TP Cryptographie
Méthode RSA

NOMS et Prénoms des étudiants :
Groupe :

Partie 1 - Préparation.

1 - Arithmétique.

On donne dans cette partie quelques commandes concernant l'arithmétique.

```
> restart;
```

On choisit deux entiers naturels :

```
> a:=15264;
  b:=84962456;
```

On calcule le pgcd de n et m (en Anglais on dit Integer Greatest Common Divisor, d'où le nom de la commande Maple.) :

```
> igcd(a,b);
```

On fait du calcul modulaire :

```
> b mod a;
  b^2-b+6 mod a;
  (((b mod a)^2 )-b mod a)+6 mod a;
```

Remarque : l'opérateur mod n'est pas tout à fait équivalent à la commande irem.

En effet ici tout est normal :

```
> b mod a;
  irem(b,a);
```

mais par contre ici c'est différent :

```
> c:=-b;
  c mod a;
  irem(c,a);
```

On peut aussi calculer les inverses modulaires lorsqu'ils existent :

```
> b^(-1) mod a;
> 84972455^(-1) mod a;
```

Vérification :

```
> 84972455*12311 mod a;
```

2 - Le hasard de Maple.

La commande rand(a..b)() renvoie un nombre choisi "au hasard" entre les entiers a et b. En réalité, cette commande exécute un certain algorithme qui génère une suite de nombres dits "aléatoires".

```
> restart;
> rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
```

Mais la commande "rand" renvoie toujours la même suite de nombre à partir du démarrage du logiciel (à savoir 6 9 5 1 3 5 ...) :

```
> restart;
> rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
```

Dans ce TP, chaque binôme va générer deux grands nombres premiers "au hasard" pour créer son propre code secret RSA. En utilisant directement la commande "rand", tous les binômes obtiendraient les mêmes nombres premiers, donc tous le même code !

Pour éviter ce (gros) problème, on joue sur l'initialisation de la commande "rand" en fixant la valeur d'un nombre appelé la **graine** (="seed" en Anglais), qui sert de point de départ à l'algorithme, par la commande suivante :

```
> restart;
> Seed:=randomize();
```

La valeur obtenue pour la graine est liée au temps d'horloge de chaque ordinateur. Elle est donc différente pour chaque binôme, sauf coincidence extraordinaire.

```
> rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
```

A chaque redémarrage du logiciel, on obtient cette fois une suite de nombres différente :

```
> restart;
> Seed:=randomize();
> rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
  rand(0..9)();
```

A partir de maintenant, la suite de nombres aléatoires de chaque occurrence de Maple est donc différente de celle de toutes les autres, et vous êtes prêts à créer votre propre code RSA, différent de celui de vos voisins.

Partie 2 : La création du code RSA.

Question 1 - Vous devez fixer le nombre de chiffres des nombres premiers que vous allez utiliser et ranger ce nombre dans la variable "nbch". Vous devez prendre un nombre de chiffres d'au moins 40 (pour rendre le code difficile à déchiffrer) et d'au plus 100 (pour que Maple puisse faire facilement les calculs). Ce n'est pas très loin des valeurs utilisées dans la pratique, puisqu'on estime par exemple pour des transactions commerciales sur Internet, que des nombres premiers de 1024 bits, c'est-à-dire environ 300 chiffres décimaux, assureront un très bon niveau de sécurité jusqu'en 2070 au moins.

On choisit un nombre de nbch chiffres au hasard et on le range dans la variable "R", puis on vérifie que le nombre de chiffres de R est bien nbch :

```
> R:=rand(10^(nbch-1)..10^(nbch)-1)();
  length(R);
```

En général, le nombre R n'est pas premier, comme on peut le voir en effectuant le test ci-dessous :

```
> isprime(R);
```

Dans ce cas, on calcule le nombre premier immédiatement supérieur à R en utilisant la commande ci-dessous, et on range le résultat dans la variable "p":

```
> p:=nextprime(R);
```

Et on peut vérifier que cette fois p est bien un nombre premier de nbch chiffres :

```
> length(p);
  isprime(p);
```

Pour être sûr que la valeur de p ne changera plus au cours du TP, on le copie-colle ci-dessous :

```
> p := ..........;
```

Question 2 - Recommencer les mêmes opérations pour fabriquer un deuxième nombre premier de nbch chiffres, et ranger le résultat dans la variable q.

Encore une fois, la valeur de q ne doit plus changer au cours de ce TP, et pour s'en assurer, vous devez la copier-coller ci-dessous :

```
> q := ..........;
```

Question 3 - Calculer les produits N=p*q et phi=(p-1)*(q-1)

Question 4 - Choisir un nombre au hasard entre 0 et phi-1 et vérifier s'il constitue un compliqueur acceptable. Si ce n'est pas le cas, recommencer l'opération jusqu'à obtenir un compliqueur acceptable. Ranger le résultat dans la variable e.

Le nombre e ne doit plus changer jusqu'à la fin de ce TP, et pour s'en assurer, vous devez le copier-coller ci-dessous :

```
> e :=.............;
```

Question 5 - Calculer le faciliteur correspondant au compliqueur e et ranger le résultat dans la variable d.

Question 6 - De quels nombres est constituée la clé publique de votre code RSA ?

Les autres éléments de votre code RSA sont privés.
Parmi ces éléments privés, il y en a que vous feriez mieux de faire disparaître par sécurité (dans la vraie vie, pas dans ce TP), et d'autres que vous devez absolument conserver en lieu sûr, lesquels ?

Eléments privés à conserver :

Eléments privés qu'on peut jeter :

Partie 3 - Test de chiffrage et déchiffrage d'un nombre

avec votre propre code RSA.

Question 1 - Votre code RSA ne permet de chiffrer que certains nombres, lesquels ?

Réponse :

Question 2 - Choisir un nombre à chiffrer au hasard et ranger le résultat dans la variable "mess". Vérifier que la taille de ce message est correcte.

Question 3 - Chiffrer le message mess avec votre code RSA et ranger le résultat dans la variable "crypto".

Remarque : La commande a^b habituelle pour calculer les puissances vous donne un overflow. Vous devez utiliser une commande plus puissante appelée "Power(a,b)" qui permet le calcul modulaire rapide.

```
> mess^e mod N;
```

Question 4 - Déchiffrer le cryptogramme crypto et ranger le résultat dans la variable "decrypto", toujours en utilisant la commande "Power".

Question 5 - Vérifier que vos opérations de chifffrage et de déchiffrage se sont effectuées correctement.

Question 6 - Il ne vous reste plus qu'à publier la partie publique de votre code secret. Pour cela, vous devez aller sur le forum de cryptographie de votre groupe de TP, sur le serveur web de l'IUT en ligne, auquel vous avez accès pendant cette séance. Vous devez créer sur ce forum votre boîte aux lettres, c'est-à-dire un nouveau sujet intitulé "Boîte aux lettres RSA de (*vos deux noms*)".
Le premier message de ce sujet sera en fait la clé publique de votre code secret :

Attention toutefois à ne poster aucun élément de la partie privée de votre code !

Partie 4 - Envoyer un message secret.

Vous devez envoyer un message secret à vos voisins de droite.
Écrire ici les noms des destinataires :

Question 1 - Tapez votre texte ci-dessous. Votre message doit comporter au moins 3 lignes de texte.

```
> texte:= ..............;
```

Question 2 : Coller ci-dessous la clé publique de vos destinataires qui se trouve sur le forum (dans leur boîte aux lettres), et la ranger dans les variables "Nbis" et "ebis".

```
> Nbis:= ...........;
```

```
> ebis:=.............;
```

Question 3 - Calculer le nombre de chiffres de Nbis.

Question 4 - Pour convertir votre texte en un message numérique, vous devez décider du nombre de chiffres des nombres que vous allez utiliser. Choisissez ce nombre de façon correcte, et rangez-le dans la variable "longbloc". Vous avez intérêt à choisir un nombre plutôt grand, pour limiter la longueur de la liste de nombres à chiffrer.

```
> longbloc:=...;
```

Pour convertir votre message texte en un message numérique, c'est-à-dire en une liste de nombres, on vous donne la fonction "encodenum" ci-dessous (voir TP1). La fonction "encodenum(k,t)" transforme une chaîne de caractères t en une liste de nombres de k chiffres, selon la méthode du TP1.
Ne pas oublier d'appuyer d'abord sur "Entrée" pour enregistrer le code de la commande dans "encodenum".

```
> encodenum:=proc(k,t)
  local nbcar,listeascii,listenum,nb,nbch,nbbl,i;
  nbcar:=length(t);
  listeascii:=convert(t,bytes):
```

```
listenum:=[seq(listeascii[i]*10^(3*(i-1)),i=1..nbcar)]:
nb:=add(listenum[i],i=1..nbcar):
nbch:=length(nb):
nbbl:=ceil(nbch/k):
return([seq(irem(iquo(nb,10^(k*(i-1))),10^(k)),i=1..nbbl)]);
end proc:
```

Question 5 : Convertir votre message texte en un message numérique en utilisant la fonction "encodenum". Ranger le résultat dans la variable "messnum".

La commande ci-dessous compte le nombre de blocs de votre message numérique.

```
> nbrebloc:=nops(messnum);
```

Question 6 : Chiffrez votre message ci-dessous et ranger le résultat dans la variable "crypto".

```
> crypto:=.........;
```

Question 7 : Poster votre cryptogramme sur le forum dans la boîte aux lettres des destinataires (c'est-à-dire "Répondre" à leur premier message). Si votre cryptogramme est trop long pour faire un Copier/Coller, ce qui est probablement le cas, vous pouvez passer par le Bloc-Note de Windows.

Partie 5 - Recevoir un message secret.

Vous devez recevoir un message de vos voisins de gauche.
Écrire ici les noms des expéditeurs :

Question 1 : Récupérer sur le forum le cryptogramme qui vous est destiné, et le ranger dans la variable "cryptobis".

```
> cryptobis := .........;
```

Question 2 : Calculer le nombre de blocs du cryptogramme qui vous est destiné, et ranger le résultat dans la variable "nbreblocbis".

```
> nbreblocbis:= ..........;
```

Question 3 : Déchiffrer le message qui vous a été envoyé et ranger le résultat dans la

variables "decrypto".

```
> decrypto:= ..........;
```

Question 4 : Il ne vous reste plus qu'à convertir en texte le message numérique que vous avez déchiffré. Pour cela, on vous donne la fonction "decodenum" ci-dessous :

La fonction decodenum prend en entrée un message numérique m et renvoie le texte qu'il représente (voir TP1).

```
> decodenum:=proc(m)
  local nbblbis,lgblbis,nbbis,nbchbis,nbcarbis,listeasciibis,i;
  nbblbis:=nops(m);
  lgblbis:=max(seq(length(m[i]),i=1...nbblbis));
  nbbis:=add(m[i]*10^(lgblbis*(i-1)),i=1..nbblbis):
  nbchbis:=length(nbbis):
  nbcarbis:=ceil(nbchbis/3):
  listeasciibis:=[seq(irem(iquo(nbbis,10^(3*(i-1))),10^(3)),i=1..
  nbcarbis)]:
  return(convert(listeasciibis,bytes));
  end proc:
```

Partie facultative - Décrypter un code RSA.

On se place dans la position d'un espion qui veut décrypter un cryptogramme qu'il a intercepté. S'il ne connaît que la partie publique du code RSA, il lui faut arriver à calculer le faciliteur d à partir du module n et du compliqueur e. C'est-à-dire qu'il faut factoriser n pour retrouver p et q, puis calculer phi=(p-1)*(q-1) et enfin inverser e modulo phi. En pratique, seule la première étape est difficile, car très longue, même avec les meilleurs algorithmes connus et en utilisant les ordinateurs les plus rapides du moment. Pour vous donner une idée de cette difficulté, on vous donne ci-dessous quelques exemples :

La commande Maple permettant de décomposer des entiers en facteurs premiers est "ifactor(...)". Par exemple :

```
> ifactor(4012);
```

Vérification :

Malheureusement, ce calcul est long pour de grands nombres (le temps est donné en secondes) :

```
> debut:=time():
  ifactor
```

```
   (168745340390678148617464115971457696673787197948617464115);
   time()-debut;
```

```
> debut:=time():
   ifactor
   (168745340390678191973498737871979486174641159717145769667416847
   451687456785);
   time()-debut;
```

```
>
```

Et surtout, le temps de factorisation augmente énormément quand on utilise des grands nombres premiers, comme on a fait dans ce TP :

Cette suite d'instructions calcule le temps de factorisation .

```
for i from 20 to 47
do
test1:=nextprime(rand(10^(i-1)..10^i-1)()):
test2:=nextprime(rand(10^(i-1)..10^i-1)()):
test:=test1*test2;
debut:=time():
ifactor(test):
print(time()-debut);
end do:
```

Nous avons obtenu les résultats suivants. Vous pouvez constater qu'à partir de 35 chiffres on atteint déjà la minute. Le dernier temps de calcul correspond à la factorisation d'un nombre composé du produit de deux nombres premiers de 47 chiffres. C'est encore très loin des valeurs qui ont été utilisées dans ce TP, et plus encore des valeurs utilisées en réalité. Et pourtant, ce calcul a déjà duré presque 3 heures.

```
0.364
0.460
0.700
0.848
1.112
1.396
2.552
3.244
4.628
7.477
8.748
18.626
23.945
38.199
41.926
```

```
                                61.480
                               109.819
                               182.195
                               305.211
                               410.114
                               713.837
                              1084.471
                              1408.408
                              1817.266
                              2939.976
                              3690.050
                              6531.416
                             10108.923
```

Vous pouvez essayer vous-même ce calcul.

****
****
****
****

ATTENTION : Sauvegarder votre fichier avant de lancer ce calcul, pour éviter les accidents !!!!

****
****
****
****

```
> for i from 20 to 40
  do
  test1:=nextprime(rand(10^(i-1)..10^i-1)()):
  test2:=nextprime(rand(10^(i-1)..10^i-1)()):
  test:=test1*test2;
  debut:=time():
  ifactor(test):
  print(time()-debut);
  end do:
```