

Cryptographie: études des divers chiffrements

Quels critères de sécurité informatique devons nous respecter afin de sécuriser nos données et nos informations personnelles?

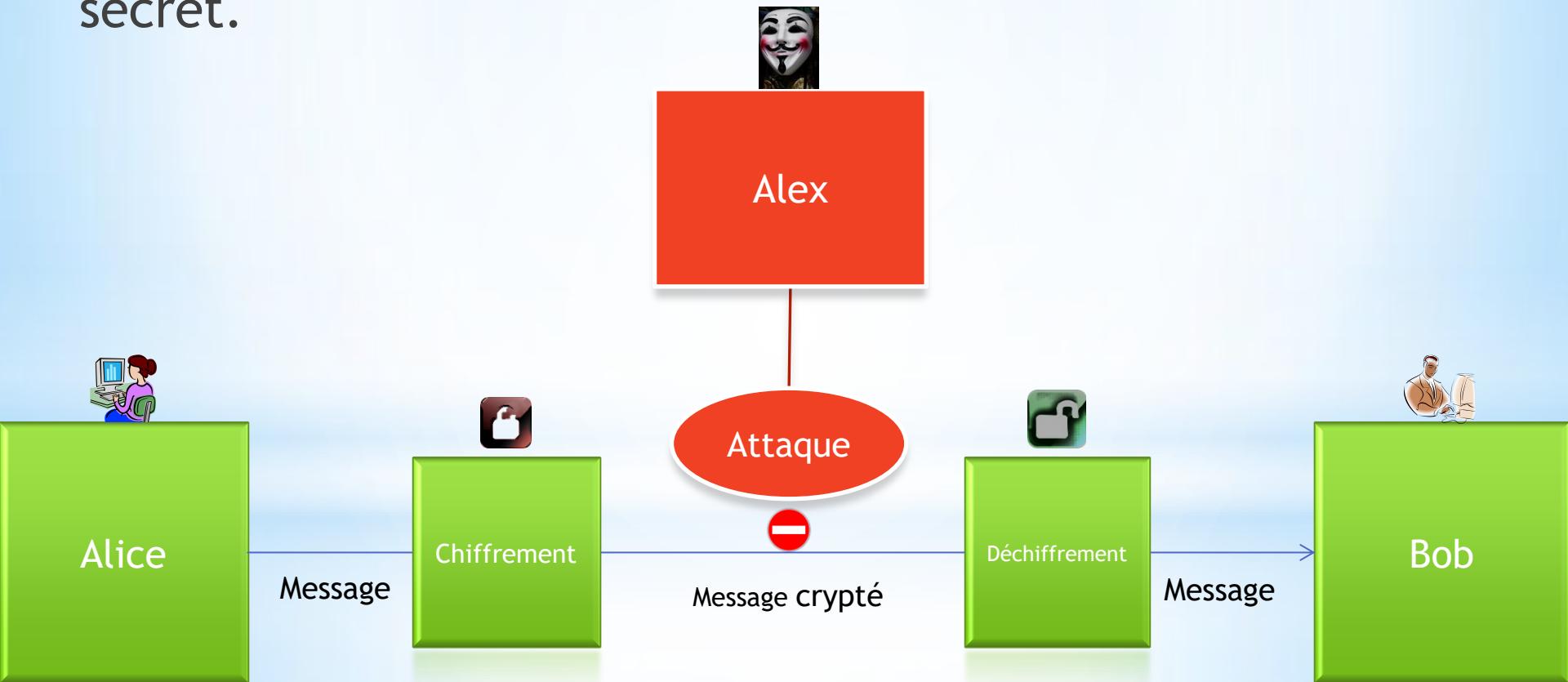
Plan:

- 1)Contexte
- 2)Différents type de chiffrements:
 - a)César
 - b)Vigenère
 - c)Chiffrement par substitution
 - d)Hill
 - e)RSA
- 3)Cryptanalyse et comparaison



1) Contexte

Supposons qu'Alice souhaite communiquer un secret avec son ami Bob, mais Alex a pour but de savoir ce secret.



On définit un ordre alphabétique et on associe à chaque lettre un ordre:

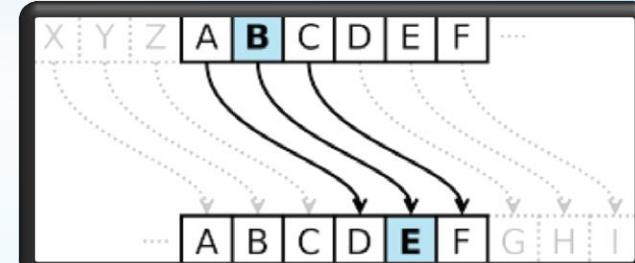
Alphabet	Ordre
A	0
B	1
C	2
D	3
E	4
F	5
G	6
H	7
I	8
J	9
K	10
L	11
M	12

Alphabet	Ordre
N	13
O	14
P	15
Q	16
R	17
S	18
T	19
U	20
V	21
W	22
X	23
Y	24
Z	25

1) Différents types de chiffrements:

a) Chiffrement de César:

- C'est un chiffrement très classique
- Il s'agit de translater l'ordre alphabétique avec un entier k $k \in \mathbb{Z}/26\mathbb{Z}$.
- 25 chiffrements possibles.
 - +) Pour que le message soit décrypté par bob il doit recevoir k (l'entier de décalage) .
 - +) Si on suppose qu'Alex connaît il s'agit d'un chiffrement de César il lui suffit de prendre un seul mot du message chiffré et de trouver l'entier de translation et l'appliquer pour tout le message.



b)Chiffrement de Vigenère(1/3):

- Il s'agit d'un chiffrement par blocs.
- La clé est une liste de caractères.
- Addition de la clé et du message.

Exemple:

Alice veut communiquer son mot de passe avec Bob qui est le suivant '**j adore ecouter la radio toute la journee**'.

Elle choisit comme clé
['M', 'U', 'S', 'I', 'Q', 'U', 'E']

Son message se crypte de la manière suivante:
On fait une opération élémentaire au message.



Chiffrement de Vigenère(2/3)

J ADORE ECOUTER LA RADIO TOUTE LA JOURNEE

$$+ \quad \underline{M \ USIQU \ EMUSIQU \ EM \ USIQU \ EMUSI \ QU \ EMUSIQU}$$

$$= \quad V \ UVWHY \ IOIMBUL \ PM \ LSLYI \ XAOLM \ BU \ NAOJVUY$$

+) Pour Bob il est suffisant de calculer l'inverse de chaque lettre qui est déterminé par la relation suivante:

$$\text{Soit } \alpha \text{ un alphabet} \quad \text{ordre}(\alpha^{-1}) = 26 - \text{ordre}(\alpha)$$

D'où Bob calcule l'inverse de ['M', 'U', 'S', 'I', 'Q', 'U', 'E']

qui est ['O', 'G', 'I', 'S', 'K', 'G', 'W'] .

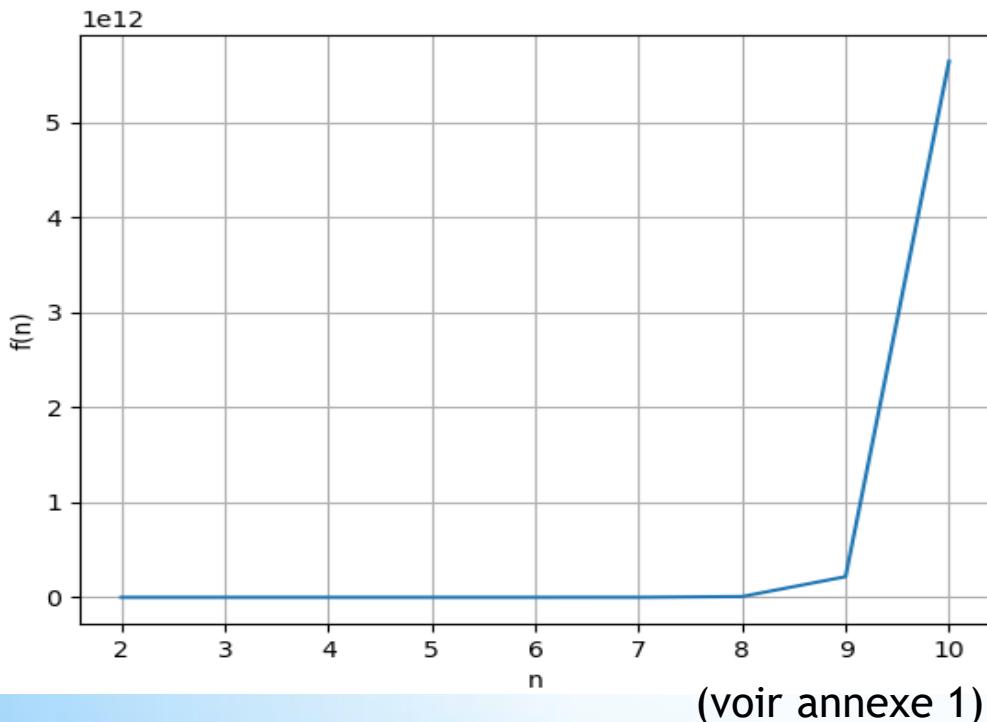
Il reste à Bob de refaire l'opération qu'Alice a fait mais avec le message crypté et l'inverse calculé.

Chiffrement de Vigenère(3/3)

+) Pour Alex, il est devant deux problèmes: le 1er est la longueur de la clé et le deuxième son contenu

Si on suppose que n est la longueur du message le nombre de clés possibles est

$$f(n) = \sum_{k=1}^n 26^k = \frac{1-26^n}{1-26} \text{ (voir annexe 5)}$$



Pour $n > 9$ il y a au moins $2 * 10^{11}$ clés possibles ce qui rend l'attaque une tâche plus difficile.

c) Chiffrement par substitution

Il s'agit d'un chiffrement qui redéfinit l'ordre alphabétique c'est-à-dire chaque alphabet a un nouvel ordre.

Exemple:

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

Pour que Alice transmet son mot de passe ‘j adore ecouter la radio toute la journee’, elle le crypte pour devenir ‘F WVLZB BCLTRBZ HW ZWVDL RLTRB HW FLTZKBB’.

- +) Pour Bob il doit refaire le même travail d'Alice.
- +) Pour Alex il est face à $26! = 4 * 10^{26}$ clés possibles.

e) Chiffrement de Hill(1/2):

- Il s'agit d'un chiffrement qui utilise les notions d'algèbre linéaire.
- La clé est une matrice inversible à coefficients dans $\mathbb{Z}/26\mathbb{Z}$.
- On dit qu'une matrice M à coefficients dans $\mathbb{Z}/26\mathbb{Z}$ est inversible dans $\mathbb{Z}/26\mathbb{Z}$ si $\text{pgcd}(\det M, 26)=1$.
- Le chiffrement de Hill s'effectue de la manière suivante:
Si Alice veut communiquer le mot 'july' avec Bob elle choisit sa clé

$$\begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix} \quad (\text{pgcd}(77-24, 26)=1)$$

July se décompose en $ju=(9,20)$ et $ly=(11,24)$



Wikipédia

Hill(2/2)

Ainsi ‘july’ deviendrait ‘DELW’

$$(9, 20) \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix} = (99 + 60, 72 + 140) = (3, 4)$$

$$(11, 24) \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix} = (121 + 72, 88 + 168) = (11, 22).$$

- +) Pour bob il suffit de calculer l'inverse de la clé qui est $\begin{pmatrix} 7 & 18 \\ 23 & 11 \end{pmatrix}$ et refaire le travail d'Alice avec le mot crypté et l'inverse de la clé.
- +) Par contre Alex se trouve devant 26^{m^2} (sans vérification de l'inversibilité de la matrice) clés possibles (m désigne la taille de la matrice.
(Pour m=4, 26^{4^2} est de l'ordre 10^{22})

e) Chiffrement RSA(1/4)

- Il s'agit d'un chiffrement symétrique qui possède des clés privées et des clés publiques.
- Le RSA se base sur les notions de l'arithmétique surtout le théorème de Bézout et Fermat .
- Chiffrement qui est utilisé très fréquemment dans le monde

Mise en évidence:

Supposons qu'Alice veut communiquer le nombre m avec Bob.

Bob choisit tout d'abord p et q deux entiers premiers et calcule $n=p*q$.

On note φ l'indicatrice d'Euler défini par



$$\varphi(n) = \text{card}(\{k \in \mathbb{N} \text{ tel que } k \wedge n = 1\})$$

RSA(2/4)

Une propriété intéressante de l'indicatrice d'Euler est:

$$\forall p, q \text{ premiers} \quad \varphi(p * q) = \varphi(p) * \varphi(q) = (p - 1) * (q - 1)$$

Bob donc calcule $\varphi(n) = (p-1)*(q-1)$ et puis choisit e tel que $e \wedge \varphi(n) = 1$

$\varphi(n)$ demeure une clé secrète et n et e des clés publiques

Puisque $e \wedge \varphi(n) = 1$ alors $\exists d \in \mathbb{Z}$ tel que $d * e = 1 \pmod{\varphi(n)}$

Une fois d calculé Bob détruit p , q et $\varphi(n)$.

Alice veut envoyer un message à Bob elle se débrouille pour qu'il soit entier $m < n$.

Elle le crypte suivant cette formule $x \equiv m^e \pmod{n}$.

Bob reçoit donc x .



+) Bob calcule $x^d \pmod{n} \equiv m^{e*d} \pmod{n} \equiv m^{1+k*\varphi(n)} \pmod{n}$

★ : $d * e = 1 \pmod{\varphi(n)}$ $\Rightarrow \exists k \in \mathbb{Z}$ tel que $d * e = 1 + k * \varphi(n)$

RSA(3/4)

Il se trouve devant deux cas:

- $\text{Pgcd}(m, n)=1$

- $\text{pgcd}(m, n)\neq 1 \Rightarrow \text{pgcd}(m, n)=p$ ou $\text{pgcd}(m, n)=q$ (le travail est le même par symétrie)

1^{er} cas : $\text{pgcd}(m, n)=1$

D'après le théorème d'Euler

$$m^{\varphi(n)} \equiv 1 \pmod{n}$$

Ainsi $m^{k*\varphi(n)} \equiv 1 \pmod{n} \Rightarrow m^{k*\varphi(n)+1} \equiv m \pmod{n}$

I.e. $x^d \pmod{n} \equiv m \pmod{n}$.

2^{ème} cas: $\text{pgcd}(m, n)\neq 1$

Supposons $\text{pgcd}(m, n)=p$

d'où $m \equiv 0 \pmod{p} \Rightarrow m^{d*e} \equiv 0 \pmod{p} \Rightarrow m^{d*e} \equiv m \pmod{p}$

$m^{d*e} = m * m^{\varphi(n)*k} = m * (m^{\varphi(q)})^{(p-1)*k} \equiv m \pmod{q}$ ($\text{pgcd}(m, q)=1$ sinon $m=n$)

Or p et q sont premiers entre eux alors:

$$x^d \pmod{n} \equiv m \pmod{n}$$

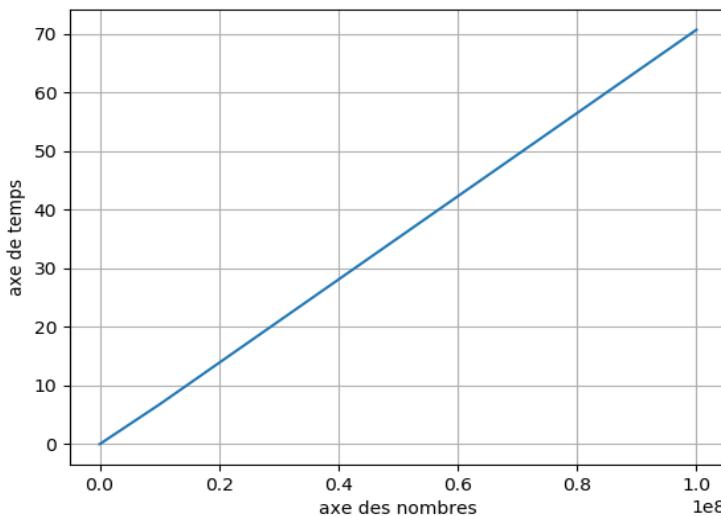
RSA(4/4)

+) Pour Alex le décryptage de ce chiffrement nécessite le nombre d puisque n et e sont publiques

Or $d * e = 1 \pmod{\varphi(n)}$ donc il suffit de calculer $\varphi(n)$

C'est-à-dire trouver p et q.

Prouvons pour p et q assez grands, trouver ces deux entiers nécessite beaucoup de temps.

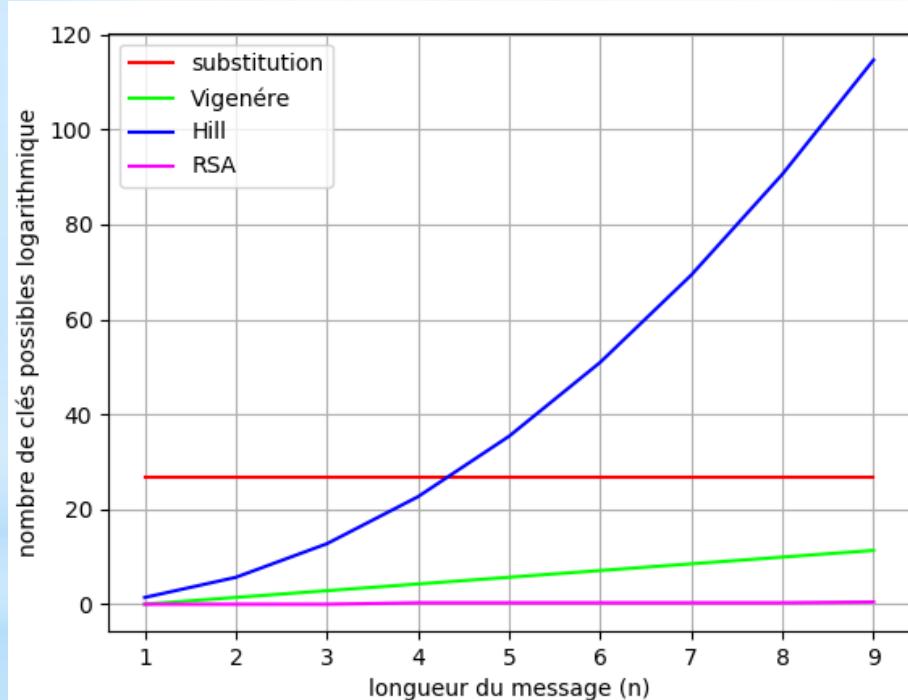


Pour trouver les facteurs de 99999989 il faut 70 secondes pour trouver n.

Pente: $70 * 10^{-8} s * n^{-1}$

2) Cryptanalyse et comparaison:

Etudions, théoriquement, le nombres de clés possibles en fonction de la longueur du message.



on remarque que le chiffrement de Hill est très efficace pour crypter les informations
La cryptanalyse montre le contraire.

$$y = \log\left(\frac{1-26^n}{1-26}\right)$$

$$y = \log(4 * 10^{26})$$

$$y = \log(26^{n^2})$$

$$y = \log(\sqrt{n})$$

(Voir annexe 3)

Cryptanalyse: Hill

Si on suppose qu'Alex a pu trouver m et le message avant son cryptage il peut s'en débrouiller pour trouver la clé.

Notons $X=(x_{i,j})_{1 \leq i,j \leq m}$ la matrice du message, $K=(k_{i,j})_{1 \leq i,j \leq m}$ la matrice clé et $Y=(y_{i,j})_{1 \leq i,j \leq m}$ la matrice du message crypté.

On a $Y=X*K$ si X est inversible alors $K=X^{-1} * Y$

Sinon il faut essayer un autre ensemble de m paires de tuples. (on admet l'existence d'une permutation de paire pour que X soit inversible)

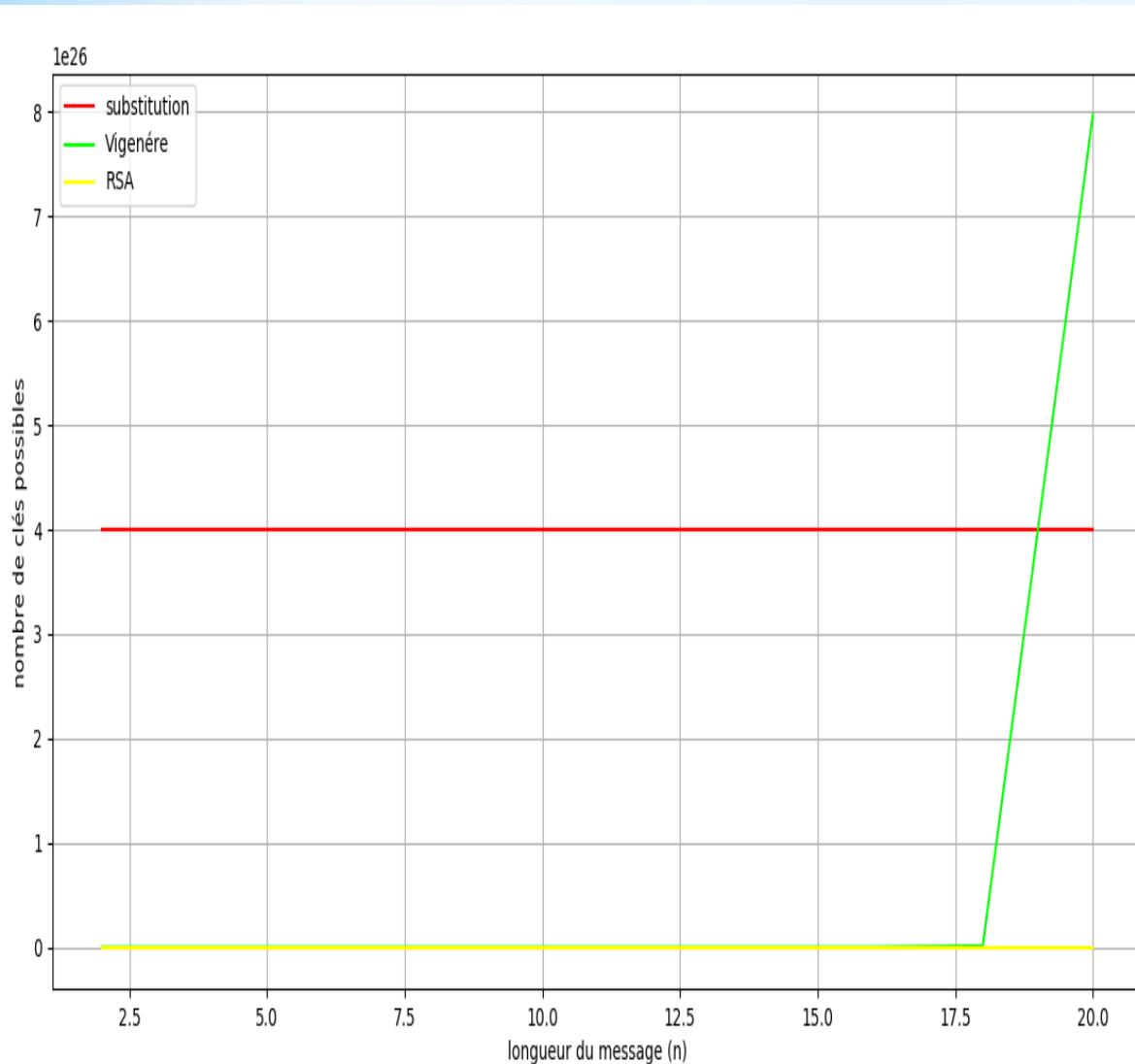
Exemple:

Supposons que le message est 'FRIDAY' qui va être chiffré en utilisant le chiffrement de Hill avec $m=2$ en 'PQCFKU'

$$\begin{pmatrix} 15 & 16 \\ 2 & 5 \\ 10 & 10 \end{pmatrix} = \begin{pmatrix} 5 & 17 \\ 8 & 3 \\ 0 & 24 \end{pmatrix} K$$

$$\left(\begin{array}{cc} 5 & 17 \\ 8 & 3 \end{array} \right)^{-1} = \left(\begin{array}{cc} 9 & 1 \\ 2 & 15 \end{array} \right)$$
$$K = \left(\begin{array}{cc} 9 & 1 \\ 2 & 15 \end{array} \right) \left(\begin{array}{cc} 15 & 16 \\ 2 & 5 \end{array} \right) = \left(\begin{array}{cc} 7 & 19 \\ 8 & 3 \end{array} \right)$$

Si on élimine le chiffrement de Hill, la courbe devient donc:



Sans Hill, Vigenère semble être un chiffrement difficile à crypter . De même, il est vulnérable à cause du test de Kasiski qui renseigne sur la longueur de la clé et ensuite un chiffrement par décalage pour déterminer la clé .

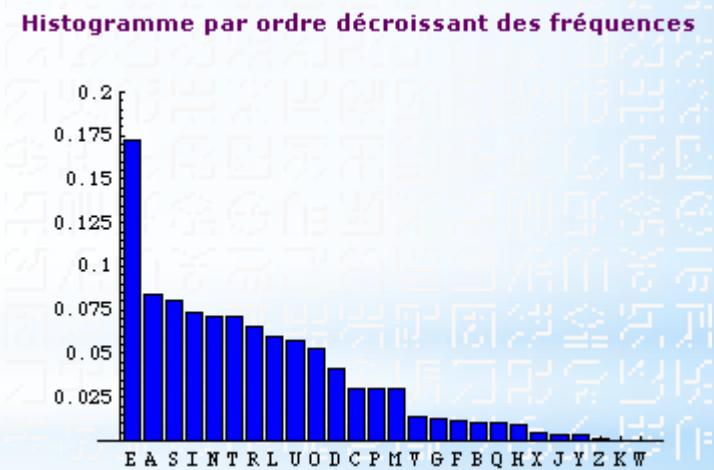
(Voir annexe 4)

Cryptanalyse: substitution

Le chiffrement de substitution qui à premier instant semble impossible à décrypter puisqu'ils existent $26! \approx 4 * 10^{26}$ clés possibles, peut être décrypté facilement à l'aide d'une analyse fréquentielle des caractères alphabétiques.

Un des grands défauts d'un chiffrement pareil est que si on augmente longueur du message il devient plus facile à décrypter puisqu'on augmente la précision de l'analyse fréquentiel.

Pour la langue française:



Source:

<https://www.apprendre-en-ligne.net/crypto/stat/francais.html>

Le RSA reste une forme de cryptage très sécurisé tant que les ordinateurs quantiques ne peuvent pas encore factoriser des grands entiers en utilisant l'algorithme de Shor et les nombres choisis sont assez grand.

Exemple:

Pour un nombre d'ordre 10^{15} il faut plus qu'une année pour le décrypter à l'aide d'un ordinateur classique .

Conclusion:

On ne peut jamais parvenir à un chiffrement parfait .

La différence entre tous ces chiffrements est la durée pour laquelle on peut cacher une information d'un attaquant.

Pour cela il faut changer d'une façon régulière la clé .

Merci pour votre
attention!

Annexe: Algorithmes

```
def graph_s_v():
    n1=[]
    t=[]
    for i in range(1,10):
        n1.append((pow(26,i)-1)/25)
        t.append(i)
    plt.plot(t,n1)
    plt.grid()
    plt.xlabel('n')
    plt.ylabel('f(n)')
    plt.show()
```

```

def attaque_rsa(n):
    i=2
    d=time()
    while n%i!=0:
        i+=1
    return(time()-d)
#fonction qui renseigne sur la durée qu'une machine prend pour trouver un facteur de n

def plusprochepremierinf(n):
    i=0
    while not(ispremier(n-i)) and n-i>0:
        i+=1
    return(n-i)
#fonction qui permet de trouver le plus proche premier inférieur à n

def graphe_rsa():
    k=1
    l=[]
    K=[]
    while k<=10**6:
        k*=10
        K.append(k)
        for i in range(k,k+1):
            l.append(attaque_rsa(plusprochepremierinf(k+1)**2))
    plt.plot(K,l)
    plt.grid()
    plt.xlabel('axe des nombres')
    plt.ylabel('axe de temps')
    plt.show()

```

```
import math as m
def comparaison():
    n=[]
    n1=[]
    n2=[]
    n3=[]
    n4=[]
    i=0
    while i<9:
        i+=1
        n.append(i)
        n1.append(m.log(4*10**26,10))
        n2.append(m.log((pow(26,i)-1)/25,10))
        n3.append(m.log(pow(26,i**2),10))
        n4.append(m.log(int(i**0.5),10))
    plt.plot(n,n1,color=(1.0,0.0,0.0),label='substitution')
    plt.plot(n,n2,color=(0.0,1.0,0.0),label='Vigenére')
    plt.plot(n,n3,color=(0.0,0.0,1.0),label='Hill')
    plt.plot(n,n4,color=(1.0,0.0,1.0),label='RSA')
    plt.grid()
    plt.xlabel('longueur du message (n)')
    plt.ylabel('nombre de clés possibles logarithmique')
    plt.legend()
    plt.show()
```

```
def comparaisonh():
    n=[]
    n1=[]
    n2=[]
    n4=[]
    i=0
    while i<20:
        i+=2
        n.append(i)
        n1.append(4*10**26)
        n2.append((pow(26,i)-1)/25)
        n4.append(int(i**0.5))
    plt.plot(n,n1,color=(1.0,0.0,0.0),label='substitution')
    plt.plot(n,n2,color=(0.0,1.0,0.0),label='Vigenére')
    plt.plot(n,n4,color=(1.0,1.0,0.0),label='RSA')
    plt.grid()
    plt.xlabel('longueur du message (n)')
    plt.ylabel('nombre de clés possibles')
    plt.legend()
    plt.show()
```

Annexe: démonstrations mathématiques

+) Puisque $[A; Z]$ est isomorphe à $\llbracket 0; 25 \rrbracket$ le nombre de clés possibles pour le chiffrement de Vigenère est:

$$\text{card}(\llbracket 0; 25 \rrbracket \cup \dots \cup \llbracket 0; 25 \rrbracket^n) = \sum_{k=1}^n 26^k = \frac{1-26^n}{1-26}$$

+) Chiffrement de Hill: une matrice M à coefficients dans $\mathbb{Z}/26\mathbb{Z}$ est inversible dans $\mathbb{Z}/26\mathbb{Z}$ si $\text{pgcd}(\det M, 26)=1$

Si on admet que $\forall A \in \mathcal{M}_n(\mathbb{Z}/26\mathbb{Z})$

$$A^{-1} = \det(A)^{-1} * {}^t(\text{com}(A))$$

$\det(A)$ admet un inverse dans $\mathbb{Z}/26\mathbb{Z}$ si $\text{pgcd}(\det M, 26)=1$

```

def concat(l):
    ch=''
        for i in range(len(l)):
            ch+=l[i]
    return(ch)

#-----dechiffrer/chiffrer cesar

def cesar(ch,k):
    #ch: message qu'on souhaite crypter(chaine de caractères)
    #k entier de translation compris entre 1 et 25
    l1=[]
    l=[ord(maj(ch[i]))-65 for i in range(len(ch))]
    for i in range(len(l)):
        if ch[i].isalpha():
            l[i]+=k
            l1.append(chr(l[i]%26+65))
        else:
            l1.append(ch[i])
    return(concat(l1))

#concat: fonction qui permet de concaténer les caractères d'une liste en une chaîne

def d_cesar(ch):
    #ch: message cryptée
    for k in range(26):
        l1=[]
        l=[ord(maj(ch[i]))-65 for i in range(len(ch))]
        for i in range(len(l)):
            if ch[i]!=' ':
                l[i]+=k
                l1.append(chr(l[i]%26+65))
            else:
                l1.append(' ')
        print(concat(l1),k)

```

Cryptage/décryptage

Algorithme d'attaque

```

#####{-----chiffrement/dechiffrement vigenére-----}

def vigenére(ch,k):
    #ch:message qu'on veut crypter
    #k:clé liste
    for i in k:
        assert ord(maj(i))-65>-1 and ord(maj(i))-65<27
    assert len(ch)>=len(k)
    i=0
    l=[]
    j=0
    for i in range(len(ch)):
        if not ch[i].isalpha():
            l.append(ch[i])
            j+=1
        else:
            l.append(k[i%len(k)-j])
    l1=[]
    for i in range(len(ch)):
        if ch[i].isalpha():
            l1.append(chr((ord(maj(ch[i]))-65+ord(maj(l[i]))-65)%26+65))
        else:
            l1.append(ch[i])
    return(concat(l1))

#concat fonction qui permet de concaténer les caractères d'une liste en une chaîne

```

cryptage

Décryptage

```

def d_vigenére(ch,k):
    l=[]
    for i in range(len(k)):
        if maj(k[i])=='A':
            l.append('a')
        else:
            l.append(chr(26-ord(maj(k[i]))+65*2))
    ch=vigenére(ch,k)
    return(vigenére(ch,l))

```

```

def distinct(ch):
    b=True
    i=-1
    while b and i<len(ch)+1:
        i+=1
        for j in range(i+1,len(ch)):
            b=b and ch[i]!=ch[j]
    return(b)
#fonction permet d'assurer que toutes
#les caractères de ch (liste de caractère) ne se répètent pas dans ch

def substitution(ch,key):
    #ch:message à crypter
    #key:liste de caractère
    assert distinct(key) and len(key)==26
    for k in key:
        assert k.isalpha()
    l=[maj(k) for k in key]
    ll=[]
    for i in range(len(ch)):
        if ch[i].isalpha():
            ll.append(l[ord(maj(ch[i]))-65])
        else:
            ll.append(ch[i])
    return(concat(ll))
#concat fonction qui permet de concaténer les caractères d'une liste en une chaîne

```

Cryptage/décryptage

```

def nb_occ(a,ch):
    #a:caractère qu'on veut chercher son nombre d'occurrence
    #ch:message
    nb=0
    for i in range(len(ch)):
        if ch[i]==a:
            nb+=1
    return(nb)
#fonction qui permet de calculer le nombre d'occurrence d'une lettre
def maxi(l):
    #l: liste de liste formé par un caractère et un entier
    m=l[0][1]
    x=l[0][0]
    for i in range(len(l)):
        if l[i][1]>m:
            m=l[i][1]
            x=l[i][0]
    return([x,m])
#permet de trouver le maximum d'une liste de liste
def tri(l):
    l1=[]
    for i in range(len(l)):
        k=maxi(l)
        l1.append(k)
        l.remove(k)
    return(l1)
#fonction pour trier une liste de liste

```

Algorithme d'attaque

```

def occ_tot(ch):
    ch=maj(ch)
    l=[]
    for i in range(26):
        l.append([chr(i+65),nb_occ(chr(i+65),ch)])
    return(tri(l))
#fonction qui renvoie une liste de liste triée donnant
#le nombre d'occurrence de chaque caractère
def permut(ch,f,c):
    assert ch.isalpha() and len(f)*len(c)==1
    ch=maj(ch)
    l=[k for k in ch]
    for i in range(len(l)):
        if l[i]==f:
            l[i]=c
    return(concat(l))
#fonction qui permet de remplacer un caractère par
#un autre dans un message
def stat(ch,t):
    l=occ_tot(ch)
    l1=['a' for i in range(len(ch))]
    for i in range(len(ch)):
        for j in range(26):
            if ch[i]==l[j][0]:
                l1[i]=t[j]
    print(concat(l1))
#algorithme d'attaque
#concat fonction qui permet de concaténer
#les caractères d'une liste en une chaîne

```

```

def Hill(ch,k):
    assert k.shape[0]%2==0
    l=[]
    l1=[]
    ch=maj(ch)
    m=k.shape[0]//2+1
    for i in range(m):
        l.append(ord(ch[i])-65)
        l1.append(ord(ch[i+m])-65)
    a=np.array([l])
    b=np.array([l1])
    l2=np.dot(a,k)
    l3=np.dot(b,k)
    m1,m2,m3,m4=[],[],[],[]
    for i in range(l2.shape[1]):
        l2[0,i]%=26
        l3[0,i]%=26
        m1.append(l2[0,i])
        m2.append(l3[0,i])
    for i in range(l2.shape[1]):
        m3.append(chr(m1[i]+65))
        m4.append(chr(m2[i]+65))
    return(concat(m3+m4))
#concat fonction qui permet de concaténer
#les caractères d'une liste en une chaîne
#Pour déchiffrer il suffit de faire appel à la même fonction mais avec le
#couple crypté et l'inverse de matrice prise en premier instant comme clé

```

Cryptage/décryptage

```

#-----RSA-----
def ispremier(n):
    b=True
    for i in range(2,int(n**1/2)):
        b=b and n%i!=0
    return(b)
def euclide(a,b):
    x,xx,y,yy=1,0,0,1
    while b!=0:
        q=a//b
        a,b=a%b,a
        xx,x=x-q*xx,xx
        yy,y=y-q*yy,yy
    return(a,x,y)
#cette fonction retourne le pgcd de a et b puis les coefficients
#de Bézout c'est-à-dire u,v tels que a*u+b*v=pgcd(a,b)
def inverse(a,n):
    x,u,v=euclide(a,n)
    if x!=1:
        return 0
    else:
        return u%n
#cette fonction retourne l'inverse de a modulo n
def cle_prv(p,q,e):
    assert ispremier(p) and ispremier(q)
    n=p*q
    phi=(p-1)*(q-1)
    return(inverse(e,phi))
def crypt_rsa(m,n,e):
    return pow(m,e,n)

```

Cryptage/decryptage