Université de Yaoundé 2005–2006

Master en informatique

TD Cryptographie à Clé Secrète
Analyse de Fréquence

Sebastien.Varrette@imag.fr

L'objectif de ce TD est la réalisation en C d'un programme d'analyse de fréquence sur un texte chiffré à l'aide d'un chiffrement affine. On veillera à programmer de façon modulaire et à utiliser l'outil `Makefile` pour automatiser la compilation. Le fichier `main.c` contiendra la fonction `main`. On veillera à bien tester les différentes fonctions demandées. Pour ceux qui le souhaitent, une archive contenant notamment un Makefile générique est disponible sur ma page : `http://www-id.imag.fr/~svarrett/enseignements.html`

1 Préliminaires

Fichiers associés : `tools.c` et `tools.h`

1.1 Calcul du pgcd par l'algorithme d'Euclide

Théorème 1.1 (Euclide)**.** *Soit a, $b \in \mathbb{N}$ / $a \leq b$. Soit r le reste de la division euclidienne de a par b. Alors pgcd(a,b)= pgcd(b,r).*

L'algorithme d'Euclide consiste donc à itérer les manipulations suivantes :
- Effectuer la division euclidienne de a par b. Soit r le reste.
- Remplacer a par b et b par r (on a donc $0 \leq r < b$ d'après la définition de la division euclidienne).

Le P.G.C.D. est le dernier reste non nul. (Si b divise a, le P.G.C.D. est b et par convention, pgcd(a,0)=a)
- Implémenter la fonction (en version itérative ou récursive) :

```
int pgcd(int a, int b);
```

On prendra garde à bien traiter le cas des paramètres négatifs.

1.2 Calcul de l'inverse modulaire par l'algorithme d'Euclide Etendu

Il existe une version étendue de l'algorithme d'Euclide, appelée "Euclide étendu", qui permet de calculer les coefficients de Bezout dont la définition est rappelée maintenant :

Théorème 1.2 (Bezout)**.** *Soient a, $b \in \mathbb{Z}$ dont au moins un est non nul et $d = pgcd(a, b)$. Alors il existe un couple d'entiers (u, v) tels que*

$$au + bv = d$$

Les entiers u et v sont appelés coefficients de Bezout.

L'algorithme d'Euclide étendu permet de construire effectivement les coefficients de Bezout. L'idée est la suivante. Supposons que a et b soient des entiers positifs et que $b \neq 0$ et soit $a = bq + r$ avec $0 \leq r < b$ la division euclidienne de a par b. Soient u' et v' les coefficients de Bezout de b et r. Alors :

$$\begin{aligned} d &= \text{pgcd}(a,b) = \text{pgcd}(b,r) \\ &= bu' + rv' = bu' + (a - bq)v' \\ &= av' + (u' - qv')b \end{aligned}$$

Ainsi, on peut prendre $u = v'$ et $v = u' - qv'$; les coefficients de Bezout de (a, b) peuvent donc être calculés à partir des coefficients de Bezout de (b, r). Cette remarque conduit directement à l'algorithme de calcul des coefficients de Bezout :

Données: a, b entiers
Résultat: (d, u, v) tels que $d = \text{pgcd}(a, b) = a.u + b.v$
si $a < 0$ **alors** $a \leftarrow -a$;
si $b < 0$ **alors** $b \leftarrow -b$;
$u \leftarrow 1, \quad v \leftarrow 0;$
$u_{\text{aux}} = 0, \quad v_{\text{aux}} = 1;$
tant que $b > 0$ **faire**
    effectuer la division euclidienne $a = bq + r$;
    $a \leftarrow b,\ b \leftarrow r$;
    $\text{tmp} \leftarrow u_{\text{aux}},\ u_{\text{aux}} \leftarrow u - qu_{\text{aux}},\ u \leftarrow \text{tmp}$;
    $\text{tmp} \leftarrow v_{\text{aux}},\ v_{\text{aux}} \leftarrow v - qv_{\text{aux}},\ v \leftarrow \text{tmp}$;
fin
si $a < 0$ **alors** $u \leftarrow -u$;
si $b < 0$ **alors** $v \leftarrow -v$;
retourner (a, u, v) ;

– Utiliser cet algorithme en pseudo-code pour implémenter la fonction :

```
int bezout(int a, int b, int * u, int * v);
```

Cette fonction renvoie l'entier $d = pgcd(a, b)$ et u et v sont tels que $d = a.u + b.v$.

– En particulier, si $d = 1$, alors $u = a^{-1} \mod b$. En utilisant la fonction `bezout`, implémenter la fonction

```
int inv_mod(int a, int n, int * res);
```

permettant de calculer l'inverse modulaire de a modulo n. Le resultat sera stocké à l'adresse pointée par `res`. Cette fonction renverra 0 en cas de succès, 1 en cas d'échec (lorsque a n'est pas inversible modulo n).

2 Chiffrement affine

Fichiers associés : `chiffrement_affine.c` et `chiffrement_affine.h`
Dans toute la suite, on supposera avoir défini la constante `A_SIZE` dans le fichier `chiffrement_affine.h` de la façon suivante :

```
#define A_SIZE 26
```

2.1 Implémentation d'une bijection

On considère un alphabet $\mathcal{A}$ composé des lettres en minuscule (a-z). Dans toute la suite, un texte sera composé de caractères de $\mathcal{A}$ ainsi que de caractères de liaison (espace, le point "." et la virgule ",") (ces derniers étant supposés non chiffrés dans la procédure de chiffrement)
On considère la bijection :

$$\mathcal{A} \longrightarrow \mathbb{Z}_{26}$$

Le tableau 1 donne la liste des codes ASCII en décimal

code	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	NP	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	SP	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	-	DEL		

TABLE 1 – Codes ASCII en décimal

– Ecrire les fonctions :

int char_to_Z26(**char** c, **int** ∗ res);

int Z26_to_char(**short int** i, **char** ∗ res);

Ces fonctions permettent d'établir la correspondance entre les éléments de $\mathcal{A}$ et ceux de $\mathbb{Z}_{26}$. Elles renverront 0 en cas de succès, 1 en cas d'erreur (l'argument `c` n'est pas une lettre minuscule par exemple). Dans tous les cas, `res` est un pointeur sur le résultat.

2.2 Implémentation du chiffrement et du déchiffrement affine

On note maintenant $\mathbb{Z}_{26}^{\times}$ l'ensemble des éléments inversibles de $\mathbb{Z}_{26}$. On rappelle que pour n et a deux entiers avec $n \geq 2$, les affirmations suivantes sont équivalentes :

1. $\bar{a}$ est un générateur du groupe additif $(\mathbb{Z}_n, +)$.
2. $\bar{a}$ est un élément inversible dans l'anneau $\mathbb{Z}_n$.
3. $\text{pgcd}(a, n) = 1$.

On rappelle que $|\mathbb{Z}_{26}^{\times}| = \varphi(26) = 12$ (ou $\varphi(n)$ est l'indicatrice d'Euler [1]).

1. La fonction φ peut être calculée par la relation :

$$\varphi(n) = n \prod_{p \text{ premier}, p|n} (1 - \frac{1}{p})$$

– Expliciter $\mathbb{Z}_{26}^{\times}$

Pour $(a, b) \in \mathbb{Z}_{26}^{\times} \times \mathbb{Z}_{26}$, on considère la fonction de chiffrement affine $e_{(a,b)}$ définie de la façon suivante :

$$\begin{array}{rccc} e_{(a,b)} : & \mathbb{Z}_{26} & \longrightarrow & \mathbb{Z}_{26} \\ & x & \longrightarrow & e_{(a,b)}(x) = a * x + b \end{array}$$

– En utilisant la clé $K = (15, 7)$, donner par exemple le chiffrement pour $x \in \{0, 1, 2\}$.
– Expliciter la fonction de déchiffrement affine $d_{(a,b)}$.
– Implémenter les fonctions :

```
int chiffrement_affine_Z26(int a, int b, int x, int * res);
int dechiffrement_affine_Z26(int inv_a, int b, int y, int * res);
```

permettant de calculer le chiffrement affine de x (resp. le déchiffrement affine de y) en utilisant la clé $K = (a, b)$ (pour le déchiffrement, `inv_a` $= a^{-1}$ mod 26 sous réserve d'existence). Le résultat sera stocké à l'adresse pointée par `res`. Ces fonctions renverront 0 en cas de succès, -1 lorsque la clé sera invalide ($a \notin \mathbb{Z}_{26}^{\times}$ typiquement) et 1 en cas d'erreur ($x \notin \mathbb{Z}_{26}$ par exemple). Dans ce dernier cas, le resultat `*res` sera égal à l'entrée x (resp. y).

Remarque : idéalement, on aurait dû représenter les éléments de $\mathbb{Z}_{26}$ par des `char`. le type `int` a été préféré pour limiter les confusions.

2.3 Chiffrement et du déchiffrement affine d'un texte

On souhaite étendre les fonctions `chiffrement_affine_Z26` et `dechiffrement_affine_Z26` au chiffrement/déchiffrement d'un fichier texte. Dans ce texte, seuls les éléments de $\mathcal{A}$ seront chiffrés.

– Implémenter les fonctions

```
int chiffrement_fichier_Z26(int a, int b, char * filein , char * fileout );
int dechiffrement_fichier_Z26(int a, int b, char * filein , char * fileout );
```

permettant de chiffrer (resp. déchiffrer) le fichier `filein`, produisant ainsi le fichier `fileout` (qui sera écrasé au besoin). Ces fonctions renverront 0 en cas de succès, 1 en cas d'erreur (dans ce cas, un message explicite sera envoyé sur la sortie d'erreur pour expliciter le problème).
– Testez vos deux fonctions sur des exemples de textes (dont on veillera à uniformiser la casse et à retirer les caractères accentués avant traitement). On pourra utiliser l'outil shell **diff(1)** pour comparer les textes initiaux à ceux produis après déchiffrement.
– Pour faciliter la cryptanalyse, on demande également de réaliser les fonctions

```
int chiffrement_fichier_n_Z26(int a, int b, int nb_char, char * filein );
int dechiffrement_fichier_n_Z26(int a, int b, int nb_char, char * filein );
```

qui affichent à l'écran le chiffrement/déchiffrement affine à partir de la clé $K = (a, b)$ des `nb_char` premiers caractères du fichier `filein`.

3 Analyse de fréquence et cryptanalyse du chiffrement affine

Fichiers associés : `analyse_frequence.c` et `analyse_frequence.h`
Si le texte chiffré par le chiffrement affine semble incompréhénsible, il ne modifie pas pour autant la fréquence d'apparition des différentes lettres qui composent le texte. La répartition statistique des lettres dans un texte écrit en français est fourni dans le tableau 2.

A	8.11 %	N	7.68 %
B	0.81 %	O	5.20 %
C	3.38 %	P	2.92 %
D	4.28 %	Q	0.83 %
E	17.69 %	R	6.43 %
F	1.13 %	S	8.87 %
G	1.19 %	T	7.44 %
H	0.74 %	U	5.23 %
I	7.24 %	V	1.28 %
J	0.18 %	W	0.06 %
K	0.02 %	X	0.53 %
L	5.99 %	Y	0.26 %
M	2.29 %	Z	0.12 %

TABLE 2 – Fréquence d'apparition des lettres en français

– Toto a trouvé un papier sur lequel se trouve le message chiffré suivant :

`YN PHEVBFVGR RFG HA IVYNVA QRSNHG!`

Appliquer l'analyse fréquentielle pour déterminer le message initial.
On définit la structure `letter_freq` de la façon suivante :

```
typedef struct {
    char   c;     // le caractère
    double freq; // la fréquence d'apparition, en %
} letter_freq_t ;
```

Cette structure permet de regrouper les informations relatives à une analyse fréquentielle. Ainsi, le tableau 2 se déclinera de la façon suivante :

```
letter_freq_t  TAB_REF_FR[A_SIZE] = {
    {'a', 8.11}, {'b', 0.81}, {'c', 3.38}, {'d', 4.28}, {'e', 17.69}, {'f', 1.13},
    {'g', 1.19}, {'h', 0.74}, {'i', 7.24}, {'j', 0.18}, {'k', 0.02},  {'l', 5.99},
    {'m', 2.29}, {'n', 7.68}, {'o', 5.20}, {'p', 2.92}, {'q', 0.83},  {'r', 6.43},
    {'s', 8.87}, {'t', 7.44}, {'u', 5.23}, {'v', 1.28}, {'w', 0.06},  {'x', 0.53},
    {'y', 0.26}, {'z', 0.12 }
};
```

– Implémenter la fonction

```
void count_letters_in_text (int tab[A_SIZE], FILE * fd, int * nb_char);
```

qui compte les occurences des lettres de $\mathcal{A}$ dans le fichier (déjà ouvert) `fd` (ainsi, conformément à la bijection initiale, `tab[0]` correspond au nombre d'occurences de la lettre 'a' dans le texte, `tab[1]` à la lettre 'b' etc...). `*nb_char` correspondra au nombre d'éléments de $\mathcal{A}$ lus dans le fichier (certainement différent du nombre de caractères du fichier dans la mesure où seuls les éléments de $\mathcal{A}$ seront chiffrés).

– À l'aide de la fonction **qsort(3)**, implémenter la fonction

void sort_by_letter_freq (letter_freq_t tab[A_SIZE]);

qui permet de trier le tableau `tab` en fonction des fréquences d'apparition (champ `freq`) pour obtenir le tableau `res`.

Indice : il faudra implémenter la fonction **int** compare(**const void** ∗ a, **const void** ∗ b);

– Afin de faciliter la cryptanalyse, il convient d'afficher à l'écran de façon lisible le résultat de l'analyse de fréquence sur le texte à déchiffrer et la version triée du tableau 2. Implémenter la fonction

void show_freq_comparison(letter_freq_t tab[A_SIZE], **int** nb_line_to_show);

réalisant cet affichage pour les `nb_line_to_show` premières lignes (les plus significatives). Voici un exemple l'appel de cette fonction avec `nb_line_to_show=7` :

```
------------------+------------------
   Texte analysé  |  Reference (fr)
------------------+------------------
  n(13) :  16.50% |  e( 4) :  17.69%
  i( 8) :  10.41% |  s(18) :  8.87%
  a( 0) :   8.38% |  a( 0) :  8.11%
  f( 5) :   8.12% |  n(13) :  7.68%
  t(19) :   7.61% |  t(19) :  7.44%
  w(22) :   7.36% |  i( 8) :  7.24%
  l(11) :   6.60% |  r(17) :  6.43%
                [...]
```

– Utiliser les fonctions précédentes pour réaliser la fonction

int get_freq_analysis (**char** ∗ filein , **int** nb_line_to_show);

qui effectue l'analyse fréquentielle du fichier `filein`. Le second argument sera passé en paramètre à la fonction `show_freq_comparison`.

Grâce à l'affichage de la fonction `show_freq_comparison`, on est capable de proposer des correspondances pour les lettres les plus courantes. Ainsi, avec l'affichage précédent, on est quasiment assuré que la lettre 'e' est chiffrée en 'n'. On en déduit l'équation :

$$e_{(a,b)}(\text{'e'}) = \text{'n'} \iff 4a + b = 13 \mod 26$$

À partir de là, deux méthodes sont possibles :

1. On résout l'équation diophantienne $4a + b = 13$ dans $\mathbb{Z}$ et on en déduit les solutions possibles dans $\mathbb{Z}_{26}$. On teste les clés admissibles (i.e pour lesquelles $a \in \mathbb{Z}_{26}^{\times}$) jusqu'à trouver la bonne. On aura de l'ordre de $\mathcal{O}(|\mathbb{Z}_{26}^{\times}|)$ essais à effectuer.
   Appliquer ce raisonnement pour trouver la clé K utilisée.
2. On peut également extraire une seconde relation pour obtenir un système d'équations à deux inconnues (a et b) qu'on résout. Par exemple, en prenant la seconde correspondance, on obtient le système suivant :

$$\begin{cases} e_{(a,b)}(\text{'e'}) = \text{'r'} \\ e_{(a,b)}(\text{'s'}) = \text{'i'} \end{cases} \iff \begin{cases} 4a + b = 13 \mod 26 \\ 18a + b = 8 \mod 26 \end{cases}$$

   Résoudre ce système. Montrer que les clés obtenues ne sont pas les bonnes. En cas d'echec, on peut tester la correspondance suivante (toujours en conservant la première hypothèse, à savoir que $e_{(a,b)}(\text{'e'}) = \text{'r'}$) par exemple : $e_{(a,b)}(\text{'s'}) = \text{'a'}$ etc... En fait, il faudra attendre la correspondance $e_{(a,b)}(\text{'s'}) = \text{'l'}$ pour trouver la bonne solution.

Pour ceux qui ont le temps : implémenter les fonctions permettant d'automatiser la résolution de l'analyse fréquencielle (par l'une ou l'autre des méthodes, ou mieux les deux).
Dans tous les cas et pour vous aider, voici la sortie d'un programme de test illustrant l'utilisation des fonctions demandées et automatisant la résolution par les deux méthodes :

```
Chiffrement affine par clé K=(a,b)=(11,21)      : texte_clair.txt -> Out_texte_chiffre.txt
  Les 50 premiers caractères du chiffrement:  rtxxn cvil mn rvl cn m'thsnawhan c'hi yfrufna, mv
Déchiffrement affine par clé K=(a^-1,b)=(19,21) : Out_texte_chiffre.txt -> Out_texte_dechiffre.txt
  Les 50 premiers caractères du déchiffrement:  comme dans le cas de l'ouverture d'un fichier, la
*** Analyse de fréquence du fichier Out_texte_chiffre.txt ***
-------------------+-------------------
   Texte analysé   |  Reference (fr)
-------------------+-------------------
  n(13) :  16.50%  |  e( 4) :  17.69%
  i( 8) :  10.41%  |  s(18) :  8.87%
  a( 0) :   8.38%  |  a( 0) :  8.11%
  f( 5) :   8.12%  |  n(13) :  7.68%
  t(19) :   7.61%  |  t(19) :  7.44%
  w(22) :   7.36%  |  i( 8) :  7.24%
  l(11) :   6.60%  |  r(17) :  6.43%
                [...]
****** Tentative de déchiffrement *******
Méthode 1 : Utilisation de l'equation obtenue par l'apparition de la lettre la plus fréquente ('e'):
On suppose: e_(a,b)('e')='n' : Résolution de l'équation diophantienne 4a + b = 13 [26]
   Résolution de l'équation homogène dans Z : 4a + b = 0 (E_h)
   -> S_h = { (-k, 4k), k in Z }
   Recherche d'une solution particulière ds Z (a0,b0)/ 4a0 + b0 = 13
   -> via Bezout: (a0,b0) = (0,13)
   Résolution finale (ds Z):
         4a +  b = 13
         4*0 + 13 = 13
        ---------------------
     4(a-0) + (b-13) = 0  (equation homogène résolue)
     => exists k in Z / (a,b) = (0 - k, 13 + 4k)
   Résolution finale ds Z/26Z :
     -> valeurs de k admissible:  0 <= a < 26  soit 0 <= 0-k< 26 soit -25 <= k <= 0
         (k=-25): K = (a,b) = (25,17) -> ayuue pwjg fe awg pe f'ykzervkre p'kj tmaxmer, fw
         (k=-23): K = (a,b) = (23,25) -> ucsse zkxw ne ukw ze n'cglerbgre z'gx jyutyer, nk
         (k=-21): K = (a,b) = (21, 7) -> yicce bsfu ze ysu be z'iaderxare b'af hqynqer, zs
         (k=-19): K = (a,b) = (19,15) -> wskke nobi te woi ne t'sqherzqre n'qb vuwduer, to
         (k=-17): K = (a,b) = (17,23) -> smaae lgtk he sgk le h'mwperdwre l'wt xcsjcer, hg
         (k=-15): K = (a,b) = (15, 5) -> guwwe fivq xe giq fe x'uonerpore f'ov dagbaer, xi
         (k=-11): K = (a,b) = (11,21) -> comme dans le cas de l'ouverture d'un fichier, la
         (k= -9): K = (a,b) = ( 9, 3) -> qwiie xcpy be qcy xe b'wmterfmre x'mp lgqzger, bc
         (k= -7): K = (a,b) = ( 7,11) -> mqyye vuha pe mua ve p'qsberjsre v'sh nomfoer, pu
         (k= -5): K = (a,b) = ( 5,19) -> kagge hqdo je kqo he j'aiferlire h'id bskvser, jq
         (k= -3): K = (a,b) = ( 3, 1) -> ogqqe jylm ve oym je v'gcxerhcre j'cl zkopker, vy
         (k= -1): K = (a,b) = ( 1, 9) -> ikooe tmzc de imc te d'kyjernyre t'yz pwilwer, dm
Méthode 2: Systemes d'equations probables à examiner:
     +--                  +--                              +--
 [1] | e_(a,b)(e)=n  soit |  4a + b = 13 [26] (L1) soit  |  4a +  b = 13 [26] (L1)
     | e_(a,b)(s)=i       | 18a + b =  8 [26] (L2)       |      12b =  6 [26] (4L2-18L1)
     +--                  +--                              +--
     -> Résolution de 12b = 6 [26] : b in {7, 20} (6^-1 mod 13 = 11)
        b = 7 : (L1): 4a = 6 [26]
             a in {21, 8} (2^-1 mod 13 = 7)
             -> K = (a,b) in { (21,7) , (8,7) }
           K = (21, 7) ->yicce bsfu ze ysu be z'iaderxare b'af hqynqer, zs
           K = ( 8, 7) ->impossible: pgcd(8,26)=2
        b = 20 : (L1): 4a = 19 [26] : Pas de solution trouvée
     +--                  +--                              +--
 [2] | e_(a,b)(e)=n  soit |  4a + b = 13 [26] (L1) soit  |  4a +  b = 13 [26] (L1)
     | e_(a,b)(s)=a       | 18a + b =  0 [26] (L2)       |      12b =  0 [26] (4L2-18L1)
     +--                  +--                              +--
```

```
    -> Résolution de 12b = 0 [26] : b in {0, 13} (6^-1 mod 13 = 11)
       b = 0 : (L1): 4a = 13 [26] : Pas de solution trouvée
       b = 13 : (L1): 4a = 0 [26]
            a in {0, 13} (2^-1 mod 13 = 7)
            -> K = (a,b) in { (0,13) , (13,13) }
          K = ( 0,13) ->impossible: pgcd(0,26)=26
          K = (13,13) ->impossible: pgcd(13,26)=13
    +--                  +--                              +--
[3] | e_(a,b)(e)=n  soit |  4a + b = 13 [26] (L1) soit  |  4a +  b = 13 [26] (L1)
    | e_(a,b)(s)=f       | 18a + b =  5 [26] (L2)       |      12b = 20 [26] (4L2-18L1)
    +--                  +--                              +--
    -> Résolution de 12b = 20 [26] : b in {6, 19} (6^-1 mod 13 = 11)
       b = 6 : (L1): 4a = 7 [26] : Pas de solution trouvée
       b = 19 : (L1): 4a = 20 [26]
            a in {18, 5} (2^-1 mod 13 = 7)
            -> K = (a,b) in { (18,19) , (5,19) }
          K = (18,19) ->impossible: pgcd(18,26)=2
          K = ( 5,19) ->kagge hqdo je kqo he j'aiferlire h'id bskvser, jq
    +--                  +--                              +--
[4] | e_(a,b)(e)=n  soit |  4a + b = 13 [26] (L1) soit  |  4a +  b = 13 [26] (L1)
    | e_(a,b)(s)=t       | 18a + b = 19 [26] (L2)       |      12b = 24 [26] (4L2-18L1)
    +--                  +--                              +--
    -> Résolution de 12b = 24 [26] : b in {2, 15} (6^-1 mod 13 = 11)
       b = 2 : (L1): 4a = 11 [26] : Pas de solution trouvée
       b = 15 : (L1): 4a = 24 [26]
            a in {6, 19} (2^-1 mod 13 = 7)
            -> K = (a,b) in { (6,15) , (19,15) }
          K = ( 6,15) ->impossible: pgcd(6,26)=2
          K = (19,15) ->wskke nobi te woi ne t'sqherzqre n'qb vuwduer, to
    +--                  +--                              +--
[5] | e_(a,b)(e)=n  soit |  4a + b = 13 [26] (L1) soit  |  4a +  b = 13 [26] (L1)
    | e_(a,b)(s)=w       | 18a + b = 22 [26] (L2)       |      12b = 10 [26] (4L2-18L1)
    +--                  +--                              +--
    -> Résolution de 12b = 10 [26] : b in {3, 16} (6^-1 mod 13 = 11)
       b = 3 : (L1): 4a = 10 [26]
            a in {9, 22} (2^-1 mod 13 = 7)
            -> K = (a,b) in { (9,3) , (22,3) }
          K = ( 9, 3) ->qwiie xcpy be qcy xe b'wmterfmre x'mp lgqzger, bc
          K = (22, 3) ->impossible: pgcd(22,26)=2
       b = 16 : (L1): 4a = 23 [26] : Pas de solution trouvée
    +--                  +--                              +--
[6] | e_(a,b)(e)=n  soit |  4a + b = 13 [26] (L1) soit  |  4a +  b = 13 [26] (L1)
    | e_(a,b)(s)=l       | 18a + b = 11 [26] (L2)       |      12b = 18 [26] (4L2-18L1)
    +--                  +--                              +--
    -> Résolution de 12b = 18 [26] : b in {21, 8} (6^-1 mod 13 = 11)
       b = 21 : (L1): 4a = 18 [26]
            a in {11, 24} (2^-1 mod 13 = 7)
            -> K = (a,b) in { (11,21) , (24,21) }
          K = (11,21) ->comme dans le cas de l'ouverture d'un fichier, la
          K = (24,21) ->impossible: pgcd(24,26)=2
       b = 8 : (L1): 4a = 5 [26] : Pas de solution trouvée
```

Dans les deux cas, on a réussi à retrouver la clé $K = (11, 21)$.

Références

[1] P. Bouvry, J.-G. Dumas, R.Gillard, J.-L. Roch, and S. Varrette. *Sécurité Multimédia : Cryptographie et Sécurité Systèmes et Réseaux*, volume 2, chapter Cryptographie à Clef Secrète, pages 23–99. Hermès, Fev 2006.

[2] Sébastien Varrette and Nicolas Bernard. Programmation Avancée en C. Technical report, Polycopié de cours UJF/INPG/UL, 2005. `http://www-id.imag.fr/~svarrett/download/polys/poly_cours_C.pdf`.