201-P15-MO – H2013 – François Strasbourg

Travail pratique 2

Théorie des nombres et cryptographie

1. **Les nombres premiers.** Afin de déterminer si un nombre n est premier, il suffit de tester si le reste de la division de n avec chacun des entiers entre 2 et $\lfloor\sqrt{n}\rfloor$ donne zéro. Voici donc le pseudo-code d'une fonction qui fait ce test :

```
fonction estPremier(n : entier > 1) retourne booléen // vrai si n est premier, faux sinon
  si n ≤ 1 alors
    lancer exception d'argument illégal;
  fin si
  pour i de 2 à plancher(racineCarrée(n)) faire
    si n mod i = 0 alors
      retourner faux;
    fin si
  fin pour
  retourner vrai;
fin fonction
```

On peut alors utiliser cette dernière fonction dans des algorithmes afin de trouver des nombres premiers. Voici le pseudo-code d'une fonction qui trouve le premier nombre premier à partir d'un nombre de départ s donné en entrée :

```
fonction prochainPremier(s : entier > 1) retourne entier // Le plus petit nombre premier ≥ s
  si s ≤ 1 alors
    lancer exception d'argument illégal;
  fin si
  n ← s;
  si n ≠ 2 et n est pair alors
    n ← n + 1;
  fin si
  tant que non(estPremier(n)) faire
    n ← n + 2;
  fin tant que
  retourner n;
fin fonction
```

a) Implémenter et tester ces fonctions en *Java*. Note: Nous recommandons d'utiliser le type **long** pour tous les entiers de ce travail, dans le but de travailler si désiré avec les plus grands entiers disponibles (simplement) en *Java* tout en évitant les débordements.

b) Choisir arbitrairement deux entiers de quatre chiffres, et utiliser la fonction trouvée en a) pour trouver deux grands nombres premiers p et q qui vous seront utiles plus tard. Il faut que le produit $n = pq$ soit supérieur à 27272727. Afficher ces deux entiers dans le programme principal.

2. **L'algorithme d'Euclide étendu.**

Vous devez implémenter et tester l'algorithme d'Euclide étendu. Il serait adéquat de valider que les nombres donnés en entrée sont des entiers supérieurs à 1. Afin de faciliter la sortie des 3 valeurs désirées : pgcd(a, b) ainsi que x et y tel que $a\,x + b\,y$ = pgcd(a, b). Dans le programme principal, afficher une combinaison linéaire qui démontre que votre fonction marche bien.

3. **L'inverseur Modulo *n*.**

a) Étant donné a, il est simple de déterminer un inverse de a (mod m). Il faut se souvenir que cet inverse existe et est unique (mod m) si pgcd(a, m) = 1. En fait, il s'agit du x tel que $a\,x + y\,m = 1$. Il est possible que la valeur calculée soit négative, auquel cas on peut trouver son équivalent positif entre 0 et (m–1), puisqu'on travaille (mod m). En conclusion, il s'agit donc essentiellement d'appliquer l'algorithme d'Euclide étendu, avec les vérifications et post-traitements suggérés dans l'algorithme suivant (qu'il faut tester, bien entendu) :

```
fonction inverseModulo(a, m : entiers > 1) retourne entier // Un inverse de a (mod m)
  si  a ≤ 1  ou  m ≤ 1  ou  pgcd(a, m) ≠ 1  alors
    lancer exception d'argument illégal;
  fin si
  exécuter Euclide étendu et récupérer x0
  tant que  x0 < 0  faire
    x0 ← x0 + m;
  fin tant que
  retourner  x0;
fin fonction
```

b) À l'aide des valeurs de $n = pq$ déterminées au numéro 2, calculer $\phi(n)$. Trouver une valeur e de 4 chiffres telle que pgcd(e, $\phi(n)$) = 1. Pour ce faire,vous pouvez déterminer arbitrairement un nombre s de 4 chiffres et ensuite utiliser le pseudo-code suivant pour déterminer rapidement une valeur de e qui satisfait pgcd(e, $\phi(n)$) = 1 :

```
fonction trouverE(s, phi : entiers > 1) retourne entier
                                  // Le plus petit nombre e ≥ s tel que pcgd(e, phi) = 1
  si  s ≤ 1  ou  phi ≤ 1  alors
    lancer exception d'argument illégal;
  fin si
  e ← s;
 tant que  non(pgcd(e, phi) ≠ 1)  faire
    e ← e + 1;
  fin tant que
  retourner e;
fin fonction
```

c) Utiliser la fonction créée en a) pour trouver $d = e^{-1} \pmod{\phi(n)}$. Vous devez vérifier que $de \equiv 1 \pmod{\phi(n)}$ devez ensuite communiquer au prof les valeurs d et e. Votre clé publique consiste des valeurs (n, e). Votre clé privée est (p, q, d).

4. **L'exponentiation rapide.** Voici un pseudo-code pour implémenter l'exponentiation rapide présentée au cours (qui avait été présentée sous forme de tableau). Il vous faut implémenter et tester cette fonction. Vous pouvez également implémenter facilement l'algorithme en *Excel*, puisqu'il s'agit d'un algorithme présentable sous forme de tableau.

```
fonction exponentiationRapide(a : entier ≥ 1, b : entier ≥ 0, m : entier > 1) retourne entier
                                                    // Retourne a^b (mod m)
  si a < 1 ou b < 0 ou m ≤ 1 alors
    lancer exception d'argument illégal;
  fin si
  produit ← 1;
  puissance ← a;
  quotient ← b;
  faire
    si quotient mod 2 ≠ 0 alors
      produit ← produit * puissance;
      produit ← produit mod m;
    fin si
    puissance ← puissance * puissance;
    puissance ← puissance mod m;
    quotient ← quotient div 2;
  tant que quotient ≠ 0;
  retourner produit;
fin fonction
```

Remarque. Afin de faciliter la gestion des messages en blocs de 4 lettres, les enseignants vous fournissent les fonctions *Java encoder*(String) et *decoder*(long[]).

5. **Une signature du prof.** Le prof vous transmet sa clé publique et une signature RSA (individualisée pour votre équipe et amputée du message), qu'il a obtenue en appliquant l'encryption avec sa clé privée sur des blocs de 4 lettres d'un texte en clair. Utiliser les outils développés et fournis plus haut pour trouver le message en clair. Affichez ce message dans le programme principal.

6. **Un message secret.** Vous devez utiliser la clé publique du prof afin de lui envoyer un message secret de votre composition, que vous afficherez dans le programme principal. (Veuillez rester civilisé et courtois dans vos propos). Si, une fois le message décrypté avec sa clé privée, le prof obtient un texte d'au moins 20 caractères qui fait du sens (du moins un énoncé en français) tout en restant civilisé et courtois, vous aurez les points pour cette question.