Projet Maple : la cryptographie

Projet à rendre au plus tard par email (`alexis.flesch@utbm.fr`) le **dimanche 23 juin** (2013).

Les projets sont à programmer en maple version 7. Le projet se travaille en binôme, qui ne rendra qu'un projet pour deux étudiants. Le fichier principal se nomme "projet". Chaque programme doit impérativement proposer une fonction `main()` offrant une présentation, une aide minimale et éventuellement un menu d'utilisation. Un utilisateur ignorant tout du programme, ne disposant pas du dossier, doit pouvoir en faire une utilisation minimale en tapant `main()`.

Le travail consiste en l'écriture d'un programme et en la confection d'un dossier d'accompagnement sur papier d'environ 4-5 pages contenant entre autres le listing du programme. Chaque fonction doit y être proprement et correctement documentée (rôle de la fonction, paramètres d'entrées, valeur retournée).

Votre rapport de projet doit être **au format pdf** (tout document `docx` ou autre format propriétaire ne sera pas lu). Votre code doit être proprement **indenté** sans quoi il ne sera pas relu.

1 Introduction

Alice souhaite communiquer des informations secrètes à son ami Bob avec qui elle communique par email. Après quelques recherches sur internet, elle apprend que les chaînes de caractère sont traitées par l'ordinateur comme des nombres. À chaque lettre correspond un nombre unique la représentant (par exemple, le a minuscule correspond au nombre 97).

Il existe en fait énormément de manières d'encoder un texte pour des raisons historiques (chaque région du globe ayant pendant longtemps utilisé sa propre recette jusqu'à la démocratisation de la norme utf-8).

Alice décide de se cantonner à la table ASCII étendue (aussi appelée ISO-latin-1) qui est une table contenant 256 caractères dont ceux accentués (et qui est celle utilisée par Maple).

Alice décide donc d'écrire une fonction Maple prenant en argument une chaîne de caractères (une phrase), et rendant une liste des codes ASCII correspondant à chacun des caractères. Alice fait donc appel à vous ! Écrivez une fonction `to_ascii` se comportera comme suit :

```
to_ascii("maple") ;
     [109,97,112,108,101]
```

Afin que Bob puisse décrypter le message d'Alice, il faut qu'il puisse reconstituer une phrase à partir d'une suite de nombres. Écrivez une fonction `to_char` permettant de faire :

```
to_char([109,97,112,108,101]) ;
     "maple"
```

2 La cryptographie symétrique

La cryptographie symétrique est sans doute la plus ancienne forme de cryptographie. Elle repose souvent sur l'utilisation d'une *clef* secrète qui doit être connue à la fois par Alice et par Bob. Quiconque connaissant la clef est à même de déchiffrer le message.

Il existe une infinité de méthodes symétriques. Alice décide d'en essayer quelques unes.

2.1 Chiffrement par décalage

Le chiffrement par décalage consiste à remplacer chaque lettre de l'alphabet par une lettre à distance fixe, soit vers la gauche, soit vers la droite. Pour Alice, cela consiste donc à ajouter au code ASCII de chaque lettre un nombre entier fixé. Par exemple, en décalant de 3, 97 devient 100, 98 devient 101, etc... Attention, il ne faut pas "sortir" de la table ! Ainsi, 255 deviendra 2 et non 258 puisque la table ne contient que 256 caractères.

Écrire une fonction `decalage` prenant en entrée une chaîne de caractères et un entier et renvoyant la chaîne décalée :

```
decalage("abcde",3) ;
    "cdefg"
```

2.2 Chiffrement par substitution

Le chiffrement par substitution est similaire au chiffrement par décalage, mais les substitutions sont faites à l'aide d'une permutation de l'alphabet.

À l'aide de la librairie `combinat` et de la fonction `permute`, choisir une permutation de l'ensemble $[\![0,255]\!]$.

Écrire une fonction `substitution` prenant en entrée une chaîne de caractères et renvoyant le message chiffré. Appliquer cette fonction 256 fois à la chaîne `"maple"`. Que remarque-t-on ?

3 La cryptographie asymétrique

Après avoir programmé les méthodes précédentes, Alice se trouve face à un problème de taille. Elle doit communiquer à Bob la manière de décrypter ses messages. Mais si quelqu'un intercepte son message, alors il pourra lui aussi décrypter les futurs messages d'Alice.

Heureusement, il existe d'autres méthodes de cryptographie, dites asymétriques, permettant de contourner ce problème.

3.1 La cryptographie RSA

La cryptographie RSA repose sur des résultats d'arithmétique simples, mais surtout sur le fait qu'on ne sait pas à l'heure actuelle factoriser rapidement des très grands nombres en produit de nombres premiers.

Pour utiliser RSA, Bob doit créer deux *clefs*. L'une publique, qu'il donnera à Alice, l'autre privée, qu'il ne dévoilera à personne.

3.1.1 Création des clefs

i) choisir deux "grands" nombres premiers p et q (on prendra des nombres à au moins 10 chiffres)

ii) calculer le produit $n = pq$;

iii) calculer $\varphi = (p-1)(q-1)$;

iv) déterminer un entier c premier avec φ ;

v) déterminer un entier d tel que φ divise $cd-1$;

On sait d'après le théorème de Bézout que l'étape v) est possible. On se reportera aux commandes données en annexe pour effectuer les différentes étapes.

Le couple (n,c) est la *clef publique* de Bob. La *clef privée* est le couple (n,d).

3.1.2 Chiffrement du message

Supposons qu'Alice veuille transmettre le nombre 97 à Bob (correspondant au caractère `a`). Elle va alors calculer :

```
x  := 97&^c mod n ;
```

On prendra soin de ne pas omettre l'esperluette (le &) permettant d'accélérer les calculs. Alice peut maintenant transmettre son message (x) à Bob.

3.1.3 Déchiffrement du message

Bob a reçu le nombre x de la part d'Alice. Il souhaite le déchiffrer. Il lui suffit pour cela de calculer :

```
y  := x&^d mod n ;
```

3.1.4 Algorithme RSA

Fabriquer une clef privée ainsi qu'une clef publique pour Bob, puis écrire deux fonctions, l'une permettant à Bob de crypter un message, et l'autre permettant à Alice de le décrypter.

4 Pour aller plus loin

4.1 Chiffrement par substitution polyalphabétique

Le chiffrement par substitution polyalphabétique est similaire au chiffrement par substitution. Cependant, la substitution effectuée dépend alors de la position de la lettre dans le texte.

Par exemple, on peut décider :

- d'effectuer un décalage de 3 sur la première lettre ;
- d'effectuer un décalage de -1 sur la deuxième lettre ;
- d'effectuer un décalage de 2 sur la troisième lettre ;

puis, on réitère sur les lettres suivantes.

4.2 Message caché dans une image

Dans la plupart des formats d'image, chaque pixel est la donnée de trois nombres entre 0 et 255 correspondant au *code RGB* du pixel (la proportion de Rouge, de Vert et de Bleu du pixel).

Imaginer une méthode permettant de cacher du texte dans une image en ne la détériorant "pas trop" (on ne demande pas d'écrire de fonction maple). Comment cacher simplement un texte dans une image tout en le cryptant ?

4.3 Cryptographie par blocs dans RSA

Plutôt que de chiffrer les messages caractère par caractère, on peut décider de regrouper ces derniers par groupes, ce qui augmente la sécurité de RSA. Par exemple, la chaîne `"abc"` deviendrait 97098099. Il ne faut cependant pas prendre des blocs trop grands puisque les nombres à chiffrer doivent être plus petits que p et q.

4.4 Casser RSA ?

Pour déchiffrer un message crypté par RSA sans connaître la clef privée de l'expéditeur, il suffit de factoriser l'entier n. Cependant, cela peut être très coûteux en temps. On pourra tracer, à l'aide de la fonction `time()` le temps nécessaire à la factorisation d'un nombre en fonction de son nombre de décimales.

4.5 Cryptographie hybride

La cryptographie hybride, très utilisée dans la vie courante (par exemple dans les logiciels PGP et GnuPG) combine les avantages des deux méthodes symétriques et asymétrique. Elle consiste par exemple à envoyer à l'expéditeur une méthode symétrique via RSA.

5 Annexe - commandes Maple utiles

5.1 Manipulation des chaînes de caractères

La librairie `StringTools` permet de manipuler des chaînes de caractères, et en particulier de travailler directement avec le code ASCII :

- `Ord("a")` : donne le code ASCII de la lettre `a` ;
- `Char(97)` : donne le caractère correspondant au 97ème caractère de la table ASCII ;
- `cat("a","b")` : concatène les chaînes de caractère `a` et `b` ;
- `length("maple")` : rend la longueur de la chaîne de caractères (ici 5) ;
- `"maple"[3]` : rend la troisième lettre de la chaîne (ici `p`) ;

5.2 Arithmétique

- `258 mod 256` : rend le reste dans la division euclidienne de 258 par 256 (essayer avec d'autres valeurs !) ;
- `gcd(25,15)` : rend le plus grand commun diviseur à 25 et 15 (ici 5). Lorsque `gcd` rend 1, cela signifie que les nombres sont premiers entre eux ;
- `25^(-1) mod 42` : rend 37, qui vérifie que $25 \times 37 - 1$ est divisible par 42 ;

5.3 Interaction avec l'utilisateur

- `x :=readstat("entrez une commande")` : demande à l'utilisateur d'entrer une commande maple (par exemple `sin(Pi/4) ;`) et la stock dans la variable `x` ;