Examen du vendredi 7 janvier 2022, de 9h à 11h.

*Sont autorisés : une calculatrice et résumé de cours manuscrit format A4 recto-verso.*
*Autres documents et portables interdits.*
*Le sujet comporte 2 pages. Le barême est indicatif.*

## 1. BASES (5 POINTS)

*La base 45 est un système d'encodage utilisé par certaines applications, par exemple les QR codes du passe sanitaire. On écrit un nombre en base 45 avec les caractères suivants :* `0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ␣$%*./:";`
*Ainsi* `;1A` *en base 45 correspond à l'entier* $44 \times 45^2 + 1 \times 45 + 10$ *en base 10.*

(1) *Déterminer la représentation en base 45 de 65535.*
$65535 = 1456 \times 45 + 15 = 32 \times 45^2 + 16 \times 45 + 15$ donc s'écrit `WGF`

(2) *En déduire n le nombre maximal de caractères qui sont nécessaires pour représenter 2 octets en base 45. Peut-on obtenir le code* `$00` *?*
Donc $n = 3$. On ne peut pas obtenir `$00` qui correspond à $37 \times 45^2 = 74925 > 65535$

(3) *On décide de représenter 2 octets par n caractères en base 45, en ajoutant des 0 au début si nécessaire. Voici un extrait de 6 caractères d'un QR code* `6BF0D`*; Quels sont les octets correspondants ?*
`6*45^2+11*45+15=12660=49*256+116` et `13*45+44=629=2*256+117` donc `[49,116,2,117]`

(4) *On suppose qu'on dispose d'une fonction* `num` *qui prend en argument un caractère d'écriture en base 45 et renvoie un entier entre 0 et 44. Donner au choix en langage naturel ou en C ou en Python un algorithme prenant en entrée une chaine de caractère représentant un QR code en base 45 et renvoyant un tableau (ou une liste) d'octets. On testera la validité de la chaine de caractères.*

```
# naturel
Fonction f:
  argument la chaine s, renvoie une liste l d'octets ou une chaine erreur
  Initialiser l a liste vide
  si la longueur de s n'est pas divisible par 3 renvoyer "longueur invalide"
  pour j de 0 jusqu'a la longueur de s sur 3 exclue faire
     c1 <- num(s[3*j])
     c2 <- num(s[3*j+1])
     c3 <- num(s[3*j+2])
     si c1, c2 ou c3 n'est pas dans [0,44] renvoyer "caractere invalide"
     c <- c1*45**2+c2*45+c3
     si c>=256**2 alors renvoyer "chaine invalide"
     ajouter a l le quotient de c par 256
     ajouter a l le reste de c par 256
  renvoyer l


# Python
```

```
def f(s):
    l=[]
    if len(s) % 3: return "longueur invalide"
    for j in range(len(s)/3):
        c1=num(s[3*j])
        c2=num(s[3*j+1])
        c3=num(s[3*j+2])
        if c1<0 or c2<0 or c3<0: return "caractere invalide"
        c=c1*45**2+c2*45+c3
        if c>=256**2: return "chaine invalide positions "+(3*j)
        l.append(c // 256)
        l.append(c % 256)
    return l
```

```
// C: s la chaine de caracteres, l le tableau d'octet
// renvoie 0 si ok, <0 sinon
int f(const char * s,unsigned char * l){
  int n=strlen(s);
  if (n%3) return -1; // invalide
  for (int j=0;j<n/3;++j){
    int c1=num(s[3*j]),c2=num(s[3*j+1]),c3=num(s[3*j+2]);
    if (c1<0 || c2<0 || c3<0) return -2;
    int c=(c1*45+c2)*45+c3;
    if (c>0xffff) return -3;
    *l=c/256; ++l;
    *l=c%256; ++l;
  }
  return 0;
}
```

Voir une implémentation complète en C (non demandée) en fin de corrigé.

## 2. CRYPTOGRAPHIE DE HILL (6 POINTS)

Soit $p = 31$ et la matrice à coefficients dans $\mathbb{Z}/p\mathbb{Z}$

$$A = \begin{pmatrix} 1 & 5 \\ 5 & 4 \end{pmatrix} \ [31]$$

(1) *Un agent secret de nom de code BOND utilise la cryptographie de Hill pour envoyer des messages, en codant les caractères de A à Z par les entiers de 0 à 25, puis les caractères +−∗/^ par les entiers 26 à 30. Déterminer les 2 vecteurs de* $(\mathbb{Z}/p\mathbb{Z})^2$ *correspondant à son nom de code, puis les 2 vecteurs cryptés correspondant.*
$v_1$=[1,14] et $v_2$=[13,3], on multiplie par $A$ et on obtient $w_1 = [9,30]$ et $w_2 = [28,15]$

(2) *Montrer que A est inversible et calculer son inverse.*
Par l'algorithme de Gauss

$$\begin{pmatrix} 1 & 5 & 1 & 0 & \\ 5 & 4 & 0 & 1 & L_2 - 5L_1 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 5 & 1 & 0 \\ 0 & 10 & -5 & 1 \end{pmatrix}$$

On inverse 10 modulo 31, comme $3 \times 10 = 31 - 1$, l'inverse de 10 est -3, on multiplie $L_2$ par -3

$$\begin{pmatrix} 1 & 5 & 1 & 0 & L_1 - 5L_2 \\ 0 & 1 & 15 & -3 & \end{pmatrix}$$
$$\begin{pmatrix} 1 & 0 & 19 & 15 \\ 0 & 1 & 15 & -3 \end{pmatrix}$$

d'où

$$A^{-1} = \begin{pmatrix} 19 & 15 \\ 15 & -3 \end{pmatrix} \pmod{31}$$

*Vérifiez en décryptant les 2 vecteurs de la question précédente.*

$$\begin{pmatrix} 19 & 15 \\ 15 & -3 \end{pmatrix} \begin{pmatrix} 9 \\ 30 \end{pmatrix} = \begin{pmatrix} 621 \\ 45 \end{pmatrix}$$

qui est bien [1,14] modulo 31. De même pour le deuxième vecteur.

(3) *Montrer que le nombre de matrices de cryptage possible est inférieur à* $31^4$.
Il y a 4 coefficients dans une matrice 2,2 et chaque coefficient peut prendre 31 valeurs, donc on a au plus $31^4$ matrices possibles, en fait un peu moins puisque toutes les matrices ne sont pas inversibles modulo 31.
*Est-ce suffisant pour assurer la sécurité des échanges ?*
Non, avec moins d'un million de clefs possibles, on peut très rapidement tester toutes les clefs (inverses) possibles.

(4) *Un espion du camp adverse intercepte un message de BOND. On suppose que BOND termine son message par les deux vecteurs de son nom de code crypté. L'espion du camp adverse peut-il en déduire la matrice A ?*
La matrice $A$ va vérifier $Av_1 = w_1$ et $Av_2 = w_2$. Soient $a, b, c, d$ les 4 coefficients de $A$, on a donc le système

$$\begin{cases} a + 14b &= 9 \\ c + 14d &= 30 \\ 13a + 3b &= 28 \\ 13c + 3d &= 15 \end{cases}$$

Ce système se découple en deux système 2, 2. On a $a + 14b = 9$ donc $a = 9 - 14b$ on remplace dans $13a + 3b = 28$ ce qui donne $13(9 - 14b) + 3b = 28$ qui permet de déterminer $b$ car $13 \times (-14) + 3 = -179 \neq 0 \pmod{31}$, puis $a$. De même pour $c$ et $d$.

(5) *Peut-on améliorer la sécurité en prenant une valeur de p plus grande, par exemple* $p = 65537$ *ou un nombre premier beaucoup plus grand de 1024 bits ?*
Si l'agent secret signe, on pourra déterminer la matrice clef secréte en résolvant le système. S'il ne signe pas, le nombre de clefs possibles devient assez grand (de

l'ordre de la vingtaine de milliards de milliards au moins), mais la sécurité n'est pas assurée pour autant, car on peut se lancer dans une analyse fréquentielle de paires de caractères. Il faudrait augmenter la taille de la matrice et injecter des composantes aléatoires.

## 3. Secret commun (9 points)

On présente une méthode permettant à deux personnes, Alice et Bob, de créer un secret commun sans avoir besoin de se rencontrer. Pour expliquer cette méthode, on commence par travailler dans $\mathbb{Z}/19\mathbb{Z}$. Puis on travaillera dans $\mathbb{Z}/p\mathbb{Z}$ pour un entier premier $p$.

3.1. **Exemple jouet.** Alice et Bob choisissent donc de travailler dans $\mathbb{Z}/19\mathbb{Z}$ et d'utiliser $g = 2$ qui est un générateur de $\mathbb{Z}/19\mathbb{Z}^*$. Alice choisit un entier secret $a$ puis calcule le reste $A$ de la division de $g^a$ par 19 et l'envoie à Bob. Bob choisit un entier secret $b$ puis calcule le reste $B$ de la division de $g^b$ par 19 et l'envoie à Alice. Alice calcule alors le reste de $B^a$ par 19, Bob calcule le reste de $A^b$ par 19, ces nombres sont identiques, c'est leur secret commun.

(1) *Alice choisit $a = 7$ et Bob choisit $b = 13$. Donner A et B puis vérifier que $A^b = B^a \pmod{19}$.*
$A = 2^7 \pmod{19} = 14, B = 2^{13} \pmod{19} = 3$ et $14^{13} \pmod{19} = 2 = 3^7 \pmod{19}$

(2) *Que se passe-t-il si Alice choisit $a = 25$ ?*
Rien ne change, car $25 = 7 \pmod{18}$ et $2^{18} = 1 \pmod{1}9$ (théorème de Lagrange).
*À quelle valeur maximale pour a et b Alice et Bob peuvent-ils se restreindre ?*
17

(3) *Vérifier que 2 est bien un générateur de $\mathbb{Z}/19\mathbb{Z}^*$.*
Les facteurs premiers de $18 = 19 - 1$ sont 2 et 3, il faut donc vérifier que $2^{18/2} \neq 1 \pmod{19}$ et $2^{18/3} \neq 1 \pmod{19}$, ce qui est bien le cas (on trouve 18 et 7).
*Déterminer la table de toutes les puissances de 2 dans $\mathbb{Z}/19\mathbb{Z}$*
1,2,4,8,16,13,7,14,9,18,17,15,11,3,6,12,5,10

(4) *Alice et Bob choisissent deux autres entiers a et b et s'envoient $A = 4$ et $B = 17$. En utilisant la table, déterminer les valeurs de a et b.*
$a = 2$ et $b = 10$, *Quelle est la valeur du secret commun ?*
Secret commun $17^2 = 4 = 4^{10} \pmod{)}19$.

3.2. **Sécurité.** On a vu qu'une personne qui connait les entiers $A$ et $B$ (par exemple en espionnant les échanges entre Alice et Bob) pouvait calculer le secret commun si on travaille dans $\mathbb{Z}/19\mathbb{Z}$. Pour espérer sécuriser le secret commun, il faut travailler dans $\mathbb{Z}/p\mathbb{Z}$ avec $p$ un nombre premier plus grand.

(1) Commençons par essayer avec $p = 65537$ et $g = 3$.

(a) *Quel est l'ordre de $\mathbb{Z}/p\mathbb{Z}^*$ ?*
$p - 1 = 65536 = 2^{16}$
*Vérifier que 3 est un générateur de $\mathbb{Z}/p\mathbb{Z}^*$.*
$p - 1$ a un seul facteur premier, 2, il suffit donc de vérifier que $3^{(2^{15})} \neq 1 \pmod{p}$.
Il faut avoir recours à l'algorithme de la puissance rapide, on calcule 15 fois de suite un carré, on obtient successivement : 3, 9, 81, 8561, 54449, 61869, 19139, 15028, 282, 13987, 8224, 65529, 64, 4096, 65281, 65536.

(b) *Si Alice choisit $a = 12345$, combien doit-elle effectuer de multiplications pour calculer A par l'algorithme de la puissance rapide ?*
On écrit 12345 en base 2, on obtient `0b11000000111001`, 14 bits et 6 bits à 1, donc il faut effectuer 19 multiplications.

(c) *À quelle valeur maximale pour a et b Alice et Bob peuvent-ils se restreindre ?*
À $p-1$, donc 65536.

(d) *Quelle est la taille de la table des puissances de 3 modulo p ? Comparez avec la question (b).*
Il y a $p-1 = 65536$ puissances dans la table. C'est beaucoup plus que le nombre de multiplications à effectuer pour que Alice et Bob calculent leur secret commun
*La sécurité du secret commun vous semble-t-elle suffisante ?*
Non, on peut rapidement déduire $a$ et $b$ de $A$ et $B$ en cherchant la valeur de $A$ et $B$ dans la table.

(2) *On prend maintenant un nombre premier dont l'écriture en base 2 comporte exactement 1024 bits, et tel que $g = 2$ est un générateur de $\mathbb{Z}/p\mathbb{Z}^*$. La sécurité du secret commun vous semble-t-elle suffisante ?*
Oui (sauf s'il existe d'autres attaques), car il est impossible de stocker une table de $p-1$ entiers avec $p > 2^{1023}$ (et parcourir une telle table prendrait un temps proportionnel à $p$). Alice et Bob peuvent toujours calculer le secret commun, car ils devront effectuer de l'ordre d'un millier de multiplications (et restes de division par $p$).

Implémentation complète en C du 1.
Il y avait une erreur d'énoncé (47 caractères au lieu de 45 dans la table), qu'on a corrigé pendant le partiel en enlevant les caractères + et - pour maintenir le `;` en dernier avec le code 44 pour la question 1.3. La vraie table utilise +− mais pas `";`, on peut passer de l'une à l'autre en modifiant la directive de compilation `#if 0` en `#if 1`.

```
#include <string.h>
#include <stdio.h>

int num(const char c){
  // conversion en utilisant un tableau initialise au premier appel
  // on peut aussi utiliser une variable globale pour tab et
  // initialiser dans main (c'est plus rapide)
#if 0 // remplacer 0 par 1 correspond a l'enonce (erreur d'enonce)
  const char s[]="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ $%*./:\";";
#else // table utilisee en realite
  const char s[]="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ $%*+-./:";
#endif
  static short int tab[256]={-2}; // tab[0]==-2 marque le premier appel
  if (tab[0]==-2){ // initialisation
    for (int i=0;i<sizeof(tab)/sizeof(short int);++i)
      tab[i]=-1;
    for (int i=0;i<strlen(s);++i){
      tab[(int) s[i]]=i;
    }
```

```
  }
  return tab[(int) c];
}

int f(const char * s,unsigned char * l){
  int n=strlen(s);
  if (n%3) return -1; // invalide
  for (int j=0;j<n/3;++j){
    int c1=num(s[3*j]),c2=num(s[3*j+1]),c3=num(s[3*j+2]);
    if (c1<0 || c2<0 || c3<0) return -2;
    int c=(c1*45+c2)*45+c3;
    if (c>0xffff) return -3;
    *l=c/256; ++l;
    *l=c%256; ++l;
  }
  return 0;
}

int main(int argc,const char ** argv){
  // exemple execution: ./a.out "6BF0D;"
  if (argc<2) {
    printf("Syntaxe %s QR-code\n",argv[0]);
    return -1;
  }
  int l=strlen(argv[1]);
  unsigned char tab[2*l/3]; // malloc necessaire avec certains compilateurs
  if (f(argv[1],tab)){
    printf("Invalid QR-code %s\n",argv[1]);
    return -1;
  }
  for (int i=0;i<2*l/3;++i){
    printf("%i ",tab[i]);
  }
  printf("\n");
  return 0;
}
```