Public Key Cryptography: example

In this short write-up we will go through an example of using public-key cryptography. Let's explore a simple example of file encryption using `ssh` and `openssl`. The demo uses OS X Terminal but it would be very similar on Linux.

The first step is to generate an RSA key pair. Normally one would do something like:

```
ssh-keygen -t rsa -C "name@host"
```

where -C is a comment (for convenience of the user only, not transmitted), -t is the type of key, -b is number of bits (the default for RSA is 2048), and -m is the format (the default is RFC4716 but PEM is also an option).

Since I am not currently using RSA keys, I could just write them to the default filepaths: `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub` .

Instead, for this demo I'm going to put them on the Desktop:

```
ssh-keygen -t rsa -C "te" -f ./kf
```

At the prompt, I enter a passphrase: `abcde`

The output is:

```
Your identification has been saved in ./kf.
Your public key has been saved in ./kf.pub.
The key fingerprint is:
05:f4:3a:1d:b6:48:7e:2f:1f:e5:a0:c7:bd:c6:bc:d8 te
The key's randomart image is:
+--[ RSA 2048]----+
|        .o       |
|          o      |
|         . =     |
|        o * o    |
|         S + . . |
|          o + =  |
|           o =oo |
|            + ++.|
|             o.E.|
+-----------------+
```

There are two files: `kf` and `kf.pub`. We can check the fingerprint like this:

```
ssh-keygen -lf kf
```

The output is:

```
2048 05:f4:3a:1d:b6:48:7e:2f:1f:e5:a0:c7:bd:c6:bc:d8 te (RSA)
```

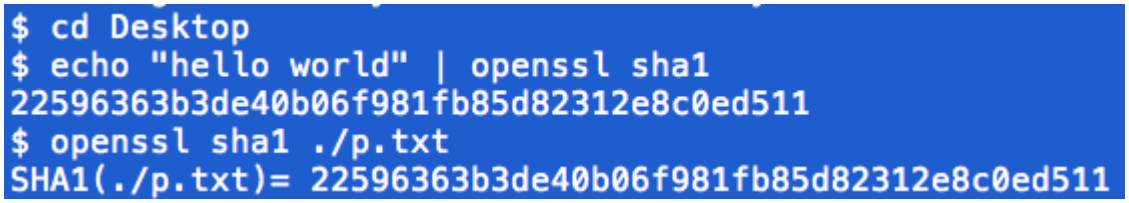
Now for an example:

```
echo "hello world" > ~/Desktop/p.txt
```

The next step uses `openssl`. It can do a lot of things, for example, digests. Here are some approaches

```
echo "hello world" | openssl sha1
echo "hello world" | openssl md5
openssl md5 ./p.txt
```

I'm having trouble typesetting the output, but you can check it out in this screenshot:



`openssl` can also do base64 encoding:

```
openssl base64 -in p.txt -out b.txt
openssl base64 -d -in b.txt
```

`en.wikipedia.org/wiki/Base64`

We first use it to re-format the public key as PEM:

```
openssl rsa -in kf -pubout > ./kf.pem
```

Now we can write this short message:

```
echo "hello world" >  /Desktop/p.txt
```

Encrypt

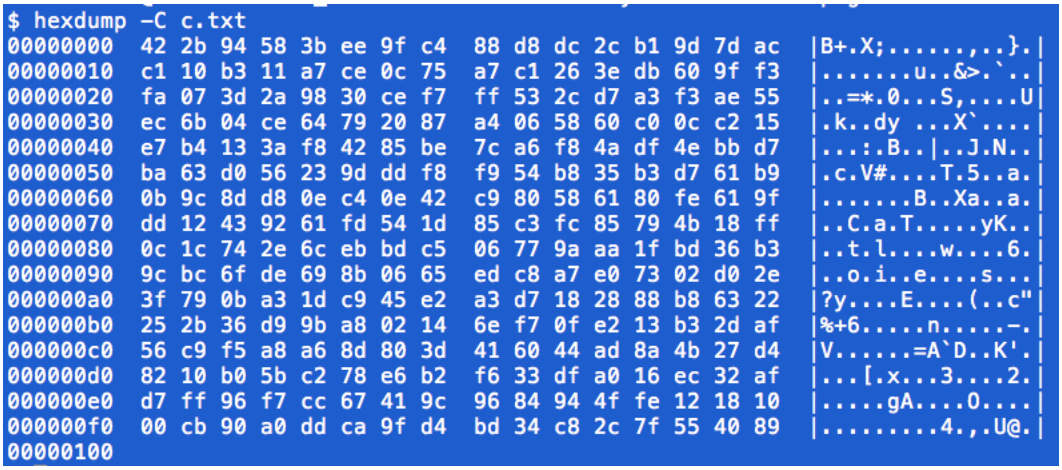
```
cat p.txt | openssl rsautl -encrypt \
-pubin -inkey kf.pem > c.txt
```

The `-pubin` option means to encrypt with the public key We can also specify the input and output files like this:

```
openssl rsautl -encrypt -in p.txt \
-out c.txt -pubin -inkey kf.pem
```

Take a look

```
hexdump -C c.txt
```



```
$ hexdump -C c.txt
00000000  42 2b 94 58 3b ee 9f c4  88 d8 dc 2c b1 9d 7d ac  |B+.X;......,..}.|
00000010  c1 10 b3 11 a7 ce 0c 75  a7 c1 26 3e db 60 9f f3  |.......u..&>.`..|
00000020  fa 07 3d 2a 98 30 ce f7  ff 53 2c d7 a3 f3 ae 55  |..=*.0...S,....U|
00000030  ec 6b 04 ce 64 79 20 87  a4 06 58 60 c0 0c c2 15  |.k..dy ...X`....|
00000040  e7 b4 13 3a f8 42 85 be  7c a6 f8 4a df 4e bb d7  |...:.B..|..J.N..|
00000050  ba 63 d0 56 23 9d dd f8  f9 54 b8 35 b3 d7 61 b9  |.c.V#....T.5..a.|
00000060  0b 9c 8d d8 0e c4 0e 42  c9 80 58 61 80 fe 61 9f  |.......B..Xa..a.|
00000070  dd 12 43 92 61 fd 54 1d  85 c3 fc 85 79 4b 18 ff  |..C.a.T.....yK..|
00000080  0c 1c 74 2e 6c eb bd c5  06 77 9a aa 1f bd 36 b3  |..t.l....w....6.|
00000090  9c bc 6f de 69 8b 06 65  ed c8 a7 e0 73 02 d0 2e  |..o.i..e....s...|
000000a0  3f 79 0b a3 1d c9 45 e2  a3 d7 18 28 88 b8 63 22  |?y....E....(..c"|
000000b0  25 2b 36 d9 9b a8 02 14  6e f7 0f e2 13 b3 2d af  |%+6.....n.....-.|
000000c0  56 c9 f5 a8 a6 8d 80 3d  41 60 44 ad 8a 4b 27 d4  |V......=A`D..K'.|
000000d0  82 10 b0 5b c2 78 e6 b2  f6 33 df a0 16 ec 32 af  |...[.x...3....2.|
000000e0  d7 ff 96 f7 cc 67 41 9c  96 84 94 4f fe 12 18 10  |.....gA....O....|
000000f0  00 cb 90 a0 dd ca 9f d4  bd 34 c8 2c 7f 55 40 89  |.........4.,.U@.|
00000100
```

Decrypt with the private key

```
$ openssl rsautl -decrypt -in c.txt -inkey kf
Enter pass phrase for kf:
hello world
```

We can use the keys the other way around, encrypting with the private key and decrypting with the public one:

The options are `-sign` and `-verify`

```
openssl rsautl -sign -in p.txt -out c.txt -inkey kf
openssl rsautl -verify -in c.txt -pubin -inkey kf.pem
```

With a larger message to encrypt, we have to be more sophisticated

`www.czeskis.com/random/openssl-encrypt-file.html`

Generate 256 random bytes (the source does it in two steps, so that's what we'll do:

```
openssl rand 128 > k1.bin
openssl rand 128 > k2.bin
```

```
cat k1.bin k2.bin > k.bin
```

To encrypt using AES with 256 bits and CBC mode:

```
openssl enc -aes-256-cbc -salt -in p.txt \
-out c.txt -pass file:./k.bin
```

To decrypt:

```
openssl enc -d -aes-256-cbc -in c.txt \
-out m.txt -pass file:./key.bin
```

In practice, use the RSA key to send this key to your cohort. You should also verify the digest (hash) of the data you send, or sign it with your private key (see above)

I have written quite a bit about the structure of RSA key files. See:

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

There are four posts, and I explore the use of Python modules to do encryption.