e62bf56456e2c7934c95d67df519d9ae79f46870bb1894bdd586f133f2
5710d62bdbf53c476a992e3db7788f028c1e383d1c9fa3676614d74797b1245e91275da3243e6a
10be1b0490f20aa29e816b60d1ae7338554348f4fa0ae99783678fe654116c3110a969e8f3dbcf83
a83e0b02012a90c0d52c8cc6cf351bcf83e81d17a9fb72e274c41d28b435848c0e0e47add67453
0409d031d542dc9f28c75b81e24ddd568e43287b853cc417c4602a78706deafd5266f8c02de517
baaeb90d00795d20cc1d7556721cf1b1165bac004a137feeab401b0e943bd54ed7ddde7004da0c
70ff20140e9bd3f91619857420e631d09ab6979d7f45525e2f5a83149b1cbd20b1a5e4160700b54
20b15f9e93c28ff4a68057354ff9329c0bea4507013eb73dcf691139e5d60b47.
This represents the modulo of the primary key, which is n. This number is the result of the multiplication of large prime numbers p and q. It is part of both the private and public keys. In the private key, it is used to both encrypt and decrypt the messages alongside d.
Then the next byte is 02 (It is located at offset 396). This means that the next item in our SEQUENCE is going to be an integer. However, the length octet is 03, which means that we will only need 3 bytes to write the next integer, which is 0x010001. This is the public exponent or e. This is also a part of the public key and should be available to everyone online. A peculiarity of the public exponent is that e < (p-1)(q - 1) and e and (p-1)(q-1)'s greatest common divisor is 1. It is used to encrypt and decrypt messages alongside the n with $M^e mod n$. Where the message is M.

Then at offset 401, we have 02 again. This means that the next item is also going to be an integer. The following octet is 82, which means that the size of the next integer is expressed with two bytes. The following two bytes are 01 80, which are in binary 0000000110000000 or 384 bytes in decimal. So the size of our next integer is 384 bytes. The next integer is the following:

0x523cfcdbcadaa4afd514558a0bb8bb5b88e86199856a63aae21095dbf9d51f739205428036fa7
75486a12e2dc796611a08f5be2e5b92fa68eb9325e8924d62bea36635df747dfb678564d1a90f3f
a30a58c2784607cb38a191bee5af92eb1313cea73f1c3c34839d6a1937cb66fc9aee5e12590f11b
eb335e167cdafb2fabb5156ae584480da761da3945a60e7e8daa3f71886478815cb9fac03af14ff2
18ccd1ade148f05d39927dbfc581200637b475527f7905e3d5189e65102a14cb41934a1a88dc83
3785d976e3941732fda263e41fa41f7e80c81ab2ead30c8167ad37213ffd99df7b3bd25679add6a
8048708d2ec590b35c8d520cc016145658dcf1fe722f9a8ecd7d764ccaac28e835eacfcabc939e2
aabf7791a1b30669c08302993a904db3ba525aca0660f8db6cf132507c52f8d167997e0dd34a05
d8490c3ba9ae67662990a44355e68ec09a123187221944d9d34e53948ab5da45a49413ef17bfd
e5d03ed353afa056c250281474c5c1ad24655b13b8553d0d717c9531a62e31
This is our privateExponent or d. d is only a part of the private key. d is used alongside n to encrypt and decrypt the messages with the private key. Some peculiarities about the private exponent is that $ed \equiv 1 mod(p-1)(q-1)$. To encrypt messages with d you do $M^d mod(n)$, where M is the message to be encrypted/decrypted.
The next integer is at offset 789. Well, as it is an integer, the first bit is 02, indicating its type. The length octet is 0x81, which is in binary 10000001, which indicates that the size of the next number can be expressed with a single octet. The next octet is 0xC1, which means that the next integer has 193 bytes. The integer is the following:

0x00cc9ef196fbb44428955e8ca6d973ba3fc2fc3225aacbcdc8bb38fa7278bb987aff18853b3295
4532835c6d638cec757c08d45e5c903fc5f5d6092f1b3a01fd74104798c71ac5b19bca9cf8c778ef
768432bf4dd31731dc0631ddea73bd9d925a197285a45dc8ae2a0ff81b68e9f43fed7327a2889e2
e59cda1bd3e7c7c02da4e7b6707d2b8a1512b9232fde6189f6071f29cc3cb31b5a10ce5cddbbcb
1b64c66d7e1acf46a70aaf2cfb27a59b6ac2f6eee1447ff749bc932615e94ed45e86839
This represents prime1, or p. It is one of the two prime factors of n, and it is a large prime. Other facts about it is that e < (p-1)(q-1), the greatest common divisor between e and (p-1)(q-1) is 1, and also $ed \equiv 1 mod(p-1)(q-1)$.
Now let's go to the next octet, which is at offset 985. The next octet is 02, so it is encoding another integer. The length octet is 0x81. Just like for the previous prime number, this means that a single octet is needed to describe the size of the integer to be encoded (one byte describes the size of the integer). The size of the integer is 0xC1 or 193 bytes long. The following integer is:
0x00c049adb72afcfbdf397b0fcc15b3d7caff7dc651a3871acd7a57ff00dd491477b74f4f156479e9
071fcc8b9ad44e82f3886430914d7542644ae38c1050b53fada8feb56f628dfdba8e9df6c9bf0a98
62e0b3d4dae73cfec6d5f9aad8bff1e7532c1e86d8433a388d87cf279b735c8fcdacf7a6731271e0
83cd6c53042dede81cf214c0137e145dde870da4505a598632a96a7f2efebf1d87a724ed96675ef
24e9486364163f5a5eeac466bb15caffb2ddac831c90ee2b685e15a4cb5fee70f7f
This represents prime2 or q. This is the other prime factor of n. Just like p, it is a large prime. What is more, e < (p-1)(q-1), and the greatest common divisor between e and (p-1)(q-1) is 1 and also $ed \equiv 1 mod(p-1)(q-1)$.
Now let's move to the next octet. It is in offset 1181. Just like the previous items encoded in this sequence, the first byte of the octet is 02, which means that the data type is integer. What is more, the length octet is 81, so it only takes one octet to write the size of the number to be encoded. The next byte, which is the number of bytes of the integer to be encoded is 0xC1. So the number has 193 bytes. The number encoded is the following:
0x0091bac58ee556de9014c990dca7d41f1a9830eb3a1e6925f74737ee4ea6a3dc9604a61f0529
225c74fc4f506ced09df2666577198f4f72672ce6c1e0b8305b5cffa64336fe2265836ff574a6b2e3b
45aa23465213fbd9fbbcc1b92de81e6c61b61f978b8ca36a7077ee85a267d6efa8b3e126a24030
800c5f4248b87f0a0732196113e0e377b1145136bf704a76d0d36c7400a8054832f62f6c5f6f96f4e
e437d9f54cfcc071e94f17770c268b08bf3004b395089a42feb5c2132ae584a489a29
This is exponent1, which is equal to dmod(p-1). According to Jeff, in class, this is probably used to make some calculations faster.
The next octet is at offset 1377 and has value 02. This means that the next item in this sequence is going to be an integer. The length octet that immediately follows it has value 81, so a single byte is enough to describe the size of the integer to be stored. The byte that describes the number of bytes of the integer to be stored has value 0xC0. So, the integer has 192 bytes. The number is the following:
0x617eba0aae4c3934f48315fe675e99627eea79bc790a8ea77289dd6cc5c6410f762d4b2ed094
13781e426ae265152dc666f84dbbcea74eb365593c722549d0f0af47fe1c6cbf0bd02471a9689f69
b2dc278c66b75b198d20a9eb7e198a31101616bf9fa55568d6b5c40f5fc8acd458c1731ada156c5
bbaf779179c7bf901b077c4b28ab5176f222d2b12daa637393f16bed99584f17b5b70a569cee13c
a6627b3f4951871434c01e64d10bd790e45599e95d2a090478250d7ab67b40a7eee5e7

This is exponent2 and is equal to dmod(q - 1). According to Jeff, it is probably used to speed up calculations.
Now we head to the last integer which is at offset 1572. Just like all the other items in the sequence, the first byte is going to be 02, which describes an integer. The length octet is 81, so a single octet is needed to describe the size of the integer. The next byte, which describes the number of bytes of the integer is C1 or 193 bytes. The next integer is going to be:
0x00994691bab627b6871a38a884023b3d4363583ceb959eed554446ba98f3f5b05605a1117eb3
a6e85aedcbdf7bf9bc011ddb8a3e1a75eaf864264853af8ab4b654e4323bd1bf6903520cf61579c5
f09601482235e4f4a2e3c8fc75c7ccd43871e4ea1a60203408dc4c0cc6809dff62a0ae9a5499883
19f0b833c979337209018f76f379f7b80cbe19193eee2db8885ba31829782edf069b85bded83df2f
ebf3147dde8760b79327cd09fd1ac9997bab1e9f27991496cf71c8709fd5a7dd57f9d2c
This is coefficient, which is defined as (inverse of q) mod p. This is probably also used to speed up some calculations according to Jeff.
The last thing in the file is -----END RSA PRIVATE KEY----- which is there because of PEM, as PEM adds both a header and a footer, the footer is at the end of the body of the private key.

Public Key

Let's start by extracting the public key from the private key file that we already have. According to Giordano, we do so by using the following command:
$openssl\ rsa\ -in\ private.pem\ -pubout$
Which will output the following contents:
-----BEGIN PUBLIC KEY-----
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAmbIZXfIS/fd4tjwgzV0J
JjRXtariwvhScqAnVpJ3g/VLPFsAeID/rcEgqQJbNhQUMW+jK4PvBPTHgN5djDd9
cQCn6puOxKA1gKQpmKYaDLPR3jUEvQdefmK/VkVuLHk0yV1n31GdmuefRocLsYlL
3VhvEz8lcQ1ivb9TxHapkuPbd4jwKMHjg9HJ+jZ2YU10eXsSRekSddoyQ+ahC+Gw
SQ8gqinoFrYNGuczhVQ0j0+grpl4Nnj+ZUEWwxEKlp6PPbz4OoPgsCASqQwNUsjM
bPNRvPg+gdF6n7cuJ0xB0otDWEjA4OR63WdFMECdAx1ULcnyjHW4HiTd1WjkMoe4
U8xBfEYCp4cG3q/VJm+MAt5Re6rrkNAHldIMwddVZyHPGxFlusAEoTf+6rQBsOlD
vVTtfd3nAE2gxw/yAUDpvT+RYZhXQg5jHQmraXnX9FUl4vWoMUmxy9ILGl5BYHAL
VCCxX56Two/0poBXNU/5MpwL6kUHAT63Pc9pETnI1gtHAgMBAAE=
-----END PUBLIC KEY-----
What is more, according to RFC 4716, section 3.4 on page 4, the body of a public key consists of a **string, which is a certificate or a public key format identifier**, and **byte[n] which is the key/certificate data.**

Just like the Private Key, the file is using PEM. By using PEM, the contents of the key are converted to base64. It also defines the line header consisting of -----BEGIN, our label which is PUBLIC KEY and -----. This indicates that the contents that follow will be a public key and the header is required by PEM.

Now I will copy all the contents of the public key and paste it to Lapo Luchini's ASN.1 decoder, which will convert the Base64 into hex and highlight the important bits.

The first 4 bytes of the file (at offset 0) are 0x30, 0x82, 0x01, 0xA2.
As seen before (in the private key example), and further confirmed by Letsencrypt.org, the octet 30 determines that the structure is a SEQUENCE. The next octet is the length octet, which has value 82. This is 10000010 in binary, which means that two octets are enough to describe the size of our entire SEQUENCE. The next two bytes, which tell the byte size of the sequence are 01 A2. Well, 0x01 is in binary 00000001, and A2 is in binary 10100010, so the entire SEQUENCE has size 0000000110100010, or 418 bytes.
Now let's check the contents of the SEQUENCE, which start at offset 4. The first octet of the sequence is 30. So it is another SEQUENCE. Its length octet is 0D, which means that 13 bytes are enough to describe the entire SEQUENCE. Let's analyze this new sequence.
The first octet of this new sequence, which is at offset 6 is 06. This represents data type Object identifier. The length octet of that is 09, meaning that 9 bytes are enough to describe the contents of the object identifier. The contents of the object identifier are:
0x2A 0x86 0x48 0x86 0xF7 0x0D 0x01 0x01 0x01
And according to the decoder, this can be translated to:
1.2.840.113549.1.1.1 (rsaEncryption)
According to letsencrypt.org Object identifiers are globally unique hierarchical identifiers composed of a sequence of integers. They are often used to identify standards, algorithms, certificate extensions, organizations, among others. It is further supported by letsencrypt.org that 1.2.840.113549.1.1.1 identifies RSA Security LLC. This represents the **string** part of our public key. I know this as it is a public key format identifier (it tells that we are using RSA).

Now let's see the other byte of the SEQUENCE (which is inside another SEQUENCE). The next byte starts at offset 17 and is of type 05, which is NULL. Naturally, the length octet is 00 as the information to be stored is NULL. This is the last item in this sequence, which leaves us with the other sequence.

Now we are at our main SEQUENCE. The next octet at offset 19 is 03 which means BIT STRING. Its length octet is 82, so two octets define the number of bytes in the string. The octets that define the number of bytes needed for the string are 01 8f which are in binary respectively 00000001 and 10001111, combining them we get 110001111. Which is in decimal 399 bytes. However, they are immediately followed by byte 00 which means an end of Sequence. According to letsencrypt.org says that BIT STRINGS are used to store unconstructed data, they can be used as a void pointer (which is this case), but this could also hold the signature of a certificate. So, this could be the **string** part of the public key. Further evidence that this is the **string** part of the public key is the data type. The encoder classified it as encrypted data.

The very next octet is at offset 24 and has value 30. Which means it is another SEQUENCE. The length octet of this sequence is 82, so two bytes can hold the size of this sequence. The bytes that hold the size of this sequence are 0x01 and 08A, which are in binary 00000001 and 10001010. So the size of this sequence is 0000000110001010 or 394 bytes.
The first octet of this sequence is at offset 28 and is 02. This means that it is going to be an integer. The length octet is 82, so 2 octets can represent the size of the integer. The two bytes taht represent the number of bytes of the number are 01 81. Well, 01 is in binary 00000001 and

81 is in binary 10000001. So the next integer is of size 0b0000000110000001 or 385 bytes in decimal. The number is:
0x0099b2195df212fdf778b63c20cd5d09263457b5aae2c2f85272a02756927783f54b3c5b00788
0ffadc120a9025b361414316fa32b83ef04f4c780de5d8c377d7100a7ea9b8ec4a03580a42998a6
1a0cb3d1de3504bd075e7e62bf56456e2c7934c95d67df519d9ae79f46870bb1894bdd586f133f2
5710d62bdbf53c476a992e3db7788f028c1e383d1c9fa3676614d74797b1245e91275da3243e6a
10be1b0490f20aa29e816b60d1ae7338554348f4fa0ae99783678fe654116c3110a969e8f3dbcf83
a83e0b02012a90c0d52c8cc6cf351bcf83e81d17a9fb72e274c41d28b435848c0e0e47add67453
0409d031d542dc9f28c75b81e24ddd568e43287b853cc417c4602a78706deafd5266f8c02de517
baaeb90d00795d20cc1d7556721cf1b1165bac004a137feeab401b0e943bd54ed7ddde7004da0c
70ff20140e9bd3f91619857420e631d09ab6979d7f45525e2f5a83149b1cbd20b1a5e4160700b54
20b15f9e93c28ff4a68057354ff9329c0bea4507013eb73dcf691139e5d60b47.
This is in the part **byte** of the public key and it is part of the key per se, remember that the byte part of the key holds the certificate/key data. This represents n. Observe that this is the same number as n in the private key. Therefore, it follows all the same properties.
The next octet, which is at offset 417 is 02. So it is an integer, However, the length octet is 03, so three bytes are enough to hold the integer. The integer will be 0x010001. This is also part of the **Byte** part of the public key. This represents the public exponent e. Observe that it has the same value of the public exponent as the private key. Therefore, it follows the same properties.

Sanity Check

I believe that these integers work as expected. Starting with the public key, the bits in the OBJECT IDENTIFIER are 1.2.840.113549.1.1.1, which according to letsencrypt.org represent RSA encryption. What is more, the data of the key in the public key file (modulus and publicExponent) match with those in the private key file. This makes me believe that n and e are correct.
To check if the values of d is also correct I wrote the following python code:

```
p =
0x00cc9ef196fbb44428955e8ca6d973ba3fc2fc3225aacbcdc8bb38fa7278bb987aff18853b3295
4532835c6d638cec757c08d45e5c903fc5f5d6092f1b3a01fd74104798c71ac5b19bca9cf8c778ef
768432bf4dd31731dc0631ddea73bd9d925a197285a45dc8ae2a0ff81b68e9f43fed7327a2889e2
e59cda1bd3e7c7c02da4e7b6707d2b8a1512b9232fde6189f6071f29cc3cb31b5a10ce5cddbbcb
1b64c66d7e1acf46a70aaf2cfb27a59b6ac2f6eee1447ff749bc932615e94ed45e86839
q =
0x00c049adb72afcfbdf397b0fcc15b3d7caff7dc651a3871acd7a57ff00dd491477b74f4f156479e9
071fcc8b9ad44e82f3886430914d7542644ae38c1050b53fada8feb56f628dfdba8e9df6c9bf0a98
62e0b3d4dae73cfec6d5f9aad8bff1e7532c1e86d8433a388d87cf279b735c8fcdacf7a6731271e0
83cd6c53042dede81cf214c0137e145dde870da4505a598632a96a7f2efebf1d87a724ed96675ef
24e9486364163f5a5eeac466bb15caffb2ddac831c90ee2b685e15a4cb5fee70f7f
p -= 1
q -= 1
d =
0x523cfcdbcadaa4afd514558a0bb8bb5b88e86199856a63aae21095dbf9d51f739205428036fa7
75486a12e2dc796611a08f5be2e5b92fa68eb9325e8924d62bea36635df747dfb678564d1a90f3f
```

```
a30a58c2784607cb38a191bee5af92eb1313cea73f1c3c34839d6a1937cb66fc9aee5e12590f11b
eb335e167cdafb2fabb5156ae584480da761da3945a60e7e8daa3f71886478815cb9fac03af14ff2
18ccd1ade148f05d39927dbfc581200637b475527f7905e3d5189e65102a14cb41934a1a88dc83
3785d976e3941732fda263e41fa41f7e80c81ab2ead30c8167ad37213ffd99df7b3bd25679add6a
8048708d2ec590b35c8d520cc016145658dcf1fe722f9a8ecd7d764ccaac28e835eacfcabc939e2
aabf7791a1b30669c08302993a904db3ba525aca0660f8db6cf132507c52f8d167997e0dd34a05
d8490c3ba9ae67662990a44355e68ec09a123187221944d9d34e53948ab5da45a49413ef17bfd
e5d03ed353afa056c250281474c5c1ad24655b13b8553d0d717c9531a62e31
e = 0x010001
print(e * d % (p * q))
```

The result of this computation is one. I copied the value of p I found and pasted it to variable p. I copied the value q and pasted it to variable q. Then I subtracted 1 form p to get p-1 and subtracted 1 from q to get q-1. I also set e to be equal to the public exponent and d to be equal to the private exponent. Then I did the computation e * d % ((p-1) * (q - 1)) As the result of the operation was 1, the values of p, q, d, and e are correct. So we have all we need to run RSA.

I will also post the code to the repository for it to be inspected.

Sources:

Giordano , L. (2018, April 25). *Public key cryptography: RSA keys*. The Digital Cat. https://www.thedigitalcatonline.com/blog/2018/04/25/rsa-keys/

Moriarty, K., Kaliski, B., Jonsson, J., & Rusch, A. (2016, November 13). *RFC 8017: PKCS #1: RSA cryptography specifications version 2.2*. IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc8017

Thayer, R. L., & Galbraith, J. (2006, November 14). *RFC 4716: The Secure Shell (SSH) public key file format*. IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc4716

A warm welcome to ASN.1 and der. Let's Encrypt. (n.d.). https://letsencrypt.org/docs/a-warm-welcome-to-asn1-and-der/

Wikimedia Foundation. (2022, June 27). *Privacy-Enhanced Mail*. Wikipedia. https://en.wikipedia.org/wiki/Privacy-Enhanced_Mail

Wikimedia Foundation. (2023a, January 28). *X.690*. Wikipedia. https://en.wikipedia.org/wiki/X.690#DER_encoding

Wikimedia Foundation. (2023b, August 31). *ASN.1*. Wikipedia. https://en.wikipedia.org/wiki/ASN.1#Example