# Symmetric Encryption

Encryption Algorithms can be either:

* Symmetric Encryption Algorithms
  + Secret key encryption
  + same key used on both sides to encrypt and decrypt
* Asymmetric Encryption Algorithms
  + Public key encryption
  + different key pairs used to encrypt and to decrypt

We will use OpenSSL to complete Asymmetric encryption exercises in the this practical.

# Asymmetric Encryption exercises

## Preparation

1. You should sit beside someone that you can work with as you will need a partner to complete some of the exercises.
2. You should probably create a folder to store your files in and navigate to that folder.
3. To complete the encryption exercises you will need to first complete the following:
   1. Generate a public/private key pair for the RSA algorithm. You will first generate a Private key and then extract the Public key from it. The command below shows the syntax to create the Private Key.

$ openssl genrsa -aes128 -out rsa\_private.pem 2048

Using the command above you will be prompted for a protecting password. If you don't want your private key to be protected by a password, remove the flag “-aes128” from the command line. The number 2048 is the size of the RSA key, in bits.

* 1. The public key should be extracted from this to allow it to be shared with others. You can extract the public key by entering the following:

$ openssl rsa -pubout -in rsa\_private.pem -out rsa\_public.pem

You now should have two files, a private one and a public one.

* 1. A selection of plaintext files which you can use to practice encryption on.

Use the Linux editor mentioned last week to create 2 plaintext files containing some text. Save them in the directory where you saved the keys created in the last step.

## RSA Encryption & Decryption

1. Exchange public keys with your partner. Encrypt your first plaintext file using your partner’s public key.

$ openssl rsautl -encrypt -inkey rsa\_public.pem -pubin -in plaintext.txt -out ciphertext.bin

*(The “-pubin” option indicates that you are using a public key from the specified input key file; the default is to look for a private key).* You should now have an encrypted file that you should exchange with your partner.

1. Decrypt the ciphertext file you received from your partner using your own private key:

$ openssl rsautl -decrypt -inkey rsa\_private.pem -in ciphertext.bin -out decrypted.txt

1. Check the contents of the decrypted ciphertext. What security service does this encryption/decryption exercise support?

$ cat decrypted.txt

## RSA signing & verification

1. Encrypt (sign) the second plaintext file you created earlier using your private key:

$ openssl rsautl -sign -inkey rsa\_private.pem -in plaintext.txt -out plaintext\_sig.bin

1. Give your partner the signed file and receive a signed file from them (which they have encrypted (signed using their own private key).
2. Decrypt (verify) the signed file using your partner’s public key.

$ openssl rsautl -verify -inkey rsa\_public.pem -pubin -in plaintext\_sig.bin

*The verified original text will be output to the console. This can alternatively be directed to a file using “-out filename”.*

## Performance of key generation

1. Create several (say 5) RSA key pairs of different sizes. Time how long the key generation takes. As the time taken for key generation varies according to the random numbers chosen, try each 3 times and take the average. Suggested key sizes for your tests: 512 bits, 1024 bits, 2048 bits, 4096 bits, 8192 bits.

To time a key generation (e.g. 2048 bits), use the time command:

**$ time openssl genrsa -out testkey.pem 2048**

real 0m0.366s

user 0m0.364s

sys 0m0.002s

(*user* time is what matters)

Plot a graph showing the relationship between key length and time required to generate the keys.