SERBIN JULIETTE numéro d'étudiant :2564018

NOUKTI SANAE numéro d'étudiant : 2500271

PROJET LM206 : LA CRYPTOGRAPHIE

INTRODUCTION :

Depuis la révolution Internet, les échanges d'informations sont grandement facilités. Reste qu'avec ce flux d'informations permanent, on peine à trouver un espace de confidentialité. Ainsi la cryptographie, science déjà très ancienne, trouve aujourd'hui des applications très nombreuses dans des domaines variés (paiements sécurisés, courrier électronique confidentiel, signatures électroniques..).

Parallèlement à cet intérêt pour un phénomène d'actualité, c'est surtout notre passion pour les Mathématiques qui nous a conduit vers le choix de ce sujet. A travers l'explication des différents procédés de cryptographie il nous sera possible d'aborder, de manipuler différentes notions mathématiques vues au cours de ce semestre.

Ce qui nous a réellement motivé dans le choix de ce sujet, c'est bien sûr la place importante qu'occupent les mathématiques. C'était pour nous une occasion de découvrir comment les mathématiques peuvent s'appliquer dans la vie de tous les jours.

I LE PRINCIPE DE VIGENERE.

a. Présentation.

Le chiffre de Vigenère a été crée par Blaise de Vigenère au XVIe siècle. On parle de système de chiffrement poly-alphabétique. Il utilise une clé qui est un mot ou une phrase. (Plus la clé est complexe, plus le chiffrement est sûr.) Pour chiffrer un texte, on substitue une lettre à une autre grâce à la clé et à la table de Vigenère.

Table de Vigenère.

	Lettre en clair																									
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
Clé Utilisée **A**	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Lettre Codée

On sélectionne dans le tableau la colonne correspondante à la lettre qu'on veut coder et pour une lettre de la clé, on sélectionne la ligne adéquate au croisement ligne/colonne, on obtient la lettre codée.

Exemple : texte à coder : J'adore les fleurs.

Clé : chat

Phrase : *J'ADORE LES FLEURS*

Clé répétée : *C HATCH ATC HATCHA*

Phrase codée : *L HDHTL LXU MLXWYS*

Si on veut déchiffrer ce texte, on regarde pour chaque lettre de la clé répétée la ligne correspondante et on y cherche la lettre codée. La première lettre de la colonne que l'on trouve ainsi est la lettre décodée.

Principe mathématique :

On suppose ici que les mots sont déjà sous forme de suites de chiffres : à chaque lettre on fait correspondre un chiffre compris entre 1 et 26.

→ Chiffrement : Codé = (Texte + Clé) modulo 26

→ Déchiffrement : Texte = (Codé –Clé) modulo 26

b. Mise en pratique dans Scilab.

On s'est intéressé ici uniquement aux fonctions de chiffrements et déchiffrements, en supposant que les phrases étaient déjà sous forme de suites de chiffres.

→Chiffrement : *function [L]= Vigenère (l, C)*

L=pmodulo(l+C,26)

endfunction _

→Déchiffrement *: function [l]= Déchiffre (L, C)*

l=pmodulo (L-C, 26)

endfunction _

II LE CRYTOSYSTEME RSA.

a. Présentation.

La cryptographie suscite de l'intérêt depuis l'antiquité, compte tenu de la nécessité de pouvoir faire parvenir des messages qui ne puissent pas être déchiffrés par un « intrus ». L'algorithme RSA
a été découvert par Rivest, Shamir et Adelman en 1977. On parle de cryptographie asymétrique car il se base sur l'utilisation de deux clés : une publique, permettant le chiffrement et une secrète permettant le déchiffrement. Il est fondé sur l'existence de fonctions à sens unique, c'est-à-dire qu'il est simple d'appliquer cette fonction à un message, mais extrêmement difficile de retrouver ce message à partir du moment où on l'a transformé. Son efficacité repose sur le fait qu'il est difficile de factoriser un grand entier (ayant disons plus de deux cents chiffres dans son écriture décimale c'est-à-dire dans son écriture en base 10) en produit de facteurs premiers .

Deux personnes qu'on nomme par convention Alice et Bob souhaitent échanger des messages confidentiels. Pour cela Alice crée une paire de clés. Elle envoie sa clé publique à Bob qui peut alors lui envoyer des informations codées. Alice sera alors la seule à pouvoir déchiffrer le message grâce à la clé privée.

Alice procède de la façon suivante pour utiliser ce crypto-système :

- elle choisit deux grands nombres premiers p et q et calcule n=p*q
- elle choisit un entier e premier avec φ (n),l'indicateur d'Euler de n , tel que 1<e<φ (n). La classe de e modulo φ (n) est donc inversible dans **Z/ φ(n)Z**.
- elle détermine l'entier d tel que 1<d<φ (n) et ed≡1 mod.[φ(n)]. La classe de d modulo φ (n) est donc l'inverse de la classe de e dans (**Z/φ(n)Z**)*. Ce calcul peut être effectué en utilisant l'algorithme d'Euclide ;
- elle publie ensuite le couple (e, n) qui est sa clé publique, et elle conserve secret le couple (d, φ (n)), qui est sa clé secrète.

Alice l' utilisatrice dont la clé publique est (e,n) et la clé secrète est (d, φ (n)). On dit que l'algorithme de chiffrement de Alice est l'application E : **Z/nZ** → **Z/nZ** définie pour tout x ∈ **Z/nZ** par :

$$\mathbf{E(x) \equiv x^e \ \ mod.n}$$

Et que son algorithme de déchiffrement est l'application D : Z/nZ→Z/nZ define par :

$$\mathbf{D(x) \equiv x^d \ \ mod .n}$$

Supposons qu'un autre utilisateur, Bob, souhaite envoyer un message secret à Alice sous la forme d'un élément $x_0 \in \mathbf{Z/nZ}$. Il utilise cet algorithme de chiffrement de Alice en lui envoyant l'élément $E(x_0)$. Afin de déchiffrer ce message, Alice détermine l'image $D(E(x_0))$ de $E(x_0)$ par D, qui n'est autre que x_0.

b. Mise en pratique dans Scilab.

L'écriture de six fonctions sur le logiciel Scilab nous a permis de mettre en application cet algorithme, les quatre premières permettent de calculer les clés et les deux dernières nous permettent le chiffrement et le déchiffrement d'un message.

- vérification q u'un nombre est premier

```
function [R] = NP(p)
   R= 1
  for i=2:sqrt(p)
      R=R*(pmodulo(p,i)<>0)
     i=i+1
 end
endfunction
```

Cette fonction vérifie si p est un nombre premier en regardant s'il n'est divisible par aucun des nombres compris entre 2 et sa racine.
Une version plus efficace consiste à sortir de la boucle dès qu'on trouve un diviseur de p :

```
function [R] = NPbis(p)
    R= 1
   for i=2:sqrt(p)
       If pmodulo(p,i)<>0 then
       R=R*(pmodulo(p,i)<>0)
        i=i+1
     else
       R=0
      break
     end
  end
endfunction
```

- calcul de l'indicateur d'Euler

```
function  [S] = Phi (p,q)
  if ( and ( [ NP (p) ==1,  NP (q)  ==1 ] ) ) then
    S = ( p - 1 ) * ( q – 1)
  else
    S = ' p et q doivent être premiers'
  end
endfunction
```

- choix de e

```
function  [E] = choixCorrect (e, p, q)
   E = and ( [gcd ( int32 ( [ e, Phi ( p, q) ] ) )  ==1 , and ( [ 1<e, e< Phi ( p, q) ] ) ] )
Endfunction
```

L'écriture de la fonction calculant la valeur de d fait appel à l'algorithme d'Euclide nous permettant de retrouver les coefficients de Bézout :

Soit φ(n) un naturel, e un naturel premier avec φ(n) tel que e<φ(n). On cherche le naturel d tel que $ed \equiv 1 \ [\varphi(n)]$ et d<φ(n). (On cherche l'inverse de e dans **Z/φ(n)Z**).

$eu \equiv 1 \ [\text{mod } \varphi(n)]$ <=> il existe v tel que $eu = 1 + \varphi(n)v$ <=> il existe v tel que $eu - \varphi(n)v = 1$

Ceci est une équation de Bézout, avec a et n connus. Il s'agit de retrouver u et v.

Tout le monde connaît l'algorithme d'Euclide qui donne le PGCD de deux nombres. En adaptant un peu cet algorithme, en le prenant à l'envers, on peut faire le cheminement contraire, c'est à dire trouver les coefficients de Bézout u et v. C'est l'algorithme d'Euclide étendu.

Généralisation :

	$r_0 = \varphi(n)$; $r_1 = e$	
division 1	$r_0 = r_1q_1 + r_2$	(soit $\varphi(n) = eq_1 + r_2$)
division 2	$r_1 = r_2q_2 + r_3$	
...	...	
division i+1	$r_i = r_{i+1}q_{i+1} + r_{i+2}$	relation de récurrence R
...	...	
division n-1	$r_{n-2} = r_{n-1}q_{n-1} + r_n$	
division n	$r_{n-1} = r_nq_n + r_{n+1}$	avec $r_{n+1} = 0$

Mise en application :

Algorithme	Commentaires
(u,u') ← (1,0) ; (v,v') ← (0,1) ; (r,r') ← (φ (n),e)	Initialisation des variables, u joue le rôle de u_i ; u" de u_{i+1}, et on calculera u_{i+2} (même notations pour v et r)
Tant que(r' > 0) Faire	Boucle, but : "descendre d'un cran" :
q ← Partie entière(r/r') (u",v",r") ← (u,v,r) (u,v,r) ← (u',v',r') (u',v',r') ← (u"-qu' , v" - qv', r - qr')	(u, v, r) prend la valeur de (u', v', r') et (u', v', r') est recalculé. (sauvegarde temporaire de u, v, r)
Fin boucle	Les variables u et v contiennent alors les coefficients recherchés, r contient le pgcd de e et φ(n).

Revenons à R.S.A : on veut inverser e modulo φ(n).
On veut un résultat de Z/φ(n)Z donc on il suffit de chercher la classe de résidus modulo de φ(n) de u, autrement dit on effectue la division euclidienne de u par n : $u = u_0 + k\varphi(n)$ ($0<u_0<\varphi(n)$), on a alors :
$ue + v\varphi(n) = 1 \Rightarrow (u_0 + k\varphi(n))e + v\varphi(n) = 1 \Rightarrow u_0e + (ke+v)\varphi(n) = 1 \Rightarrow u_0e \equiv 1 \pmod{\varphi(n)}$
avec $0<u_0<\varphi(n)$.

Pour cela dans Scilab nous avons crée une matrice R prenant la suite des restes de l'algorithme d'Euclide entre e et φ(n), la matrice Q prends la suite des quotients. Les coefficients de Bézout sont trouvés par la création de deux matrices U et V. La valeur de d est le dernier élément de la matrice U.

* calcul de d : algorithme d'Euclide

```
function [Q, R, U, V, d] = euclide (e, p, q)
Q = [ ] ;
R = [Phi (p, q), e] ;
U = [1, 0] ;
V = [0, 1] ;
   while R ($) < > 0
      Q = [Q,   int (R ($-1) / R ($))]
      R = [ R , pmodulo ( R ($-1) , R ($) ) ]
      U = [ U , U ($-2) – U ($-1) * Q ($-1) ]
      V = [V, V ($-2) – V ($-1) * Q ($-1)]
   end
d= U($)
endfunction
```

Ecriture des fonctions de cryptage/ décryptage :

* message chiffré

```
function  [C] = chiffrement (M, e, p, q)
   C = pmodulo ( M**e , p*q )
endfunction
```

* déchiffrement

```
function  [M] = dechiffrement (C, d, p, q)
   M =p modulo (C**d, p*q)
endfunction
```

c. Exemple d'utilisation.

- L'utilisateur choisit deux nombres et vérifie grâce à *NP* s'ils sont premiers p=5 et q=7 (n = 35)
- Il calcule l'indicateur d'Euler φ(n) à l'aide de la fonction *Phi* (φ(n) =24)
- Il choisit un naturel e et vérifie grâce à *choixCorrect* s'il est bien choisit (ie premier avec φ(n)=24 et compris entre 1 et φ (n)), par exemple e=11
  Détermination de d, grâce à la fonction *euclide :*
  $\varphi(n) = eq + r_2$
  24 = 11x2 + 2
  11 = 2x5 + 1
  5 = 1x5 + 0

  On remonte ensuite :
  1 = 11 – (2x5)
  1 = 11 – ((24 – 11x2) x5)
  1 = 11(11) + 24(-5)

  Les matrices de la fonction *euclide* sont donc les suivantes à la sortie de la boucle :
  Q = [2, 5,2,]
  R = [24, 11, 2,1]
  U = [1, 0, 1,-5]
  V = [0, 1,-2,11]

  On a alors d qui est la dernière valeur de V : d=11

Bilan : L'utilisateur a donc pour clé publique le couple (11, 35) et secrète (11, 24) et peut donc ainsi chiffrer et déchiffrer des informations confidentiels à l'aide des fonctions *chiffrement* et *dechiffrement*

III OUTILS ET DIFFICULTES :

Pour écrire les fonctions de l'algorithme RSA dans Scilab, nous avons utilisé de nouveaux outils.

Nous intéresserons plus particulièrement à l'utilisation de *and, gcd, for,* et *while*

- Traduction du « et » logique :

Le « et logique » (noté ^ ou .) prend deux arguments. Si on veut traduire que la condition A et la condition B doivent être vérifiées pour effectuer une opération, on écrit A^B. Dans Scilab, le « et logique » se note *and*. Notre erreur a été de croire que cette fonction fonctionnait dans Scilab aussi simplement et nous avons écrit *and*(A,B). En fait la fonction *and* prend une matrice en argument. Cette matrice est à 1 ligne et 2 colonnes qui correspondent aux 2 arguments. Il nous fallait donc écrire : *and*([A,B]). Dans la fonction *choixCorrect* on utilise 2 and imbriqués pour traduire que e doit être premier avec φ (n) et 1<e<φ (n) : *and ([gcd (int32 ([e, Phi (p, q)])) ==1 , and ([1<e, e< Phi (p, q)])])*

- Calcul du modulo, PGCD :

→Deux fonctions existent déjà dans Scilab pour calculer le reste dans la division euclidienne d'un nombre par un autre. Il s'agit de *modulo* et de *pmodulo*. Elles fonctionnent toutes les deux de la même manière mais renvoient des résultats un peu différents.
On écrit *r = modulo (n, m)* ou *r = pmodulo (n, m)* lorsqu'on veut trouver le reste dans la division euclidienne de n par m. La différence entre ces 2 fonctions vient du fait que le *pmodulo* renvoie un résultat positif, c'est celle-là qui nous intéresse. On l'a utilisé par exemple dans la fonction NP qui vérifie le caractère premier d'un nombre :
R=R(pmodulo(p,i)<>0)*

→Une fonction existe dans Scilab pour calculer le PGCD de deux nombres. Il s'agit de la fonction *gcd*. Là encore, cette fonction prend une matrice pour argument. La matrice argument devra comporter une ligne et n colonnes qui correspondent aux arguments du PGCD. Les arguments étant des polynômes. La fonction *gcd* fonctionne avec des arguments qui doivent être de type 8 c'est-à-dire codés sur 1,2 ou 4 octets. Pour calculer le PGCD d'entiers nous devons donc les rendre de type 8, pour cela il faut utiliser la fonction *int32* qui effectue la conversion au codage entier à 4 octets. Nous avons utilisé gcd dans la fonction *choixCorrect* : *gcd (int32 ([e, Phi (p, q)]).*

- Boucles logiques :

Dans l'écriture de nos fonctions dans Scilab, nous avons utilisé deux types de boucles.

→Les boucles *for* qui permettent de répéter des opérations un certain nombre de fois, par exemple dans la fonction *NP* on veut vérifier si un nombre p est divisible par les entiers compris entre 2 et sqrt(p). La boucle *for* a la sémantique suivante :

for i = deb :fin

```
        instructions
    end
```
Par exemple dans *NP* :
```
    for i=2:sqrt(p)
        R=R*(pmodulo(p,i)<>0)
        i=i+1
    end
```
Notre erreur a été d'oublier l'instruction i=i+1 ce qui fait que la boucle ne pouvait pas fonctionner puisque le « i » n'avançait pas. Le résultat de NP (p) était donc R=1 (initialisation de la variable R en début de fonction) pour tout entier p mis en argument.

→Les boucles *while* qui permettent d'effectuer une opération tant qu'une condition est réalisée. Leur sémantique est la suivante :
```
    while condition
            instructions
    end
```
Nous avons utilisé une telle boucle dans la fonction *euclide :*
```
    while R ($) < > 0
      Q = [Q,   int (R ($-1) / R ($))]
      R = [ R , pmodulo ( R ($-1) , R ($) ) ]
     U = [ U , U ($-2) – U ($-1) * Q ($-1) ]
     V = [V, V ($-2) – V ($-1) * Q ($-1)]
    end
```

- Notre principale difficulté a été d'écrire la fonction de calcul de d. La fonction écrite ne fonctionne pas et nous renvoie comme résultat de l'appel de la fonction *euclide(e,p,q)* = [1,0] qui est l'initialisation de la matrice U qui sert à calculer un coefficient de Bézout. Nous n'avons malheureusement pas réussi à trouver d'où provenait notre erreur.

CONCLUSION :

Ce projet nous a permis essentiellement de mettre en pratique des notions d'analyse numériques et de découvrir quelques fonctions importante de l'environnement Scilab. Nous nous sommes appuyées principalement sur deux types de crypto-systèmes qui font appel à des algorithmes de chiffrement faible (asymétrique pour RSA), ce qui nous a permis d'enrichir notre connaissance de Scilab. L'outil Scilab nous a semblé adapté à notre projet , cependant, l'utilisation de matrices ou vecteurs lignes pour la plupart des fonctions complique leur utilisation et nous pensons que coder des algorithmes plus complexe aurait été plus fastidieux.