B5 - Computer Numerical Analysis

B-CNA-500

my_PGP

A Song of Ciphers and Primes

my_PGP

binary name:	mypgp
language:	everything working on "the dump"
compilation:	via Makefile, including re, clean and fclean rules
build tool:	no need here

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

Alice and Bob want to exchange data without Eve to notice and get access to the messages.

Wait, no, this is not right. Let's start this all over again…

Arya Stark is in a secret mission in the free city of Pentos. She has gathered vital information that she must transmit to Brienne of Tarth back in Westeros. Alas, Euron Greyjoy and his fleet of ironborn roams the Narrow Sea and they search from top to bottom all the ships crossing the border. If Arya tries to send her message this way, her precious information will be known by the Evil Queen in King's Landing in no time.

Fortunately, the maesters of Pentos have kept alive the old Valyrian art of concealing messages, known as cryptography. The knowledge is all there, but they need you to actually implement these ancient algorithms and finally bring peace to the Seven Kingdoms.

— What do we say to the God of Death?
— 074f154d54010b41164045

Nomenclature

For ease of use, we will represent the data to be ciphered and the keys to cipher them as sequence of bytes in hexadecimal.
For example, the string **"4142434445464748"** represents an array of 8 bytes: **[65, 66, 67, 68, 69, 70, 71, 72]**.
Our representation is **little endian**, that is the lower byte is on the lower address (on the left). For example a 32 bits integer with the value 1 is represented as: **"01 00 00 00"**. The number 0x12345678 will be represented as: **"78 56 34 12"**.

Usage

Terminal

```
~/B-CNA-500> ./mypgp -h
USAGE
      ./mypgp [-xor | -aes | -rsa | -pgp] [-c | -d] [-b] KEY
      the MESSAGE is read from standard input
DESCRIPTION
       -xor       computation using XOR algorithm
       -aes       computation using AES algorithm
       -rsa       computation using RSA algorithm
       -pgp       computation using both RSA and AES algorithm
       -c         MESSAGE is clear and we want to cipher it
       -d         MESSAGE is ciphered and we want to decipher it
       -b         block mode: for xor and aes, only works on one block
                  MESSAGE and KEY must be of the same size
       -g P Q     for RSA only: generate a public and private key
                  pair from the prime number P and Q
```

All your symmetric algorithms must accept "-b" (for **block mode**) as an option. In block mode, only one block will be treated. A block is of the same size of the key. It means the message must be of the same length as the key.
Without the "-b" modifier, your algorithm must work in **stream mode** : the message to cipher/decipher can be of any length.

SYMMETRIC ENCRYPTION

Symmetric algorithms use the same key for ciphering and deciphering. You have to implement two of these algorithms, one is very simple but either impractical or easy to crack, the other one is more robust and practical (but harder to implement).

SIMPLE XOR

To hide the message, Arya proposes to compute a XOR of the message with some random data called key. You have to code a program that takes a message m and a key k, and returns

$$c = m \oplus k$$

Terminal

```
~/B-CNA-500> echo "68656c6c6f20776f726c64" > message
~/B-CNA-500> cat message | ./mypgp -xor -c -b 766572792073656372657 > ciphered
~/B-CNA-500> cat ciphered
1e001e154f53120c000910
~/B-CNA-500> cat ciphered | ./mypgp -xor -d -b 766572792073656372657
68656c6c6f20776f726c64
```

A simple XOR is a good cryptosystem, but only if the key is as long as the message and is used only one time! Which is not very practical.

AES

Arya could encrypt her message using XOR but in this case she will need a key as long as her message. The key needs to be secretly shared between Arya and Brienne. If she had such a way to transmit a long secret key, she could just as well send directly her message through this secure system. Therefore they need a better algorithm to cipher their messages using a smaller key while being stronger than XOR.
She chooses to use the AES algorithm, with 128 bit key.
Your AES implementation must be able to work in block mode (-b) or in stream mode.

Here is an example for block mode:

Terminal

```
~/B-CNA-500> echo "c2486f4796f0657481a655c559b38aaa" > message
~/B-CNA-500> cat message | ./mypgp -aes -c -b 6b50fd39f06d33cfefe6936430b6c94f >
ciphered
~/B-CNA-500> cat ciphered
0fc668acd39462d17272fe863929973a
~/B-CNA-500> cat ciphered | ./mypgp -aes -d -b 6b50fd39f06d33cfefe6936430b6c94f
c2486f4796f0657481a655c559b38aaa
```

Asymmetric Cryptographic Algorithm

That's great ! Arya and Brienne now have a secure way to exchange messages !
But to do so they need a shared secret key. If one of them chooses a key and sends it to the other by boat, Euron will certainly intercept it. He then will be able to read Arya secret message when she sends it.

So they need another method to exchange their key without a prior shared secret, and without Euron being able to interfere.
Here comes to the rescue the asymmetric cryptography, which will allow anybody to cipher a text with Brienne's public key, which only Brienne can decipher using her private key!

RSA

Code a program that, given 2 large prime numbers, will display public and private key according to the RSA cryptosystem.

Here is an example with small keys:

```
Terminal
~/B-CNA-500> ./mypgp -rsa -g d3 e3
public key: 010001-19bb
private key: 81b3-19bb
~/B-CNA-500> echo "2a" > message
~/B-CNA-500> cat message | ./mypgp -rsa -c 010001-19bb > ciphered
~/B-CNA-500> cat ciphered
b104
~/B-CNA-500> cat ciphered | ./mypgp -rsa -d 81b3-19bb
2a
```

And here is another one with larger keys:

```
Terminal
~/B-CNA-500> ./mypgp -rsa -g 4b1da73924978f2e9c1f04170e46820d648edbee12ccf4d4462a\
f89b080c86e1 bb3ca1e126f7c8751bd81bc8daa226494efb3d128f72ed9f6cacbe96e14166cb
public key: 010001-c9f91a9ff3bd6d84005b9cc8448296330bd23480f8cf8b36fd4edd0a8cd925d\
e139a0076b962f4d57f50d6f9e64e7c41587784488f923dd60136c763fd602fb3
private key: 81b08f4eb6dd8a4dd21728e5194dfc4e349829c9991c8b5e44b31e6ceee1e56a11d66e\
f23389be92ef7a4178470693f509c90b86d4a1e1831056ca0757f3e209-c9f91a9ff3bd6d84005b9cc8\
448296330bd23480f8cf8b36fd4edd0a8cd925de139a0076b962f4d57f50d6f9e64e7c41587784488f9\
23dd60136c763fd602fb3
~/B-CNA-500> echo "c1fa29d40054f3fcb1c15fe4d63d3887" > message
~/B-CNA-500> cat message | ./mypgp -rsa -c 010001-c9f91a9ff3bd6d84005b9cc844829633\
0bd23480f8cf8b36fd4edd0a8cd925de139a0076b962f4d57f50d6f9e64e7c41587784488f923dd601\
36c763fd602fb3 > ciphered
~/B-CNA-500> cat ciphered
dc0bd7367d04e5a9e9e14467ff38de0625b3cfa5aabbe86def48bfc93e97aab713d70abf83d263a6dd\
6570c6d297cc44bad2e0dd2cf7b4c3e0a9749d68ca11a8
~/B-CNA-500> cat ciphered | ./mypgp -rsa -d 81b08f4eb6dd8a4dd21728e5194dfc4e349829\
c9991c8b5e44b31e6ceee1e56a11d66ef23389be92ef7a4178470693f509c90b86d4a1e1831056ca07\
57f3e209-c9f91a9ff3bd6d84005b9cc8448296330bd23480f8cf8b36fd4edd0a8cd925de139a0076b\
962f4d57f50d6f9e64e7c41587784488f923dd60136c763fd602fb3
c1fa29d40054f3fcb1c15fe4d63d3887
```

Pretty Good Privacy

RSA solves the problem of exchanging secrets, but it is very slow. To make a complete application convenient to use for sharing secret messages, you have to combine the use of **asymmetric** and **symmetric** cryptography:

- Pick a random symmetric key
- Cipher the symmetric key with the recipient public key
- Cipher your message with the symmetric key
- Produce a final file which contains both the ciphered symmetric key and the ciphered message
- Implement the deciphering process from this file and the private key

```
Terminal
~/B-CNA-500> ./mypgp -pgp -c [PUB_KEY] < MESSAGE_FILE
[CIPHERED MESSAGE]
~/B-CNA-500> ./mypgp -pgp -d [PRIV_KEY] < CIPHERED_FILE
[MESSAGE]
```

What you have built gets close to the PGP cryptosystem.
With this, Arya could ask Brienne to generate a pair of keys and send back the public one. Arya can use this public key to cipher her long message with this cryptosystem, and send Brienne the message only she can read. Even if Euron intercepts it, he wouldn't be able to read it.

Bonus

As bonuses, you could:

- implement a tool or an option to generate the RSA keys.
- implement a way to **sign** the messages to guarantee who wrote them.

Appendix

As we don't ask you to generate large prime numbers for your RSA keys, here you will find some prime numbers for different key sizes.

Four 8 bits primes

```
d3, e3, e9, f1.
```

Four 32 bits primes

```
13d3c01d, 0b1e8a1e, d3dd082f, 0bfe6c0d.
```

Four 256 bits primes

```
4b1da73924978f2e9c1f04170e46820d648edbee12ccf4d4462af89b080c86e1,
bb3ca1e126f7c8751bd81bc8daa226494efb3d128f72ed9f6cacbe96e14166cb,
d5da1a8d443812956185f9fe2d8696ea4959d3415967c7a8c5fec34bf7dd53e1,
4fbd36c46bd3fa40ebcef3447a4e2f2f16a6cee884bf889fd58ec5bca6024fe1.
```

Two 1024 bits primes

```
d7c9d58c694fe7ad5d77d888c98d71e7f6a58b4f4dc90582668fd28c0bc20f51
c667ba7b70cb94842006eb5b223065346f7a6bb307ef572fee882c8ef420410b
8b8fa8278ae6a300e63123b28ba1d47259bc308827fcd509585bcee1d98b461f
9eff9c20559540b8c0d6036eff7caf0107d935ecefd5faab87b802bf74c041c8

e9dfe6b53248c4dc0f391fdda5694bd9f68e111dac5e921a942a157ec92431dc
4833e1a327d36cebcc4ac9b76f2ac2a643db8a04c14963759a800f75915de3ff
beccf86118923b388b352f7e3d20edfd5bb6609fd0b6416da6a56050b000ae9e
14065f06f46ccfaa84c755689e8d95525bb1de8b8a43d380f4db68baac92e0bf
```

Larger primes

If you want more or larger primes, you can use this command line for OpenSSL:

```
Terminal
~/B-CNA-500> openssl genrsa 2048 | openssl rsa -text
```

OpenSSL prints it's values in a **big endian** notation, if you want to use them you will have to convert them according to our nomanclature described at the begining of this document