
------------------Exercice 1 Résolution du problème de primalite---------
----

Tout au long du code, on modifiera le type long par le type uint16_t et uint32_t en incluant la bibliotheque <stdint.h> (Cette utilisation était provisoire du fait de plusieurs debugs dans la fonction long_extended gcd)

===
===
===
===
===
=======

-----------------Implementation par une methode naïve--------------------
----

----

q1.1 La complexité de la fonction est en O(p). Elle prend un entier impair ; va enumerer tout les entiers naturels compris entre 3 et p-1 ; puis renvoyer 1 si p est premier ; 0 sinon. Ce qui correspond à une methode naïve.

q1.2 le plus grand nombre premier qui est testé en moins de 2 secondes avec long plus_grand() est 213133 (Inclusion de la bibliotheque time.h et de srand(time(NULL)) pour generer des valeurs non deterministes est
une precaution à prendre)

-----------------Exponentiation Modulaire Rapide-------------------------
----

----

q1.3. La complexite de la fonction long modpow_naive(long a, long m, long n) est en O(m). Le corps de la boucle s'effectue m fois, on realise 2 instructions elementaires, plus precisement, le mod n.

Explication en pseudo algorithme :

```
z = a
                            res = z
pour i = 1 jusqu'a m faire : z = z*a
                            res = res % n
```

Retourner z
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||||||||||||
On effectue m - 1 multiplications dans ce cas precis
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||||||||||||
q 1.4. La complexite de la fonction par methode recursive est en O(log(n))

Ecriture du code en pseudo algorithme :

```
y = x
z = x
tant que m > 1  faire :
            m = |n/2| (la partie entiere inferieur)
            si n > 2*m alors z = z *x
                sinon n = m
z = z*y
Retourner z
```

|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||||||||||||
q1.5 la courbe de la fonction modPow doit etre logarithmique du fait de sa complexité.
Cependant,la courbe representative de modpow_naive est lineaire ; donc y = n

===
===
===
===
===
=======

-------------------Implementation du test de Miller-Rabin----------------
----

Le test de Miller-rabin est un algorithme probabliste. Soit p un entier naturel impair de la forme 2^s*d +1|/|rm| avec s et d des entier naturel.
L'idée de cet algorithme est d'arriver ; avec un nombre d'essais finis, de conjecturer que p est premier.
Dire que a est un temoin de Miller pour p signifie que :
    * le reste de la division euclidienne par p de a^p est differente de 1|/|rm|
    * le reste de la division euclidienne par p de a^2^r-d est differente de -1|/|rm| avec r prenant les valeurs de 0 à s-1

q.1.7. On pourrait supposer q'une borne superieure de l'agorithme testPrimaliteMillerrabin serait en (1|/|rm|/4)^k
L'algorithme de Miller - Rabin annonce qu'un nombre n est premier avec une probabilite d'erreur inferieur à 1|/|rm|/4 pour tout n > 9

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||||||||||||||||||||||||||||||||||

On decompose l'algorithme de miller-Rabin de cette maniere :

    1|/|rm|. Ecrire n−1=2^k*m, ou m est impair
    2. Choisir un entier aléatoire a, 1≤ a ≤ n − 1
    3. b= a^m (mod n)
    4. Si b = 1|/|rm|(mod n) alors retourner ”n est premier”
    5. Pour i de 0 jusqu'à k−1 Si b = −1 (mod n) alors retourner ”n est premier”
        Faire : Si b = −1 (mod n) alors retourner ”n est premier”
        sinon b = b^2 (mod n)

    6. retourner ”n est composé”

------------------Exercice 2 implementation du protocome RSA-------------
----

Implementation de l'algorithme d'Euclide.   Euclide(a,b)

1   r0 = a
2. r1 = b
3. m = 1/rm
4. tant que rm != 0 :
                faire :   qm = |rm-1|/|rm|
                            rm+1 = rm-1 -qm*rm
                            m = m+1
5 retourner rm

L'algorithme d'Euclide calcule le pgcd de deux entier a et b en effectuant au maximum
|2*log(M)| + 1 divisions entieres ; ou M est le maximum entre a et b. M = max(a,b)

==
==
==
==
==
=====

------------------Declaration securisées---------------------------------
----

------------------Exercice 3 Manipulations de Structures Securisées------
----

------------------Manipulation de cles-----------------------------------
----

-----

q.3.1. Les deux entiers long declares dans la structures correspondent ou bien à la cle privee ou à la cle publique.
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||

q.3.2. La cle est supposé alloué. Donc, il est inutile de passer par un mallocce qui evite d'autre part les fuites memoires
        dans notre programme.(Utilisation de gdb recommandé sur cette question)
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||

q.3.3. On decide de generer un nombre premier p de taille 5000 entre low size puis size up
        De meme avec q.
        Tant que ces nombres premiers sont identiques, on reeitere la meme instruction.
        Puis on genere une cle publique et privee et on ajoute une condition : u < 0 afin de calculer s*u mod t =1
        Enfin, on termine par initialiser la cle publique puis la cle privee
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||

===
===
===
===
===
=====

----------SIGNATURES----------
---

q 3.5. (en cours)