Janvier 2001

Durée : 2 heures. Tous documents autorisés. Calculatrice conseillée !

Exercice 1. Cryptographie (7 pts)

Soit p un nombre premier et soient a, b deux entiers non nuls choisis dans $\{1, \ldots, p-1\}$. On note $\Phi_{a,b,p}$ la fonction de $\mathbb{Z}/p\mathbb{Z}$:

$$\Phi_{a,b,p}(x) = (a.x + b) \bmod p$$

Pour un message dans $\{0, \ldots, p-1\}$, on considère la fonction de codage $E = \Phi_{a,b,p}$.

1. Expliciter la fonction de décodage D associée à E en montrant qu'elle s'écrit sous la forme $D = \Phi_{\alpha,\beta,p}$: expliciter α et β en fonction de a et b.

Correction: *Soit $\alpha = a^{-1}$ mod p et $\beta = -b.a^{-1}$ mod p.*
On a alors $D(E(x)) = a^{-1}(a.x + b) - b.a^{-1}$ mod $p = x$ et, de même, $E(D(x)) = x$.

2. On suppose $p = 43, a = 5, b = 37$; on reçoit le message $E(x) = 28$. Que valait x ?

Correction: 5^{-1} *mod 43 = 26. D'où x = 26.28-26.37 mod 43 = 24* $\Longrightarrow \boxed{x = 24}$

3. On suppose que l'on connaît a, b et p. Soit M un message de n bits avec $n \gg \log_2 p$. Expliquer comment crypter M avec E et donner une estimation du temps nécessaire au cryptage de M avec E en fonction de n et p.

Correction: *On coupe le message $M = (M_1, \ldots, M_k)$ en blocs de $\lfloor \log_2 p \rfloor$ bits; ainsi $\forall i = 1..k$: $0 \leq M_i < p$. Il suffit alors de coder chaque M_i avec E.*
Avec les algorithmes vus dans le poly cela prend un temps $O(\log^2 p)$ pour chaque M_i – et même $O(\log p. \log\log p \log\log\log p)$ –.
Donc en tout un temps $\frac{n}{\log_2 p} O(\log^2 p) = \boxed{O(n \log p)}$.

4. Est-il pertinent d'utiliser les fonctions précédentes comme un système cryptographique à clef publique, la clef publique étant (a, b, p) et la clé privée (α, β, p) ? Justifiez brièvement mais précisément votre réponse.

Correction: *Si a et b sont publiques, il suffit pour calculer α et β d'inverser a modulo p et de faire une multiplication. En utilisant l'algorithme d'Euclide étendu, ceci prend un temps $O(\log^2 p)$. Ce temps est négligeable devant le temps de codage (resp. décodage) avec la clef publique (resp. privée) qui est linéaire en n – $O(n \log p)$ –.*
Il est donc très facile de casser la clef privée à partir de la clef publique: le code ne peut pas être utilisé comme système à clef publique.

5. On désire maintenant utiliser les fonctions précédentes dans un système à clef privée : a, b, p et α, β, p sont privés, connus par Alice et Bob seulement.
Un espion sait qu'Alice envoie en secret à Bob les 2 messages différents x_1 et x_2, avec $0 \leq x_1, x_2 < p$. L'espion voit donc passer sur le réseau $y_1 = a.x_1 + b \bmod p$ et $y_2 = a.x_2 + b \bmod p$. Connaissant (x_1, y_1) et (x_2, y_2), montrer que l'espion peut alors facilement casser le code.

$$a = (y_1 - y_2).(x_1 - x_2)^{-1} \ mod\ p$$

Donc a est facilement calculé en temps $\log^2 p$. Une fois a connu, l'espion peut calculer $b = y_1 - a.x_1 \ mod\ p$, puis α et β comme à la question précedente.
Ce code est donc très facile à casser et ne peut pas être utilisé comme code cryptographique.

Exercice 2. Compression. (7 pts)

1. On cherche à coder des jets successifs (en nombre supposé infini) d'un dé faussé. Les symboles de la source sont notés (1,2,3,4,5,6), et suivent la loi de probabilité d'apparition (0.12,0.15,0.16,0.17,0.18,0.22).

a. Quelle est l'entropie de cette source?

Correction: *H(dé)=2.58*

b. Proposer un code ternaire (sur un vocabulaire à trois chiffres) issu de l'algorithme de Huffman pour cette source. Quelle est sa longueur moyenne? Quelle longueur moyenne minimale peut-on espérer pour un tel code ternaire?

Correction: *sur {0,1,2}, Huffman construit le code (par exemple) : (1->10 , 2->12 , 3->01 , 4->00 , 5->02 , 6-> 2), de longueur moyenne (fixe) égale à* $0.22 * 1 + (1 - 0.22) * 2 = 1.78$ *chiffres. C'est le code optimal (par théorème), on ne pourra obtenir mieux (et toujours moins que* $2.58/log_2(3) = 1.62$*) qu'en codant la source par séquences de chiffres.*

c. Même question pour un code binaire.

Correction: *sur {0,1}, un code possible est : (1->000 , 2->001 , 3->010 , 4->011 , 5->10 , 6->11), de longueur moyenne* 2.6. *C'est aussi le code optimal.*

2. L'algorithme de Huffman construit un code où des mots de source de longueur fixes sont codés par des mots de code de longueur variable. L'organisation de la mémoire où l'on stocke le code impose parfois une longueur fixe aux mots de code, et on code alors des séquences de longueur variable des chiffres de la source.

Les codes dits de Tunstall sont les codes optimaux qui suivent ce principe. En voici la méthode de construction dans le cas d'un vocabulaire binaire. Si la longueur choisie des mots de code est k, il faut trouver les 2^k séquences de chiffres de source qu'on va choisir de coder. Au départ, l'ensemble des candidats est l'ensemble des mots de longueur 1 (ici, {1,2,3,4,5,6}). Soit dans cet ensemble le mot le plus probable (ici, 6). On construit toutes les séquences réalisables en rajoutant un chiffre à ce mot, et on remplace ce mot par ces séquences dans l'ensemble des candidats (ici, on obtient alors {1,2,3,4,5,61,62,63,64,65,66}). On recommence alors cette opération tant que la taille de l'ensemble des candidats n'est pas strictement supérieure à 2^k (on arrête avant d'avoir dépassé cette valeur). On code alors toutes ces séquences par des mots de longueur k.

a. Sur un vocabulaire binaire, construire un code de Tunstall pour le dé, pour des longueurs de mots de code $k = 4$.

Correction: *On code les séquences {1,2,3,4,51,52,53,54,55,56,61,62,63,64,65,66} (au nombre de 16) avec les 16 mots de longueur* 4.

Correction: *Selon le code choisi (dont ne dépent pas la longueur des séquences), on peut avoir par exemple : Tunstall(66->1111 et 64->1100) ->11111100 et Huffman ->111111001.*

c. Pour chaque mot de code, calculer le nombre de bits de code nécessaires par bit de source. En déduire la longueur moyenne *par bit de source* du code de Tunstall

Correction: *Pour coder les séquences de deux chiffres, on a besoin de 4 bits, soit un rendement de 2, et de même pour les mots de un chiffre, soit un rendement de 4.*
La longueur moyenne par bit de source est donc : $2 * 0.22 + 2 * 0.18 + 4 * 0.60 = 3.2$

Exercice 3. Codes correcteurs (6 pts)

On communique sur un réseau qui transmet des octets avec un taux d'erreur de 1%.
Pour y remédier, lorsqu'on envoie n octets, on veut détecter et corriger $0.01 \times n$ erreurs.
Expliquer comment construire un code correcteur de type Reed-Solomon en précisant :

a. le corps de base et la valeur choisie pour n;

b. le degré du polynôme générateur;

c. le nombre maximal d'erreurs détectées;

d. la dimension d'une matrice génératrice du code. Comment s'écrit cette matrice à partir des coefficients du polynôme générateur ?

S'il vous reste du temps, pour bonus uniquement...
Soit $P = \alpha^d + ...$ un polynôme irréductible de degré d dans $\mathbb{F}_2[\alpha]$.
P est dit **primitif** si et seulement si α est générateur de $\mathbb{F}_{2^d}^\star$, i.e. $\mathbb{F}_{2^d}^\star \equiv \{\alpha^i \bmod P;\ 1 \leq i < q\}$.
Pour $d = 3, .., 10$, les polynômes suivants à coefficients dans $\mathbb{F}_2$ sont primitifs :

degré d	Polynôme primitif
3	$1 + \alpha + \alpha^3$
4	$1 + \alpha + \alpha^4$
5	$1 + \alpha^2 + \alpha^5$
6	$1 + \alpha + \alpha^6$

degré d	Polynôme primitif
7	$1 + \alpha^3 + \alpha^7$
8	$1 + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^8$
9	$1 + \alpha^4 + \alpha^9$
10	$1 + \alpha^3 + \alpha^{10}$

e. donner le polynôme utilisé pour implémenter le corps de base et expliquer brièvement comment réaliser les opérations d'addition et de multiplication;

f. donner l'expression du polynôme générateur en fonction de α.

Correction:

a Comme on envoie des octets, on choisit $\mathbb{F}_{256}$*, qui a* $q = 256 = 2^8$ *éléments. comme corps de base. Un code de Reed-Solomon impose alors de choisir* $n = q - 1 = 255$.

b Pour corriger au moins $.01n$ *erreurs, il faut avoir pouvoir corriger* $2.56 < 3$ *erreurs, donc choisir un code de distance* $\delta \geq 2 * 3 + 1 = 7$. *Avec un code de Reed-Solomon, le polynôme générateur est de degré* $r = \delta - 1 = 6$.

c Le nombre maximal d'erreurs détectées est 6.

sous forme. Elle s'écrit sous la forme

$$\begin{bmatrix} g_0 & g_1 & \dots & g_6 & 0 & \dots & 0 \\ 0 & g_0 & g_1 & \dots & g_6 & \dots & 0 \\ & & & \dots & & & \end{bmatrix}$$

e *On choisit donc* $P = 1+\alpha^2+\alpha^3+\alpha^4+\alpha^8$ *pour implémenter* $\mathbb{F}_{256} = \mathbb{F}_2[\alpha]/P$. *Soit* $X = \sum_{i=0}^{7} x_i\alpha^i$ *avec* $x_i \in \{0,1\}$ *; X est représenté par l'octet* $(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$. *Soit* $Y = \sum_{i=0}^{7} y_i\alpha^i$ *avec* $y_i \in \{0,1\}$ *un autre élément du corps.*
X + Y est alors représenté par l'octet $(x_0 + y_0, \dots x_7 + y_7)$.
Pour X.Y, on calcul le produit de polynômes X.Y mod P; les coefficients de ce polynôme permettent d'obtenir le codage de XY sous forme d'octet.

f *Comme P est primitif,* α *un élément générateur de* $\mathbb{F}_{256}^{\star}$ *(une racine primitive 255-ième de l'unité). Il suffit donc de choisir comme polynôme générateur* $g = \prod_{i=1..6}(X-\alpha^i) \mod P = \sum_{i=0}^{6} g_i X^i$, *où chaque* g_i *est un élément de* $\mathbb{F}_{256}$ *représenté par un octet (cf question précédente).*