Public Key Cryptography: key formats

In this short write-up we will take a quick look at the insides of a public key cryptography key pair. I will use the Python `rsa` module to generate very short keys to make it easier to see what's what. These are what we previously called type C `pkcs1` keys.

Start with the following values

```
e = 17
p = 151
q = 211
n = p*q # 31861
phi = (p-1)*(q-1) # 31500
d = modinv.modinv(e,phi) # 1853
d*e % phi # 1
```

I found the value of d as described in a previous write-up of this series.

In Python:

```
>>> import rsa
>>> pbk = rsa.PublicKey(n=31861,e=17)
>>> pbk
PublicKey(31861, 17)
>>> data = pbk.save_pkcs1()
>>> b64 = data.split(NL)[1]
>>>
```

Here I have defined `NL` to be the standard Unix newline, which I'm having trouble representing in this document. Now we decode the base64

```
>>> b64
u'MAcCAnx1AgER'
>>> import base64
>>> h = base64.b64decode(b64)
```

I can't show you the original hex either. But I can do this:

```
>>> import struct
>>> for c in h:
... print hex(ord(c)),
...
0x30 0x7 0x2 0x2 0x7c 0x75 0x2 0x1 0x11
```

An alternative, equivalent way would be:

```
 >>> t = (struct.unpack("B",c)[0] for c in h)
```

The advantage of `struct.unpack` is it allows you to do multibyte representations like `int64`.

The value of t is `48, 7, 2, 2, 124, 117, 2, 1, 17`.

The way the layout works is that each region consists of a byte describing the type of value, then the size of the value, then the value itself. For example, `2,2,124,117` is a value of type integer of length 2 bytes, and the value is `124*256 + 117` which is equal to

```
>>> 124*256 + 117
31861
```

This is just n. Similarly, `2,1,17` is e. The leading terms `48,7` obviously describe the format.

If we generate a private key

```
pk = rsa.PrivateKey(n=31861,e=17,p=151,q=211,d=1853)
```

and export the data in the same way, the base64 is

>>> b64 u'MCACAQACAnx1AgERAgIHPQICAJcCAgDTAgE1AgIArQIBSQ=='

And the decoded ints are

`48 32` header

`2 1 0` value 0 ??

`2 2 124 117` n, as before

`2 1 17` e, as before

`2 2 7 61` This is `1853`, that is, d.

`2 2 0 151` p

`2 2 0 211` q

`2 1 53`

`2 2 0 173`

2 1 73

The last three values are 53, 173 and 73. I believe from looking at the XML format that these are DP, DQ and InverseQ, but I am not sure yet what they are or how they are used.

0.1

As long as we're at it, let's look at what we called type A `ssh-rsa` keys. This is the type that I analyzed in my blog posts.

`telliott99.blogspot.com/2011/08/dissecting-rsa-keys-in-python-2.html`

I split off the first and the last part (separated by spaces) by hand.

In Python:

```
>>> import utils
>>> data = utils.load_data("kf.pub")
>>> data
'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC8u8w9..
>>> data = data.split(" ")[1]
>>> data
'AAAAB3NzaC1yc2EAAAADAQABAAABAQC8u8w9K4aRPglzdPj..
>>> import base64
>>> s = base64.b64decode(data)
>>> fn = "out.txt"
>>> FH = open(fn,"wb")
>>> FH.write(s)
>>> FH.close()
```

```
$ hexdump -C out.txt
00000000 00 00 00 07 73 73 68 2d 72 73 61 00 00 00 03 01 |....ssh-rsa.....|
..
```

ssh-rsa again!

```
00 00 00 03 is a size, 3 bytes
01 00 01 is an integer = 1 * 16**4 + 1 = 65537
00 00 01 01 is a size, 257 bytes
```

that's how many there are starting from 00 bc

In Python, again:

```
>>> data = load_data('out.txt')
>>> data = data[22:]
>>> len(data)
257
```

```
>>> L = list(data)
>>> L.reverse()
>>> import struct
>>> L2 = [struct.unpack("B",c)[0] for c in L]
>>>
```

```
>>> b = 256
>>> n = 0
>>> for i,x in enumerate(L2):
...  x *= b**i
...  n += x
...
>>> n
2382540788442484304389274272494727..
```

check it with rsa

```
>>> import rsa
>>> k = utils.load_data('kf')
>>> pk = rsa.PrivateKey.load_pkcs1(k)
>>> pk.n
2382540788442484304389274272494727..
>>> n == pk.n True
>>>
```

It matches!