IUT Littoral **2019-2020**
Info 2A **C++**

Projet - RSA

Le but de ce devoir est d'implémenter la méthode de chiffrement RSA ex nihilo (ou presque).

MODALILTÉ

Le devoir est à faire seul ou en binôme. Il est à rendre au plus tard le vendredi 20 mars. Vous m'enverrez par mail, ou déposerez dans la boîte de dépot, une archive `.zip` ou `.tar.gz` contenant :

- le code source correctement indenté ;
- un fichier `info.txt` contenant les noms, prénoms et adresses emails des membres du binôme ;
- un fichier `rapport.pdf` contenant les réponses aux exercices (qui ne sont pas de la création de code) ;
- les fichiers `middle_square.txt` et `standard_minimal.txt` dont le contenu est précisé dans le devoir.

Le devoir est composé de plusieurs parties plus ou moins indépendantes, certaines d'entre elles peuvent être traitées en parallèles.

1. ARITHMÉRIQUE

Tout au long de ce devoir nous allons manipuler des entiers sur 64 bits. Il est donc commode de définir un type `Entier` à l'aide de la commande

```
typedef long long int Entier;
```

Assurez-vous que pour votre compilateur et votre machine le type `Entier` utilise au moins 64 bits c'est à dire 8 octets. La commande `sizeof(Entier)` doit retourner un nombre supérieur ou égal à 8. Si ce n'est pas le cas il faut m'en prévenir.

Exercice 1. Écrire une fonction

```
Entier pgcd(Entier a,Entier b)
```

qui retourne le pgcd des entiers `a` et `b`.

Exercice 2. Écrire une fonction

```
Entier pgcd_etendu(Entier a,Entier b,Entier &u,Entier &v)
```

qui retourne le pgcd d des entiers `a` et `b` et qui donne des valeurs à `u` et `v` vérifiant $\mathtt{a} \times \mathtt{u} + \mathtt{b} \times \mathtt{v} = d$.

Exercice 3. Écrire une fonction

```
Entier exp_rapide(Entier a,Entier p,Entier n)
```

qui, utilisant la méthode d'exponentiation rapide vue en cours, retourne l'unique entier b de $[0, \mathtt{n} - 1]$ congru à $\mathtt{a}^{\mathtt{p}}$ modulo `n`

Exercice 4. Écrire une fonction

```
Entier inverse_modulaire(Entier a,Entier n)
```

qui étant donnés deux entiers `a` et `n` premiers entre eux retourne l'unique entier b vérifiant $\mathtt{a}b \equiv 1 \mod \mathtt{n}$.

Exercice 5. Écrire une fonction

```
bool est_premier(Entier n)
```

qui teste si un nombre `n` compris entre 0 et $2^{32} - 1$ est premier ou pas. Pour cela on va tester si le nombre est possiblement premier à l'aide du test de primalité de Fermat avec les témoins $2, 3, 5, 7, 11, 13$ et 17. Si le nombre est détecté possiblement premier, on décide s'il est ou non premier en le recherchant dans la liste des exceptions se trouvant sur mon site dans le fichier `Exceptions.txt`

2. GÉNÉRATEURS PSEUDO-ALÉATOIRES

Pour générer de manière aléatoire de grands nombres premiers nous allons d'abord générer de grands nombres (sur 32 bits).

Exercice 6. Écrire une fonction

```
Entier prochain_middle_square(Entier x)
```

qui étant donner un entier `x` inférieur ou égal à 9999 retourne le terme généré par `x` après application de l'algorithme `middle_square`.

Exercice 7. Dans un fichier nommé `middle_square.txt` mettre les 60 premiers entiers générés par `middle_square` à partir de la valeur initiale 1111.

Exercice 8. Écrire une fonction

```
Entier periode_middle_square(Entier x)
```

qui retourne la période de `middle_square` pour la valeur initiale `x`, c'est-à-dire, le plus grand entier n tel que la suite

$$x = x_0, x_1, ..., x_n$$

générée par `middle_square` soit sans doublon.

Exercice 9. Quelle est la périodicité de `middle_square` pour la valeur initiale $\texttt{x} = 1111$? et pour $\texttt{x} = 1234$?

Exercice 10. En utilisant la fonction `prochain_middle_square` écrire une fonction

```
Entier middle_square(Entier x,Entier n)
```

qui retourne le `n` ème terme généré par l'algorithme `middle_square` à partir de la valeur initiale `x`.

Exercice 11. Écrire une fonction

```
Entier prochain_minimal_standard(Entier x)
```

qui étant donné un nombre $\texttt{x} < 2^{31}-1$ retourne le terme généré à partir de `x` par l'algorithme `minimal_standard`.

Exercice 12. Dans un fichier nommé `minimal_standard.txt` mettre les 60 premiers entiers générés par `middle_standard` à partir de la valeur initiale 1.

Exercice 13. En utilisant la fonction `prochain_minimal_standard` écrire une fonction

```
Entier minimal_standard(Entier x,Entier n)
```

qui retourne le `n` ème terme généré par l'algorithme `minimal_standar` à partir de la valeur initiale `x`.

Nous allons maintenant mélanger les deux méthodes pour générer de manière aléatoire de grands nombres.
Définir une variable globale `Entier alea` qui sera utilisée par le générateur pseudo-aléatoire que nous allons construire.

Exercice 14. Écrire une fonction

```
void init_alea()
```

qui initialise la variable globale `alea` de la manière suivante :

- `t` est le nombre de secondes depuis le 1 janvier 1970 à minuit, `t=time(NULL)` ;
- `s` est l'entier $\texttt{t}\%50 + 3$;
- `n` est le `s` ème entier généré par `middle_square` à partir de la valeur initiale 1111 ;
- `alea` est le `n` ème entier généré par `standard_minimal` à partir de la valeur initiale `time(NULL)`

Exercice 15. Écrire une fonction

```
Entier aleatoire()
```

qui retourne la valeur de `alea` et redéfinit `alea` à l'aide de `prochain_minimal_standard` appliquée à l'ancienne valeur de aléa.

Exercice 16. A partir des fonction `aleatoire` et `est_premier` écrire une fonction

```
Entier premier_aleatoire()
```

qui retourne un nombre premier généré de manière pseudo-aléatoire.

3. Génération de clés

Le cryptosystème RSA est une méthode de chiffrement asymétrique qui nécessite la génération d'une clé publique et d'une clé privée. Voici comment sont créées ces clés :

(1) on génère aléatoirement deux nombres premiers distincts p et q ;

(2) on calcule $n = p \times q$ et $\phi(n) = (p-1) \times (q-1)$;

(3) on sélectionne un nombre premier e qui ne divise pas $\phi(n)$ (souvent $e = 3, 257$ ou 65537) ;

(4) à l'aide de l'algorithme d'Euclide étendu on calcule d tel que $ed \equiv 1$ modulo $\phi(n)$.

La clé publique est alors le couple (e, n) tandis que la clé privée est le couple (d, n).

Exercice 17. Calculer (à la main) la clé publique et la clé privée correspondant aux choix $p = 5$, $q = 11$ et $e = 3$?

Exercice 18. Proposer une structure `ClePublique` pour la clé publique et une structure `ClePrivee` pour la clé privée.

On crée une structure `Cles` contenant une clé publique et une clé privée :

```
typedef struct{
    ClePublique cle_publique;
    ClePrivee cle_privee;
} Cles;
```

Exercice 19. Ecrire une fonction

`Cles genere_cles(Entier p,Entier q,Entier e)`

qui génère une paire clé publique / clé privée associée à `p`, `q` et `e`.

Exercice 20. Tester la fonction précédente sur les valeurs de p, q et e données à l'exercice **??**. Mettre une trace d'exécution dans le rapport.

Exercice 21. Ecrire une fonction

`Cles genere_cles_aleatoire()`

qui génère une paire clé publique, clé privée de manière aléatoire avec $n \geqslant 256$.

4. Chiffrement, déchiffrement et test

Dans cette section, (e, n) désigne une clé publique et (d, n) une clé privée. Pour chiffrer un message M de $\mathbb{Z}/n\mathbb{Z}$ on calcule $C = M^e \mod n$. Pour déchiffrer un message C de $\mathbb{Z}/n\mathbb{Z}$ on calcule $M = C^d \mod n$.

Exercice 22. Reprenons la clé publique et la clé privée de l'exercice **??**. Chiffrer à la main, le message $M = 10$. Déchiffrer, toujours à la main, le message $C = 13$.

Exercice 23. Ecrire une fonction

`Entier chiffre(ClePublique cle, Entier M)`

retournant le chiffré du message `M` à l'aide de la clé publique `clé`.

Exercice 24. Ecrire une fonction

`Entier dechiffre(ClePrivée cle, Entier C)`

retournant le déchiffré du message `C` à l'aide de la clé privée `clé`.

Exercice 25. Tester les fonctions `chiffre` et `déchiffre` avec les données de l'exercice **??**. Mettre une trace d'exécution dans le rapport.

5. Application

Le but de cette section est de constuire une application permettant de

(1) créer une clé ;

(2) chiffrer un fichier en utilisant une clé ;

(3) déchiffrer un fichier en utilisant une clé.

L'application proposera alors un menu principal qui appelera des différents sous menus correspondant à chacune des actions précédentes. Après l'éxécution d'une action nous souhaitons retourner au menu principal.

Exercice 26. Mettre en place le système de menu. On ne demande pas de coder les actions pour le moment mais seulement d'appeler une procédure se contentant d'afficher un message. Par contre nous demandons que toutes les interactions nécessaire avec l'utilisateur soient présentes : demande de nom de fichier, ... Vous devrez aussi gérer les erreurs de saisie.

Les différents fichiers seront ouvert en mode binaire à l'aide de la libraire `fstream` et plus particulièrement des focntions dont une documentation est disponible ici :

`http://www.cplusplus.com/reference/fstream/fstream/`

On utilisera en particulier les fonctions `open`, `read`, `write`, `seekg`, `tellg`.

Exercice 27. Finir de coder la fonction permettant de générer un couple de clés. La clé privée sera enregistrée dans un fichier `filename.pri` tandis que la clé publique sera enregistré dans un fichier `filename.pub`. Les différents entiers seront enregistré sur 8 octets. Chaque fichier devra donc contenir 16 octets.

Afin de chiffer/déchiffrer un fichier nous devons découper celui-ci en blocs de k octets de tel sorte que chaque bloc puisse être coder par un entier compris enter 0 et $n-1$.

Exercice 28. Ecrire une fonction

`int taille_bloc(Entier n)`

qui étant un entier n retourne le plus grand entier i tel qu'on ait $256^i \leqslant n$.

Exercice 29. Ecrire un fonction

`Entier code(char* buffer,int k)`

qui étant un tableau `buffer` de taille `k` retourne l'entier

$$\sum_{i=0}^{k-1} \texttt{buffer[i]} \times 256^i.$$

Exercice 30. Décomposer le nombre 123456789 en base 256.

Exercice 31. A l'aide des divisions euclidiennes successives par 256 écrire un fonction

`void decode(Entier m,int k,char* c)`

qui étant un tableau `c` de taille k détermine `c[i]` pour $i = 0, \ldots, k-1$ tels qu'on ait

$$\texttt{m} = \sum_{i=0}^{k-1} c_i \times 256^i.$$

Pour chiffre un fichier avec la clé publique `cle` $= (n, e)$ on utilisera la méthode suivante :

(1) on déterminera la taille t du fichier à chiffrer
(2) à l'aide de la fonction `taille_bloc` on déterminera la taille k des blocs qu'on peut représenter par un entier $\leqslant n$;
(3) dans le fichier chiffré on écrira les nombres t et k à l'aide de 8 octets chacun ;
(4) on calculera le nombre $\ell = \lceil t/k \rceil$ de blocs de taille k composant le fichier à chiffrer ;
(5) on lira le fichier à chiffrer bloc par bloc ;
   (a) pour chaque bloc b ainsi lu on calculera `M = code(b,k)` ;
   (b) on calculera `C = chiffre(cle,M)` ;
   (c) on calculera le tableau `c` $= [$`c[0],...,c[k-1]`$]$ à l'aide de `decode(C,k,c)` ;
   (d) on écrira alors le bloc `c` de `k` octets ainsi obtenu dans le fichier chiffré.

Il se peut que le dernier bloc lu d fichier à chiffré ne contienne moins de k octets. On supposera alors que les octets manquant valent 0, il faudra alors faire attention lors du déchiffrage. C'est pour cela qu'on a enregistré la taille du ficher original dans le fichier chiffré.

Exercice 32. Finir de coder la focntion pour l'action de chiffrer.

Exercice 33. En vous inspirant de la méthode utilisée pour le chiffrement coder la fonction pour l'action de déchiffrer.