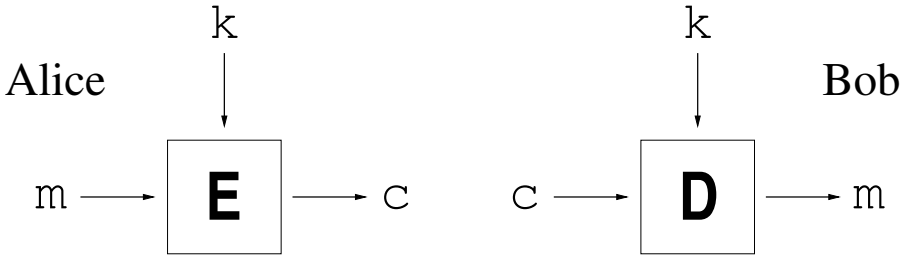
TD/TP Sécurité et aide à la décision

Un des principaux buts de la *cryptographie* est d'assurer la confidentialité des communications : Alice souhaite envoyer un message à Bob, mais elle veut que seul *Bob* puisse en avoir connaissance. Si elle l'envoie tel quel, le message pourra être lu par n'importe qui "écoutant" la communication. Typiquement, les données échangées sur Internet peuvent être, en général, écoutées. Aussi, si le canal de communication n'est pas sûr, c'est le message qu'il faut transformer pour le rendre illisible à toute autre personne que Bob : c'est le rôle du *chiffrement*.

Chiffrer un message consiste à lui appliquer une transformation que seul le destinataire du message peut inverser.

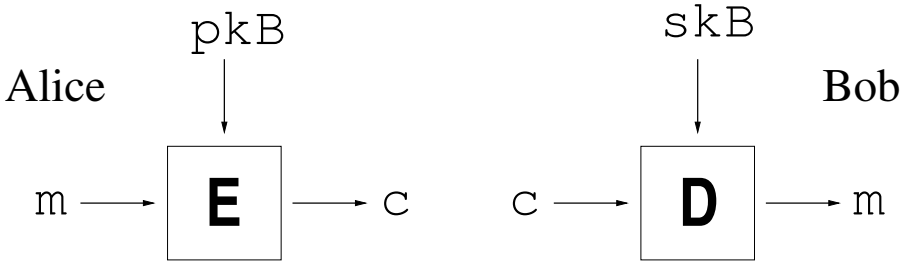
Jusqu'en 1977, pour tenter de sécuriser leurs communications, Alice et Bob commençaient par s'échanger un *secret* k, puis s'en servait comme *clé* d'un algorithme de chiffrement et de déchiffrement : c'est la *cryptographie à clé secrète*, ou *symétrique*. Dans la figure suivante, m désigne le *texte clair* (le message), et c le *chiffré*.



Les algorithmes de chiffrement E et de déchiffrement D sont connus de tous le monde : la sécurité repose sur le fait que la clé k est secrète.

Un exemple ancestral d'algorithme de chiffrement de ce type est le *chiffrement de César*. L'opération de chiffrement consiste à remplacer une lettre par la lettre qui se trouve 3 rangs plus loin dans l'alphabet : ainsi a est remplacé par d, b par e, c par f, etc...

En 1976, W. Diffie et M. Hellman inventent le concept de *cryptographie à clé publique* ou *asymétrique*. Ils règlent en particulier le problème de l'échange du secret, puisque dorénavant, chaque utilisateur possède 2 clés, dont l'une est publique, et l'autre est gardée secrète. La connaissance de la clé publique ne permet pas de retrouver la clé secrète. Pour chiffrer, R. Rivest, A. Shamir et L. Adleman ont proposé en 1977 un système de chiffrement, appelé RSA, qui est la première incarnation du chiffrement à clé publique. Pour chiffrer un message à destination de Bob, Alice obtient, dans un annuaire par exemple, la clé publique de Bob, et s'en sert dans l'algorithme de chiffrement. Bob, recevant le message chiffré, utilise dans l'algorithme de déchiffrement sa propre clé secrète. Dans la figure suivante, skB désigne la clé secrète de Bob, et pkB sa clé publique.



Exercice 1. Cryptographie asymétrique : le chiffrement basé sur le problème du sac à dos.

C'est un algorithme de chiffrement à clé publique qui est dû à R. Merkle et M. Hellman. Il repose sur la complexité algorithmique du problème du *sac à dos* : on a n objets de poids $a_1, \ldots, a_n$ respectivement. Le sac à dos a une capacité totale de S. Quels objets doit-on mettre dans le sac à dos pour obtenir un poids total égal à S ? Plus formellement, le problème général consiste, étant donné un n-uplet $(a_1, a_2, \ldots, a_n)$ d'entiers positifs, et un nombre S positif également, à trouver un n-uplet $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ tel que $S = \sum a_i x_i$.

Si les $(a_i)_i$ n'ont pas de propriétés particulières, la recherche des $(x_i)_i$ (le message codé en binaire) demande un travail exponentiel en n. Ainsi la résolution du problème est *impossible en temps raisonnable* et donc on peut présumer que l'algorithme du sac à dos est bien adapté au cryptage.

Pour pouvoir utiliser cette propriété pour assurer la confidentialité, il faut trouver une *trappe* dont la connaissance permet de résoudre le problème efficacemnt. Ainsi le destinataire sera la seule personne à connaître cette trappe et pouura déchiffrer efficacement. Heureusement, il existe une classe de n-uplet $(a_1, \dots, a_n)$ pour lesquels on puisse résoudre facilement le problème (en ayant eventuellement une connaissance supplémentaire tenue secrète), sinon il serait impossible de déchiffrer (y compris pour le destinataire).

Les suites super-croissantes Un n-uplet $(b_1, \dots, b_n)$ est dit *super-croissant* si et seulement si

$$\forall i \in [\![2, n]\!] \quad b_i > \sum_{j=1}^{i-1} b_j.$$

— Écrire une procédure `est_super(b)` qui renvoit *vrai* si la suite b est super-croissante et *faux* sinon.

— Dans le cas ou le n-uplet b est super-croissant, le problème du sac à dos est particulièrement simple à résoudre. Donnons nous un n-uplet $b = (b_1, \dots, b_n)$ super-croissant et un entier $S = \sum_{i=1}^{n} x_i b_i$ avec $x_i \in \{0, 1\}$. Montrez que si $b_n > S$ alors $x_n = 0$ et si $b_n \leq S$ alors $x_n = 1$. En déduire un algorithme pour déterminer les x_i connaissant b et S.
Écrire une procédure `resout(S,b)` qui étant donnés une suite super-croissante $(b_i)_{1 \leq i \leq n}$ et un entier $S \leq \sum_{i=1}^{n} b_i$ renvoit un entier M dont l'ecriture sur n bits $M = x_1 x_2 \dots x_n$ avec $x_i \in \{0, 1\}$ vérifie [1] $S = \sum_{i=1}^{n} x_i b_i$.

Evidemment, sous cette forme particulière (avec des suites super-croissantes), l'algorithme du sac à dos ne présente aucun intérêt pour la cryptographie (tout le monde, connaissant la suite super-croissante $(b_i)_i$ est en mesure de décoder un message). L'idée de Merkle et Hellman consiste à "tordre" les b_i de facon à obtenir un n-uplet qui n'est plus super-croissant.

Pour ce faire, on choisit un nombre $u > \sum_{i=1}^{n} b_i$, et un entier $v \in \mathbb{N}$ tel que $\text{pgcd}(u, v) = 1$. Alors v est inversible dans $\mathbb{Z}/u\mathbb{Z}$. On note $w = v^{-1} \in \mathbb{Z}/u\mathbb{Z}$, on a ainsi $v \cdot w \equiv 1 \mod u$. On calcule alors le n-uplet $a = (a_1, \dots, a_n)$ tel que pour tout $i \in [\![1, n]\!]$,

$$a_i = v \cdot b_i \in \mathbb{Z}/u\mathbb{Z}.$$

Le n-uplet a est alors rendu public, tandis que v est gardé secret. Pour transmettre le message m à Bob, Alice va à partir de l'écriture de m sur n bits $m = x_n x_{n-1} \dots x_1$, calculer la somme

$$c = \sum_{i=1}^{n} x_i a_i$$

qu'elle va envoyer à Bob. Si le message est intercepté par Ève, celle-ci ne pourra pas (à priori) retrouver la suite des x_i à partir des c et du n-uplet a car c'est un problème de type sac à dos.

Pour décoder le message Bob n'aura alors qu'à calculer $c \cdot w$ modulo u, il obtiendra ainsi la somme $\sum_{i=1}^{n} x_i b_i$ et le n-uplet b étant super-croissant, il n'aura aucun mal à retrouver le nombre S.

— Écrire une procédure **Csac(txt,*a*)** qui, étant donné un texte clair **txt**, renvoit le message chiffré à l'aide du n-uplet a par la méthode décrite ci-dessus.

— Écrire une procédure **Dsac(*c*,*b*,*w*,*u*)** qui, à partir du message chiffré c, de la clé privée b, w, u renvoie le message déchiffré.

— Vérifiez que $b = [5, 13, 21, 89, 233, 377, 987, 2584, 6765, 17711, 251516, 6548456, 65484652, 114845465]$, $u = 463814521$, $v = 234203372$ et $w = 101$ vérifient bien les propriétés voulues. Déchiffrer le message suivant [648640591, 1329005307, 1162545881, 1016018455, 1116558815, 1033898626]

Cette méthode de chiffrement fut présentée par Merkle et Hellman en 1978 et semblait promise à un brillant avenir. Malheureusement, en 1982, Shamir publiait un algorithme pour casser ce schéma de chiffrement, car si le problème du sac à dos est difficile à résoudre dans le cas général, le n-uplet public a n'est pas tout à fait quelconque et une attaque est possible. Ainsi l'algorithme du sac à dos a été abandonné au profit de la méthode RSA dont l'un des inventeurs est justement Shamir...

1. Attention le problème n'admet pas forcemment de solution.

Exercice 2. Cryptographie à clé publique : le chiffrement RSA.

Cette méthode a donc été inventée en 1977 par Rivest, Shamir et Adleman. Elle est aujourd'hui extrêmement populaire et il en existe beaucoup de versions plus ou moins raffinées, mais toutes reposent sur le même principe.

Tout d'abord on se donne deux grands nombres premiers distincts p et q (typiquement de 512 bits). Considérons alors l'entier n produit de ces deux nombres $n = pq$. Il est très difficile de retrouver p et q à partir de n seul : c'est le problème de la *factorisation*.

On choisit $e \in \mathbb{N}$ premier à $\varphi(n) = (p-1)(q-1)$ et l'on calcule alors $d := e^{-1}$ l'inverse de e dans $\mathbb{Z}/\varphi(n)\mathbb{Z}$. On a la propriété suivante :

$$\forall x \in \mathbb{Z}/n\mathbb{Z};\ (x^e)^d \equiv x \mod n.$$

Les entiers n et e sont rendus publics tandis que d est gardé secret. Le chiffrement d'un message $m < n$ consiste alors à calculer $c := m^e$ dans $\mathbb{Z}/n\mathbb{Z}$. Le destinataire n'aura alors qu'à calculer c^d dans $\mathbb{Z}/n\mathbb{Z}$ à l'aide de la clé privée d pour retrouver M.

- Écrire une procédure `CRSA(`m`,`e`, `n`)` qui chiffre un message m par la méthode RSA avec les clés publiques e et n.
- Écrire une procédure `DRSA(`c`,`d`, `n`)` qui déchiffre un message c à l'aide de la clef privée d.

- Tester la consistance de votre système, sur des exemples de textes clairs (Lorsque l'on chiffre un message, avec une clé publique et qu'on déchiffre le texte chiffré, avec la clé privée associée, on retrouve le texte clair original. Pour les applications numériques vous pourrez prendre par exemple $p = 47$, $q = 59$ alors $d = e^{-1} = 17$ dans un premier temps.

Remarque : Dans la vraie vie, il faut choisir p et q des nombres premiers sur au moins 500 bits. Comment faire le système de chiffrement et déchiffrement avec les paramètres suivants ?

$$p = 378668090616600572642192533 97 \text{ et } q = 2^{60} - 173$$

et

$$e = 101 \text{ et } d = 133998139883060209235322604129728377068664013 22.$$

Exercice 3. Cryptographie symétrique : les registres à décalage.

Dans le contexte de cryptographie symétrique, deux utilisateurs souhaitant communiquer de façon confidentielle partagent un secret (préalablement échangé), que l'on va noter k.

Une technique de chiffrement connue sous le nom de *chiffrement de Vernam* permet d'obtenir une confidentialité *parfaite*. Cet algorithme théorique est malheureusement trop inefficace pour être utilisé en pratique. Il fonctionne de la façon suivante :

- Soit $m = m_1 m_2 \dots m_n \in \{0,1\}^n$ le message écrit sous forme binaire.
- La clé $k = k_1 k_2 \dots k_n$ est un suite de bits *aléatoires de la taille du message*
- Le chiffré $c = c_1 c_2 \dots c_n$ est obtenu de la façon suivante :

$$c_i = m_i \oplus k_i \quad \forall i = 1 \dots n.$$

Le problème majeur de cet algorithme, également appelé *one-time pad*, est que la clé doit être de la même longueur que le message, et qu'elle doit être changée et tirée au hasard à chaque nouvel envoi.

Cet algorithme a néanmoins inspiré les algorithmes de *chiffrement à flot* qui fonctionne de la façon suivante :

- la clé secrète k (de l'ordre de 128 bits) est utilisée comme *graine* d'un générateur pseudo-aléatoire.
- La suite chiffrante produite par le générateur pseudo-aléatoire est alors additionnée bit à bit avec le message comme précédemment.

Une façon de produire un générateur pseudo-aléatoire est d'utiliser la combinaison de *registres à décalage à rétroaction linéaire* (LFSR pour *Linear Feedback Shift Register*).

Les LFSR sont des registres formés de cellules contenant des 0 ou des 1. À chaque top d'horloge, le contenu de chaque cellule se déplace d'un cran vers la droite : le bit le plus à droite "sort" du registre et constitue un bit de la suite chiffrante, et la cellule la plus à gauche, se retrouvant vide, est rempli par un bit qui se calcule comme une combinaison linéaire des bits présents dans les différentes cellules avant le déplacement.

Les coefficients a_i sont binaires et forment les coefficients du *polynôme caractéristique* du LFSR :

$$P(X) = X^N + a_1 X^{N-1} + \dots + a_{N-1} X + a_N.$$

La clé k, de longueur N, constitue les bits d'initialisation du registre.

1. Calculer la suite produite par le LFSR de polynôme caractéristique $P(X) = X^{10} + X^9 + X^7 + X^6 + X^3 + 1$, initialisé grâce à la clé $k = k_9k_8 \dots k_0 = 1001001001$. Que remarquez-vous ?
2. Calculer la suite produite par le LFSR de polynôme caractéristique $P(X) = X^3 + 1$, initialisé grâce à la clé 001.

Un système de chiffrement à flot est maintenant constitué d'un simple LFSR $\mathcal{R}$ dont le polynôme caractéristique est $P(X)$

1. Écrire une procédure de chiffrement.
2. Écrire une procédure de déchiffrement.

Testez votre programme grâce au polynôme $P(X) = 1 + X + X^4$. La clé secrète utilisée pour le chiffrement est $k = k_3k_2k_1k_0 = 1101$ et constitue la graine du générateur pseudo-aléatoire correspondant au LFSR $\mathcal{R}$, produisant la suite chiffrante $(s_i)_{i\in\mathbb{N}}$. Le chiffrement s'effectue classiquement en faisant un *ou exclusif* entre les bits du message $M = (m_0, \dots, m_\ell)$ et les bits issus du générateur :

$$c_i = m_i \oplus s_i.$$

Un mot est transformé en binaire en prenant simplement comme encodage pour une lettre l'équivalent binaire de son rang dans l'alphabet, *i.e.*

$$\begin{array}{lcl} A(1) & \mapsto & 00001 \\ B(2) & \mapsto & 00010 \\ C(3) & \mapsto & 00011 \\ & \vdots & \\ X(24) & \mapsto & 11000 \\ Y(25) & \mapsto & 11001 \\ Z(26) & \mapsto & 11010 \end{array}$$

Le mot "AX" est donc transformé en $m_0 \dots m_9 = 0000111000$.

Le chiffré reçu est le suivant :

$$C = c_0 \dots c_{14} = 000001111001000.$$

1. De combien de lettres est formé le message clair ?
2. Donner la suite produite par ce LFSR.
3. En déduire le message alphabétique envoyé.

Remarque : Un algorithme de chiffrement à flot constitué d'un seul LFSR n'est pas sûr. Pour le rendre sûr, on utilise plusieurs LFSR en parallèle, dont on combine les sorties grâce à une fonction booléenne. Vous pouvez transformer votre programme pour obtenir un tel mécanisme de chiffrement.