ÉCOLE POLYTECHNIQUE (PSI*)

CONCOURS D'ADMISSION 2009 **FILIÈRES MP ET PC**

ÉPREUVE D'INFORMATIQUE

(Durée : 2 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation choisi par le candidat doit être spécifié en tête de copie.

⋆ ⋆ ⋆

Chiffrement par blocs

Notation. Dans tout l'énoncé, $[\![a, b[\![$ désigne l'ensemble des entiers naturels supérieurs ou égaux à a et strictement inférieurs à b.

Lorsque l'on souhaite communiquer des données confidentielles, il convient de *chiffrer* ces données, c'est-à-dire de les rendre inintelligibles. Les algorithmes étudiés ici relèvent du chiffrement *symétrique* : une transformation de chiffrement donnée est identifiée par une clé (un entier), qui la désigne et permet également le déchiffrement.

Dans une approche simplifiée du *chiffrement par blocs*, le chiffrement d'un message de taille arbitraire est effectué d'abord en découpant le message en blocs de taille fixée puis en chiffrant chaque bloc. Nous nous limitons ici au chiffrement d'un bloc considéré indépendamment des autres. Dans ce modèle, on se donne un entier $N > 0$, dit *taille* (en pratique N est une puissance de deux). Un bloc (clair ou chiffré) est un entier de $[\![0, N[\![$, et un algorithme de chiffrement est une application de $[\![0, N[\![$ dans $[\![0, N[\![$. Pour permettre le déchiffrement, cette application doit être une permutation de $[\![0, N[\![$ (autrement dit une bijection).

Important. Dans tout le problème, on suppose que le langage de programmation utilisé possède certaines propriétés.

1. Les programmes agissent sur des entiers (naturels) « de taille arbitraire » c'est-à-dire que l'on ignore toutes les questions liées à la taille finie des entiers machine. Autrement dit, on considère que les opérations usuelles (+, * etc.) sont celles des entiers naturels.
2. Il existe deux fonctions `rem`(a, b) et `quo`(a, b) calculant respectivement le reste r et le quotient q de la division euclidienne de a par $b > 0$. Il est rappelé que l'égalité $a = bq+r$ et la condition $r < b$ définissent q et r. Autrement dit, si $a = bq + r$, alors il existe un unique quotient q et un unique reste $r < b$, dont les valeurs sont données précisément par les fonctions `quo` et `rem`.
3. Certaines des fonctions demandées sont spécifiées comme renvoyant un tableau ou une liste. Tableau ou liste sont au choix du candidat. En cas de doute, le candidat est invité à définir les primitives dont il juge avoir besoin et à les employer de façon cohérente dans tout le problème.

I. Approche naïve

On cherche à désigner (dans un premier temps) une application arbitraire de $[\![0, N[\![$ (ensemble à N éléments) dans lui-même. Le nombre total de telles applications est N^N.

Considérons un entier k (une clé) pris dans $[\![0, N^N[\![$. L'entier k s'écrit de manière unique sous la forme :

$$k = a_{N-1}N^{N-1} + \ldots + a_i N^i + \ldots + a_1 N^1 + a_0,$$

où chaque coefficient vérifie $a_i \in [\![0, N[\![$ (c'est l'écriture de k en base N). On considère que k représente l'application f_k de $[\![0, N[\![$ dans lui-même définie par $f_k(0) = a_0$, $f_k(1) = a_1$, etc.

Question 1 Écrire la fonction `DecomposerBase`(N, k) qui prend en arguments la taille N, une clé k de $[\![0, N^N[\![$, et qui renvoie la décomposition de k en base N. En pratique, `DecomposerBase` renvoie donc le tableau ou la liste des a_i, dans l'ordre des i croissants.

En réalité nous nous intéressons aux permutations de $[\![0, N[\![$. On sait qu'il existe $N!$ permutations d'un ensemble de N éléments. Dans la suite logique de la question précédente, considérons donc une clé k prise dans $[\![0, N![\![$. On admet que k s'écrit de manière unique sous la forme :

$$k = a_{N-1}(N-1)! + a_{N-2}(N-2)! + \ldots + a_i i! + \ldots + a_2 2! + a_1 1! + a_0,$$

où les coefficients vérifient $a_i \in [\![0, i+1[\![$. L'écriture ci-dessus est dite *décomposition sur la base factorielle*. Par exemple, pour $N = 4$ et $k = 17$, on a $k = 2 \cdot 3! + 2 \cdot 2! + 1 \cdot 1! + 0$.

Question 2 Écrire la fonction `DecomposerFact`(N, k) qui prend en argument la taille N et une clé k de $[\![0, N![\![$, et qui renvoie la décomposition de k sur la base factorielle.

Une fois k décomposée sur la base factorielle, la permutation σ_k de $[\![0, N[\![$ représentée par k se calcule comme suit. En premier lieu, on considère la séquence $\mathcal{L} = (0, 1, ..., N-1)$ à N éléments. Cette séquence est modifiée au fur et à mesure que les valeurs prises par la permutation σ_k sont calculées.

La première valeur calculée est $\sigma_k(0)$, égal au $1 + a_{N-1}$-ème élément de $\mathcal{L}$ (c'est-à-dire à a_{N-1}). Une fois $\sigma_k(0)$ calculé, cet entier est retiré de $\mathcal{L}$, qui ne contient plus que $N-1$ entiers.

La seconde valeur calculée est $\sigma_k(1)$, égal au $1 + a_{N-2}$-ème élément de $\mathcal{L}$. Une fois $\sigma_k(1)$ calculé, cet entier est retiré de $\mathcal{L}$. Le procédé est répété jusqu'au calcul de $\sigma_k(N-1)$, égal à l'unique élément de $\mathcal{L}$ restant.

Par exemple, dans le cas $N = 4$, $k = 17$ on a : $\sigma_{17}(0) = 2$ ($a_3 = 2$), et $\mathcal{L}$ devient $(0, 1, 3)$. Ensuite $\sigma_{17}(1) = 3$ ($a_2 = 2$), et $\mathcal{L}$ devient $(0, 1)$. Ensuite $\sigma_{17}(2) = 1$ ($a_1 = 1$), et pour finir $\sigma_{17}(3) = 0$.

Question 3 Écrire la fonction `Retirer`(L, ℓ, j) qui prend en argument un tableau L à ℓ éléments, et qui renvoie un tableau de taille $\ell - 1$. Le tableau renvoyé est une copie du tableau L dans laquelle le j-ème élément a été retiré.

Question 4 Écrire la fonction `EcrirePermutation`(N, k) qui prend en arguments la taille N, la clé k de $[\![0, N![\![$, et qui renvoie la permutation σ_k. La permutation sera représentée par le tableau ou la liste des $\sigma_k(i)$, dans l'ordre des i croissants.

Question 5 Écrire les fonctions `Chiffrer`(N, k, b) et `Dechiffrer`(N, k, b), qui prennent en arguments la taille N, la clé k et un bloc b. La fonction `Chiffrer` renvoie $\sigma_k(b)$, tandis que la fonction `Dechiffrer` renvoie l'unique bloc b' tel que $\sigma_k(b') = b$.

II. Réseau de Feistel

Nous prenons ici le parti de fabriquer des permutations particulières. Notre motivation ici est double : (1) réduire la taille des clés (un entier de $[\![0, N![\![$ dans la partie précédente) et (2) effectuer des calculs peu coûteux lors du chiffrement et du déchiffrement.

On commence par fixer la taille à la valeur $N = 2^{64}$. Un bloc b est donc un entier de $[\![0, 2^{64}[\![$. L'ingrédient essentiel du chiffrement est le *réseau de Feistel*. Un réseau de Feistel est une suite de plusieurs opérations, appelées *tours*. Un `tour` est décrit par la figure 1. Sur la figure, *l'entrée* est le bloc $b_i = 2^{32}q_i + r_i$, la *sortie* est $b_{i+1} = 2^{32}q_{i+1} + r_{i+1}$.

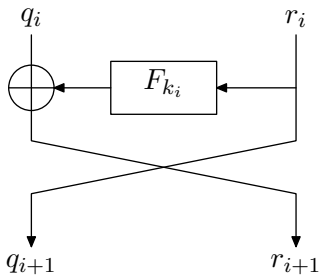

Fig. 1 : Un tour de réseau de Feistel.

La figure peut aussi se lire comme définissant q_{i+1} égal à r_i, et r_{i+1} égal à $q_i \oplus F_{k_i}(r_i)$. Le symbole $\oplus$ désigne ici une opération appelée `xor`. Cette fonction est associative, commutative, et vérifie $\texttt{xor}(\texttt{xor}(x, y), y) = x$ pour tout couple d'entiers (x, y). On suppose que la fonction `xor` est disponible dans le langage de programmation utilisé, accessible sous le nom `xor`. Le symbole F_{k_i} désigne une application sur $[\![0, 2^{32}[\![$, paramétrée par une clé k_i. Par la suite, on suppose donnée une fonction $F(k_i, r)$ qui calcule $F_{k_i}(r)$.

Question 6 Écrire la fonction $\texttt{FeistelTour}(k, b)$ qui prend en argument une clé k et un bloc b (k est un certain k_i, et b est un certain b_i), et renvoie la sortie (notée b_{i+1} ci-dessus) du tour qui utilise la clé k.

Question 7 Écrire la fonction $\texttt{FeistelInverseTour}(k, b)$ qui réalise l'application inverse de la fonction précédente, c'est-à-dire qui calcule et renvoie b_i en fonction de b_{i+1}.

Question 8 Écrire la fonction $\texttt{Feistel}(K, l, b)$ qui prend en entrée le bloc b, et renvoie la sortie d'un réseau de Feistel à ℓ tours. Plus précisément, l'entrée b_0 du premier tour est b, puis l'entrée b_i ($i > 0$) d'un tour est la sortie du tour précédent. Enfin, la sortie du réseau est la sortie b_ℓ du dernier tour. Chaque tour utilise une clé différente. Les clés sont fournies (dans l'ordre) par le tableau K de taille ℓ. Indépendamment du langage de programmation considéré, on supposera qu'un tableau est un argument standard et que ses indices sont les entiers de $[\![0, \ell[\![$.

Question 9 Écrire la fonction $\texttt{FeistelInverse}(K, \ell, b)$ qui effectue l'opération inverse de la fonction précédente. Cette opération inverse est le déchiffrement, et l'identité suivante doit être vérifiée pour tout bloc b : $\texttt{FeistelInverse}(K, \ell, \texttt{Feistel}(K, \ell, b)) = b$.

III. Vérification de propriétés statistiques

Dans cette partie la taille N est fixée à la valeur $N = 2^{64}$, comme dans la partie précédente. On explore la mise en œuvre de critères de qualité du chiffrement. Certains tests couramment employés sont des tests statistiques effectués sur les message chiffrés. Ces tests servent à mettre en évidence des biais indésirables.

On considère le message clair (infini) formé de la séquence des blocs 0, 1, ... Pour une permutation de chiffrement des blocs σ, le message chiffré est donc la séquence des blocs $\sigma(0)$, $\sigma(1)$,...

Les tests portent sur le message chiffré vu comme une séquence de bits, un bit étant un chiffre en base 2, soit 0 ou 1. En fonction d'une longueur paramétrable n, nécessairement multiple de 64, la séquence étudiée est la séquence

$$S_n = \underbrace{1010\ldots1101}_{\sigma(0)(64\text{ bits})}\underbrace{1001\ldots1110}_{\sigma(1)(64\text{ bits})}\ldots\underbrace{1101\ldots0010}_{\sigma(\frac{n}{64}-1)(64\text{ bits})}$$

où par convention, l'écriture binaire (complète) d'un entier x de $[\![0, 2^{64}[\![$, $x = \sum_{i=0}^{63} b_i 2^i$, est la séquence $b_{63}b_{62}\ldots b_1b_0$ (le bit « le plus significatif » apparaît en premier). Dans tout ce qui suit, on considère que la permutation étudiée σ est fixée, et calculée par une fonction **Sigma**(x), qui prend en entrée un entier x de $[\![0, 2^{64}[\![$ et renvoie un entier de $[\![0, 2^{64}[\![$.

Question 10 Écrire la fonction `Sequence`(n) qui construit la séquence S_n ci-dessus, sous la forme d'un tableau de taille n ou d'une liste (on rappelle que n est un multiple de 64). L'ordre des éléments du tableau ou de la liste sera évidemment l'ordre des bits de S_n défini précédemment.

Un premier critère consiste à tester dans quelle mesure les bits 0 et 1 apparaissent avec une fréquence suffisamment proche. Sur un total de n bits ($n \geqslant 1$), on calcule pour cela la valeur $V_1 = \frac{1}{n}(n_0 - n_1)^2$, où n_0 et n_1 représentent respectivement le nombre de bits 0 et 1 dans la séquence de n bits considérée. En fonction de cette valeur V_1, des tables permettent de dire si un biais statistique est visible.

Question 11 Écrire la fonction `CalculerV1`(n) qui détermine la valeur V_1 correspondant à la séquence S_n. Attention, on observera que V_1 n'est pas un entier, il sera représenté en machine par un nombre flottant.

Un second critère généralise le précédent en considérant les séquences de deux bits. Pour n bits ($n \geqslant 2$), on calcule la valeur V_2 donnée par :

$$V_2 = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1,$$

où n_{00}, n_{01}, n_{10}, n_{11} désignent respectivement le nombre d'occurrences des séquences `00`, `01`, `10`, `11`. On notera qu'on autorise les séquences de deux bits à se recouper. Ainsi la séquence de cinq bits 01100 contient exactement une fois chacune des quatre séquences de deux bits possibles.

Question 12 Écrire la fonction `CalculerV2`(n) qui détermine la valeur V_2 correspondant à S_n.