Cryptographie

Vidéo ■ partie 1. Le chiffrement de César
Vidéo ■ partie 2. Le chiffrement de Vigenère
Vidéo ■ partie 3. La machine Enigma et les clés secrètes
Vidéo ■ partie 4. La cryptographie à clé publique
Vidéo ■ partie 5. L'arithmétique pour RSA
Vidéo ■ partie 6. Le chiffrement RSA

1. Le chiffrement de César

1.1. César a dit...

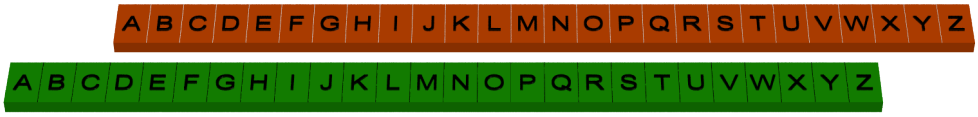
Jules César a-t-il vraiment prononcé la célèbre phrase :

DOHD MDFWD HVW

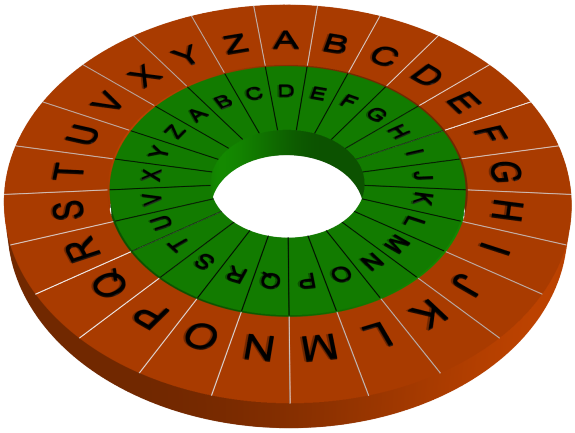
ou bien comme le disent deux célèbres Gaulois : « Ils sont fous ces romains ! ».
En fait César, pour ses communications importantes à son armée, cryptait ses messages. Ce que l'on appelle le chiffrement de César est un décalage des lettres : pour crypter un message, A devient D, B devient E, C devient F,...

$$A \longmapsto D \quad B \longmapsto E \quad C \longmapsto F \quad \dots \quad W \longmapsto Z \quad X \longmapsto A \quad Y \longmapsto B \quad Z \longmapsto C$$

Voici une figure avec l'alphabet d'origine en haut et en **rouge**, en correspondance avec l'alphabet pour le chiffrement en-dessous et en **vert**.



Nous adopterons la convention suivante, en **vert** c'est la partie du message à laquelle tout le monde a accès (ou qui pourrait être intercepté), c'est donc le message crypté. Alors qu'en **rouge** c'est la partie du message confidentiel, c'est le message en clair.
Pour prendre en compte aussi les dernières lettres de l'alphabet, il est plus judicieux de représenté l'alphabet sur un anneau. Ce décalage est un ***décalage circulaire*** sur les lettres de l'alphabet.



Pour déchiffrer le message de César, il suffit de décaler les lettres dans l'autre sens, D se déchiffre en A, E en B,...
Et la célèbre phrase de César est :

ALEA JACTA EST

qui traduite du latin donne « Les dés sont jetés ».

1.2. Des chiffres et des lettres

Il est plus facile de manipuler des nombres que des lettres, aussi nous passons à une formulation arithmétique. Nous associons à chacune des 26 lettres de A à Z un nombre de 0 à 25. En termes mathématiques, nous définissons une bijection :

$$f : \{A, B, C, \ldots, Z\} \longrightarrow \{0, 1, 2, \ldots, 25\}$$

par

$$A \longmapsto 0 \quad B \longmapsto 1 \quad C \longmapsto 2 \quad \ldots \quad Z \longmapsto 25$$

Ainsi "**A L E A**" devient "**0 11 4 0**".
Le chiffrement de César est un cas particulier de ***substitution mono-alphabétique***, c'est-à-dire un chiffrement lettre à lettre.
Quel est l'intérêt ? Nous allons voir que le chiffrement de César correspond à une opération mathématique très simple. Pour cela, rappelons la notion de congruence et l'ensemble $\mathbb{Z}/26\mathbb{Z}$.

1.3. Modulo

Soit $n \geqslant 2$ un entier fixé.

Définition 1.
On dit que a ***est congru à*** b ***modulo*** n, si n divise $b - a$. On note alors

$$a \equiv b \pmod{n}.$$

Pour nous $n = 26$. Ce qui fait que $28 \equiv 2 \pmod{26}$, car $28 - 2$ est bien divisible par 26. De même $85 = 3 \times 26 + 7$ donc $85 \equiv 7 \pmod{26}$.
On note $\mathbb{Z}/26\mathbb{Z}$ l'ensemble de tous les éléments de $\mathbb{Z}$ modulo 26. Cet ensemble peut par exemple être représenté par les 26 éléments $\{0, 1, 2, \ldots, 25\}$. En effet, puisqu'on compte modulo 26 :

$$0,\ 1,\ 2,\ \ldots,\ 25, \quad \text{puis} \quad 26 \equiv 0,\ 27 \equiv 1,\ 28 \equiv 2, \ldots, \quad 52 \equiv 0,\ 53 \equiv 1, \ldots$$

et de même $-1 \equiv 25$, $-2 \equiv 24$,...

Plus généralement $\mathbb{Z}/n\mathbb{Z}$ contient n éléments. Pour un entier $a \in \mathbb{Z}$ quelconque, son ***représentant*** dans $\{0, 1, 2, \ldots, n-1\}$ s'obtient comme le reste k de la division euclidienne de a par n : $a = bn + k$. De sorte que $a \equiv k \pmod{n}$ et $0 \leqslant k < n$.

De façon naturelle l'addition et la multiplication d'entiers se transposent dans $\mathbb{Z}/n\mathbb{Z}$.

Pour $a, b \in \mathbb{Z}/n\mathbb{Z}$, on associe $a + b \in \mathbb{Z}/n\mathbb{Z}$.

Par exemple dans $\mathbb{Z}/26\mathbb{Z}$, $15+13$ égale 2. En effet $15+13=28\equiv 2 \pmod{26}$. Autre exemple : que vaut $133+64$? $133+64=197=7\times 26+15\equiv 15 \pmod{26}$. Mais on pourrait procéder différemment : tout d'abord $133=5\times 26+3\equiv 3 \pmod{26}$ et $64=2\times 26+12\equiv 12 \pmod{26}$. Et maintenant sans calculs : $133+64\equiv 3+12\equiv 15 \pmod{26}$.

On fait de même pour la multiplication : pour $a, b\in\mathbb{Z}/n\mathbb{Z}$, on associe $a\times b\in\mathbb{Z}/n\mathbb{Z}$.
Par exemple 3×12 donne 10 modulo 26, car $3\times 12=36=1\times 26+10\equiv 10 \pmod{26}$. De même : $3\times 27=81=3\times 26+3\equiv 3 \pmod{26}$. Une autre façon de voir la même opération est d'écrire d'abord $27=1 \pmod{26}$ puis $3\times 27\equiv 3\times 1\equiv 3 \pmod{26}$.

1.4. Chiffrer et déchiffrer

Le chiffrement de César est simplement une addition dans $\mathbb{Z}/26\mathbb{Z}$! Fixons un entier k qui est le décalage (par exemple $k=3$ dans l'exemple de César ci-dessus) et définissons la ***fonction de chiffrement de César de décalage k*** qui va de l'ensemble $\mathbb{Z}/26\mathbb{Z}$ dans lui-même :

$$C_k : \begin{cases} \mathbb{Z}/26\mathbb{Z} & \longrightarrow & \mathbb{Z}/26\mathbb{Z} \\ x & \longmapsto & x+k \end{cases}$$

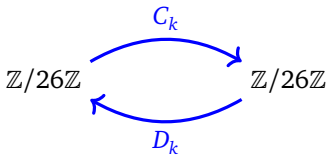
Par exemple, pour $k=3$: $C_3(0)=3$, $C_3(1)=4\ldots$
Pour déchiffrer, rien de plus simple ! Il suffit d'aller dans l'autre sens, c'est-à-dire ici de soustraire. La ***fonction de déchiffrement de César de décalage k*** est

$$D_k : \begin{cases} \mathbb{Z}/26\mathbb{Z} & \longrightarrow & \mathbb{Z}/26\mathbb{Z} \\ x & \longmapsto & x-k \end{cases}$$

En effet, si 1 a été chiffré en 4, par la fonction C_3 alors $D_3(4)=4-3=1$. On retrouve le nombre original. Mathématiquement, D_k est la bijection réciproque de C_k, ce qui implique que pour tout $x\in\mathbb{Z}/26\mathbb{Z}$:

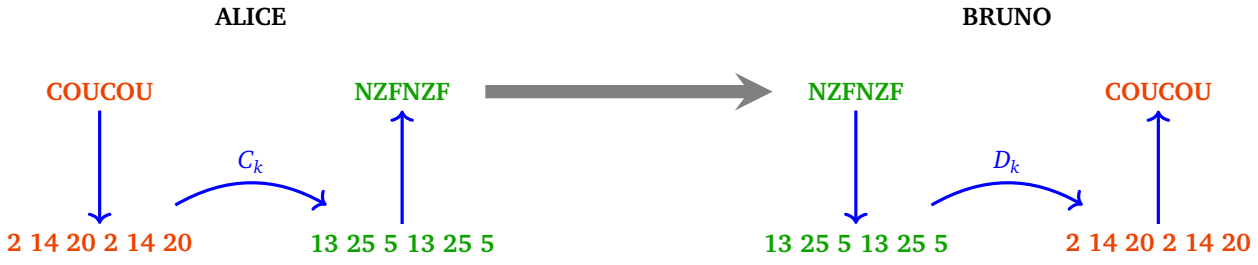
$$D_k\big(C_k(x)\big)=x$$

En d'autres termes, si x est un nombre, on applique la fonction de chiffrement pour obtenir le nombre crypté $y=C_k(x)$; ensuite la fonction de déchiffrement fait bien ce que l'on attend d'elle $D_k(y)=x$, on retrouve le nombre original x.



Une autre façon de voir la fonction de déchiffrement est de remarquer que $D_k(x)=C_{-k}(x)$. Par exemple $C_{-3}(x)=x+(-3)\equiv x+23 \pmod{26}$.

Voici le principe du chiffrement : Alice veut envoyer des messages secrets à Bruno. Ils se sont d'abord mis d'accord sur une clé secrète k, par exemple $k=11$. Alice veut envoyer le message "**COUCOU**" à Bruno. Elle transforme "**COUCOU**" en "**2 14 20 2 14 20**". Elle applique la fonction de chiffrement $C_{11}(x)=x+11$ à chacun des nombres : "**13 25 5 13 25 5**" ce qui correspond au mot crypté "**NZFNZF**". Elle transmet le mot crypté à Bruno, qui selon le même principe applique la fonction de déchiffrement $D_{11}(x)=x-11$.



Exemple 1.
Un exemple classique est le "rot13" (pour rotation par un décalage de 13) :

$$C_{13}(x) = x + 13$$

et comme $-13 \equiv 13 \pmod{26}$ alors $D_{13}(x) = x + 13$. La fonction de déchiffrement est la même que la fonction de chiffrement !
Exemple : déchiffrez le mot "**PRFNE**".

Notons ici deux points importants pour la suite : tout d'abord nous avons naturellement considéré un mot comme une succession de lettres, et chaque opération de chiffrement et déchiffrement s'effectue sur un bloc d'une seule lettre. Ensuite nous avons vu que chiffrer un message est une opération mathématique (certes sur un ensemble un peu spécial).

1.5. Espace des clés et attaque

Combien existe-t-il de possibilités de chiffrement par la méthode de César ? Il y a 26 fonctions C_k différentes, $k = 0, 1, \ldots, 25$. Encore une fois, k appartient à $\mathbb{Z}/26\mathbb{Z}$, car par exemple les fonctions C_{29} et C_3 sont identiques. Le décalage k s'appelle la ***clé de chiffrement***, c'est l'information nécessaire pour crypter le message. Il y a donc 26 clés différentes et l'***espace des clés*** est $\mathbb{Z}/26\mathbb{Z}$.

Il est clair que ce chiffrement de César est d'une sécurité très faible. Si Alice envoie un message secret à Bruno et que Chloé intercepte ce message, il sera facile pour Chloé de le décrypter même si elle ne connaît pas la clé secrète k. L'attaque la plus simple pour Chloé est de tester ce que donne chacune des 26 combinaisons possibles et de reconnaître parmi ces combinaisons laquelle donne un message compréhensible.

1.6. Algorithmes

Les ordinateurs ont révolutionné la cryptographie et surtout le décryptage d'un message intercepté. Nous montrons ici, à l'aide du langage `Python` comment programmer et attaquer le chiffrement de César. Tout d'abord la fonction de chiffrement se programme en une seule ligne :

Code 1 (*cesar.py (1)*).
```
def cesar_chiffre_nb(x,k):
    return (x+k)%26
```

Ici `x` est un nombre de $\{0, 1, \ldots, 25\}$ et `k` est le décalage. `(x+k)%26` a pour valeur le reste modulo 26 de la somme `(x+k)`. Pour le décryptage, c'est aussi simple :

Code 2 (*cesar.py (2)*).
```
def cesar_dechiffre_nb(x,k):
    return (x-k)%26
```

Pour chiffrer un mot ou une phrase, il n'y a pas de problèmes théoriques, mais seulement des difficultés techniques :

- Un mot ou une phrase est une chaîne de caractères, qui en fait se comporte comme une liste. Si `mot` est une chaîne alors `mot[0]` est la première lettre, `mot[1]` la deuxième lettre... et la boucle `for lettre in mot:` permet de parcourir chacune des lettres.
- Pour transformer une lettre en un nombre, on utilise le code Ascii qui à chaque caractère associe un nombre, `ord(A)` vaut 65, `ord(B)` vaut 66... Ainsi `(ord(lettre) - 65)` renvoie le rang de la lettre entre 0 et 25 comme nous l'avons fixé dès le départ.
- La transformation inverse se fait par la fonction `chr` : `chr(65)` renvoie le caractère `A`, `chr(66)` renvoie `B`...
- Pour ajouter une lettre à une liste, faites `maliste.append(lettre)`. Enfin pour transformer une liste de caractères en une chaîne, faites `"".join(maliste)`.

Ce qui donne :

Code 3 (*cesar.py (3)*).
```
def cesar_chiffre_mot(mot,k):
```

```
    message_code = []                       # Liste vide
    for lettre in mot:                      # Pour chaque lettre
        nb = ord(lettre)-65                 # Lettre devient nb de 0 à 25
        nb_crypte = cesar_chiffre_nb(nb,k) # Chiffrement de César
        lettre_crypte = chr(nb_crypte+65)  # Retour aux lettres
        message_code.append(lettre_crypte) # Ajoute lettre au message
    message_code = "".join(message_code)    # Revient à chaine caractères
    return(message_code)
```

Pour l'attaque on parcourt l'intégralité de l'espace des clés : k varie de 0 à 25. Noter que pour décrypter les messages on utilise ici simplement la fonction de César avec la clé $-k$.

Code 4 (*cesar.py (4)*).

```
def cesar_attaque(mot):
    for k in range(26):
        print(cesar_chiffre_mot(mot,-k))
    return None
```

2. Le chiffrement de Vigenère

2.1. Substitution mono-alphabétique

Principe

Nous avons vu que le chiffrement de César présente une sécurité très faible, la principale raison est que l'espace des clés est trop petit : il y a seulement 26 clés possibles, et on peut attaquer un message chiffré en testant toutes les clés à la main.
Au lieu de faire correspondre circulairement les lettres, on associe maintenant à chaque lettre une autre lettre (sans ordre fixe ou règle générale).
Par 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
F	Q	B	M	X	I	T	E	P	A	L	W	H	S	D	O	Z	K	V	G	R	C	N	Y	J	U

Pour crypter le message

ETRE OU NE PAS ETRE TELLE EST LA QUESTION

on regarde la correspondance et on remplace la lettre E par la lettre X, puis la lettre T par la lettre G, puis la lettre R par la lettre K...
Le message crypté est alors :

XGKX DR SX OFV XGKX GXWWX XVG WF ZRXVGPDS

Pour le décrypter, en connaissant les substitutions, on fait l'opération inverse.

Avantage : nous allons voir que l'espace des clés est gigantesque et qu'il n'est plus question d'énumérer toutes les possibilités.
Inconvénients : la clé à retenir est beaucoup plus longue, puisqu'il faut partager la clé constituée des 26 lettres "FQBMX...". Mais surtout, nous allons voir que finalement ce protocole de chiffrement est assez simple à « craquer ».

Espace des clés

Mathématiquement, le choix d'une clé revient au choix d'une bijection de l'ensemble $\{A, B, \ldots, Z\}$ vers le même ensemble $\{A, B, \ldots, Z\}$. Il y a 26! choix possibles. En effet pour la lettre A de l'ensemble de départ, il y a 26 choix possibles (nous avions choisi F), pour B il reste 25 choix possibles (tout sauf F qui est déjà choisi), pour C il reste 24 choix... enfin pour Z il ne reste qu'une seule possibilité, la seule lettre non encore choisie. Au final il y a : $26 \times 25 \times 24 \times \cdots \times 2 \times 1$ soit 26! choix de clés. Ce qui fait environ 4×10^{26} clés. Il y a plus de clés différentes que

de grains de sable sur Terre ! Si un ordinateur pouvait tester 1 milliard de clés par seconde, il lui faudrait alors 12 milliards d'années pour tout énumérer.

Attaque statistique

La principale faiblesse du chiffrement mono-alphabétique est qu'une même lettre est toujours chiffrée de la même façon. Par exemple, ici **E** devient **X**. Dans les textes longs, les lettres n'apparaissent pas avec la même fréquence. Ces fréquences varient suivant la langue utilisée. En français, les lettres les plus rencontrées sont dans l'ordre :

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

avec les fréquences (souvent proches et dépendant de l'échantillon utilisé) :

E	S	A	I	N	T	R	U	L	O	D
14.69%	8.01%	7.54%	7.18%	6.89%	6.88%	6.49%	6.12%	5.63%	5.29%	3.66%

Voici la méthode d'attaque : dans le texte crypté, on cherche la lettre qui apparaît le plus, et si le texte est assez long cela devrait être le chiffrement du **E**, la lettre qui apparaît ensuite dans l'étude des fréquences devrait être le chiffrement du **S**, puis le chiffrement du **A**... On obtient des morceaux de texte clair sous la forme d'une texte à trous et il faut ensuite deviner les lettres manquantes.

Par exemple, déchiffrons la phrase :

LHLZ HFQ BC HFFPZ WH YOUPFH MUPZH

On compte les apparitions des lettres :

H : 6 **F** : 4 **P** : 3 **Z** : 3

On suppose donc que le **H** crypte la lettre **E**, le **F** la lettre **S**, ce qui donne

***E** ES* ** ESS** *E ***SE ****E**

D'après les statistiques **P** et **Z** devraient se décrypter en **A** et **I** (ou **I** et **A**). Le quatrième mot "**HFFPZ**", pour l'instant décrypté en "**ESS****", se complète donc en "**ESSAI**" ou "**ESSIA**". La première solution semble correcte ! Ainsi **P** crypte **A**, et **Z** crypte **I**. La phrase est maintenant :

***E*I ES* ** ESSAI *E ***ASE **AIE**

En réfléchissant un petit peu, on décrypte le message :

CECI EST UN ESSAI DE PHRASE VRAIE

2.2. Le chiffrement de Vigenère

Blocs

L'espace des clés du chiffrement mono-alphabétique est immense, mais le fait qu'une lettre soit toujours cryptée de la même façon représente une trop grande faiblesse. Le chiffrement de Vigenère remédie à ce problème. On regroupe les lettres de notre texte par blocs, par exemple ici par blocs de longueur 4 :

CETTE PHRASE NE VEUT RIEN DIRE

devient

CETT EPHR ASEN EVEU TRIE NDIR E

(les espaces sont purement indicatifs, dans la première phrase ils séparent les mots, dans la seconde ils séparent les blocs).

Si k est la longueur d'un bloc, alors on choisit une clé constituée de k nombres de 0 à 25 : $(n_1, n_2, \ldots, n_k)$. Le chiffrement consiste à effectuer un chiffrement de César, dont le décalage dépend du rang de la lettre dans le bloc :

- un décalage de n_1 pour la première lettre de chaque bloc,
- un décalage de n_2 pour la deuxième lettre de chaque bloc,
- ...
- un décalage de n_k pour la k-ème et dernière lettre de chaque bloc.

Pour notre exemple, si on choisit comme clé $(3, 1, 5, 2)$ alors pour le premier bloc "**CETT**" :

- un décalage de 3 pour **C** donne **F**,
- un décalage de 1 pour **E** donne **F**,
- un décalage de 5 pour le premier **T** donne **Y**,
- un décalage de 2 pour le deuxième **T** donne **V**.

Ainsi "**CETT**" de vient "**FFYV**". Vous remarquez que les deux lettres **T** ne sont pas cryptées par la même lettre et que les deux **F** ne cryptent pas la même lettre. On continue ensuite avec le deuxième bloc...

Mathématiques

L'élément de base n'est plus une lettre mais un ***bloc***, c'est-à-dire un regroupement de lettres. La fonction de chiffrement associe à un bloc de longueur k, un autre bloc de longueur k, ce qui donne en mathématisant les choses :

$$C_{n_1,n_2,\dots,n_k} : \begin{cases} \mathbb{Z}/26\mathbb{Z} \times \mathbb{Z}/26\mathbb{Z} \times \cdots \times \mathbb{Z}/26\mathbb{Z} & \longrightarrow & \mathbb{Z}/26\mathbb{Z} \times \mathbb{Z}/26\mathbb{Z} \times \cdots \times \mathbb{Z}/26\mathbb{Z} \\ (x_1, x_2, \dots, x_k) & \longmapsto & (x_1 + n_1, x_2 + n_2, \dots, x_k + n_k) \end{cases}$$

Chacune des composantes de cette fonction est un chiffrement de César. La fonction de déchiffrement est juste $C_{-n_1,-n_2,\dots,-n_k}$.

Espace des clés et attaque

Il y a 26^k choix possibles de clés, lorsque les blocs sont de longueur k. Pour des blocs de longueur $k = 4$ cela en donne déjà 456 976, et même si un ordinateur teste toutes les combinaisons possibles sans problème, il n'est pas aisé de parcourir cette liste pour trouver le message en clair, c'est-à-dire celui qui est compréhensible !
Il persiste tout de même une faiblesse du même ordre que celle rencontrée dans le chiffrement mono-alphabétique : la lettre **A** n'est pas toujours cryptée par la même lettre, mais si deux lettres **A** sont situées à la même position dans deux blocs différents (comme par exemple "**ALPH ABET**") alors elles seront cryptées par la même lettre.
Une attaque possible est donc la suivante : on découpe notre message en plusieurs listes, les premières lettres de chaque bloc, les deuxièmes lettres de chaque bloc... et on fait une attaque statistique sur chacun de ces regroupements. Ce type d'attaque n'est possible que si la taille des blocs est petite devant la longueur du texte.

2.3. Algorithmes

Voici un petit algorithme qui calcule la fréquence de chaque lettre d'une phrase.

Code 5 (*statistiques.py*).

```
def statistiques(phrase):
    liste_stat = [0 for x in range(26)]     # Une liste avec des 0
    for lettre in phrase:                   # On parcourt la phrase
        i = ord(lettre)-65
        if 0 <= i < 26:                     # Si c'est une vraie lettre
          liste_stat[i] = liste_stat[i] + 1
    return(liste_stat)
```

Et voici le chiffrement de Vigenère.

Code 6 (*vigenere.py*).

```
def vigenere(mot,cle):                  # Clé est du type [n_1,...,n_k]
    message_code = []
    k = len(cle)                        # Longueur de la clé
    i = 0                               # Rang dans le bloc
    for lettre in mot:                  # Pour chaque lettre
        nomb = ord(lettre)-65           # Lettre devient nb de 0 à 25
        nomb_code = (nomb+cle[i]) % 26  # Vigenère : on ajoute n_i
        lettre_code = chr(nomb_code+65) # On repasse aux lettres
        i=(i+1) % k                     # On passe au rang suivant
        message_code.append(lettre_code) # Ajoute lettre au message
    message_code = "".join(message_code) # Revient à chaine caractères
    return(message_code)
```

3. La machine Enigma et les clés secrètes

3.1. Un secret parfait

L'inconvénient des chiffrements précédents est qu'une même lettre est régulièrement chiffrée de la même façon, car la correspondance d'un alphabet à un ou plusieurs autres est fixée une fois pour toutes, ce qui fait qu'une attaque statistique est toujours possible. Nous allons voir qu'en changeant la correspondance à chaque lettre, il est possible de créer un chiffrement parfait !
Expliquons d'abord le principe à l'aide d'une analogie : j'ai choisi deux entiers m et c tels que $m+c=100$. Que vaut m ? C'est bien sûr impossible de répondre car il y a plusieurs possibilités : $0+100$, $1+99$, $2+98$,... Par contre, si je vous donne aussi c alors vous trouvez m immédiatement $m=100-c$.

Voici le principe du chiffrement : Alice veut envoyer à Bruno le message secret M suivant :

ATTAQUE LE CHATEAU

Alice a d'abord choisi une clé secrète C qu'elle a transmise à Bruno. Cette clé secrète est de la même longueur que le message (les espaces ne comptent pas) et composée d'entiers de 0 à 25, tirés au hasard. Par exemple C :

[4, 18, 2, 0, 21, 12, 18, 13, 7, 11, 23, 22, 19, 2, 16, 9]

Elle crypte la première lettre par un décalage de César donné par le premier entier : **A** est décalé de 4 lettres et devient donc **E**. La seconde lettre est décalée du second entier : le premier **T** devient **L**. Le second **T** est lui décalé de 2 lettres, il devient **V**. Le **A** suivant est décalé de 0 lettre, il reste **A**... Alice obtient un message chiffré X qu'elle transmet à Bruno :

ELVALGW YL NEWMGQD

Pour le décrypter, Bruno, qui connaît la clé, n'a qu'à faire le décalage dans l'autre sens.
Notez que deux lettres identiques (par exemples les **T**) n'ont aucune raison d'être cryptées de la même façon. Par exemple, les **T** du message initial sont cryptés dans l'ordre par un **L**, un **V** et un **M**.

Formalisons un peu cette opération. On identifie A avec 0, B avec 1, ..., Z avec 25. Alors le message crypté X est juste la "somme" du message M avec la clé secrète C, la somme s'effectuant lettre à lettre, terme à terme, modulo 26.
Notons cette opération $M \oplus C = X$.

	A	T	T	A	Q	U	E	L	E	C	H	A	T	E	A	U
	0	19	19	0	16	20	4	11	4	2	7	0	19	4	0	20
$\oplus$																
	4	18	2	0	21	12	18	13	7	11	23	22	19	2	16	9
$=$	4	11	21	0	11	6	22	24	11	13	4	22	12	6	16	3
	E	L	V	A	L	G	W	Y	L	N	E	W	M	G	Q	D

Bruno reçoit X et connaît C, il effectue donc $X \ominus C = M$.
Pourquoi ce système est-il inviolable ? Pour chacune des lettres, c'est exactement le même problème que trouver m, sachant que $m+c=x$ (où $x=100$), mais sans connaître c. Toutes les possibilités pour m pourraient être juste. Et bien sûr, dès que l'on connaît c, la solution est triviale : $m=x-c$.
Il y a trois principes à respecter pour que ce système reste inviolable :

1. La longueur de la clé est égale à la longueur du message.
2. La clé est choisie au hasard.
3. La clé ne sert qu'une seule fois.

Ce système appelé "masque jetable" ou chiffrement de Vernam est parfait en théorie, mais sa mise en œuvre n'est pas pratique du tout ! Tout d'abord il faut que la clé soit aussi longue que le message. Pour un message court cela ne pose pas de problème, mais pour envoyer une image par exemple cela devient très lourd. Ensuite, il faut trouver un moyen sûr d'envoyer la clé secrète à son interlocuteur avant de lui faire parvenir le message. Et il faut recommencer cette opération à chaque message, ou bien se mettre d'accord dès le départ sur un ***carnet de clés*** : une longue liste de clés secrètes.
Pour justifier que ce système est vraiment inviolable voici une expérience amusante : Alice veut envoyer le message M ="**ATTAQUE LE CHATEAU**" à Bruno, elle choisit la clé secrète C=[4, 18, 2, 0,...] comme ci-dessus et obtient le message chiffré X="**ELVA**..." qu'elle transmet à Bruno.

Alice se fait kidnapper par Chloé, qui veut l'obliger à déchiffrer son message. Heureusement, Alice a anticipé les soucis : elle a détruit le message M, la clé secrète C et a créé un faux message M' et une fausse clé secrète C'. Alice fournit cette fausse clé secrète C' à Chloé, qui déchiffre le message par l'opération $X \ominus C'$ et elle trouve le message bien inoffensif M' :

RECETTE DE CUISINE

Alice est innocentée !
Comment est-ce possible ? Alice avait au préalable préparé un message neutre M' de même longueur que M et calculé la fausse clé secrète $C' = X \ominus M'$. Chloé a obtenu (par la contrainte) X et C', elle déchiffre le message ainsi

$$X \ominus C' = X \ominus (X \ominus M') = (X \ominus X) \oplus M' = M'$$

Chloé trouve donc le faux message.
Ici la fausse clé C' est :

[13, 7, 19, 22, 18, 13, 18, 21, 7, 11, 10, 14, 20, 24, 3, 25]

La première lettre du message chiffré est un **E**, en reculant de 13 lettres dans l'alphabet, elle se déchiffre en **R**...

3.2. La machine Enigma

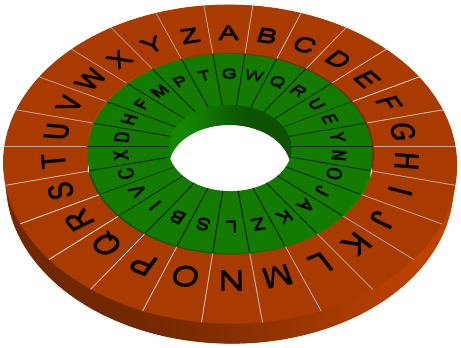
Afin de s'approcher de ce protocole de chiffrement parfait, il faut trouver un moyen de générer facilement de longues clés, comme si elles avaient été générées au hasard. Nous allons étudier deux exemples utilisés en pratique à la fin du siècle dernier, une méthode électro-mécanique : la machine Enigma et une méthode numérique : le DES.
La machine Enigma est une machine électro-mécanique qui ressemble à une machine à écrire. Lorsque qu'une touche est enfoncée, des disques internes sont actionnés et le caractère crypté s'allume. Cette machine, qui sert aussi au déchiffrement, était utilisée pour les communications de l'armée allemande durant la seconde guerre mondiale. Ce que les Allemands ne savaient pas, c'est que les services secrets polonais et britanniques avaient réussi à percer les secrets de cette machine et étaient capables de déchiffrer les messages transmis par les allemands. Ce long travail d'études et de recherches a nécessité tout le génie d'Alan Turing et l'invention de l'ancêtre de l'ordinateur.

Nous symbolisons l'élément de base de la machine Enigma par deux anneaux :

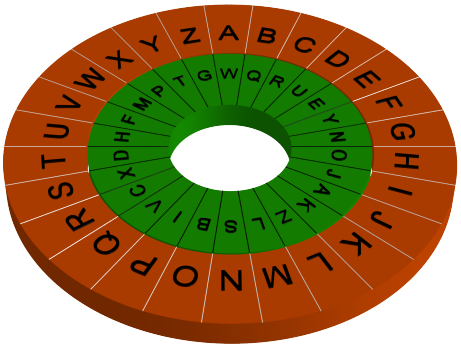
- Un anneau extérieur contenant l'alphabet "**ABCDE**..." symbolisant le clavier de saisie des messages. Cet anneau est fixe.
- Un anneau intérieur contenant un alphabet dans le désordre (sur la figure "**GWQRU**..."). Cet anneau est mobile et effectue une rotation à chaque touche tapée au clavier. Il représente la clé secrète.

Voici, dans ce cas, le processus de chiffrement du mot "**BAC**", avec la clé de chiffrement "**G**" :

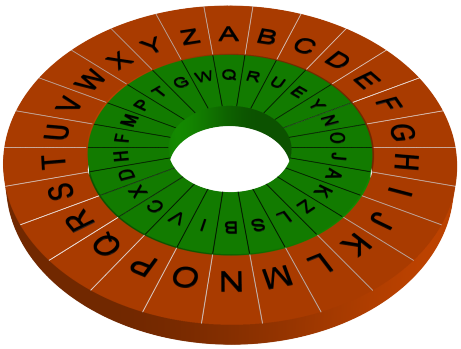
1. **Position initiale.** L'opérateur tourne l'anneau intérieur de sorte que le **A** extérieur et fixe soit en face du **G** intérieur (et donc **B** en face de **W**).



2. **Première lettre.** L'opérateur tape la première lettre du message : **B**, la machine affiche la correspondance **W**.
3. **Rotation.** L'anneau intérieur tourne de 1/26ème de tour, maintenant le **A** extérieur et fixe est en face du **W**, le **B** en face du **Q**,...

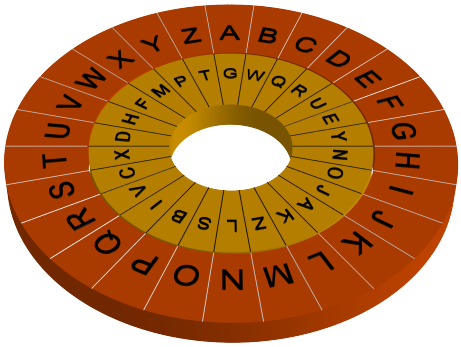


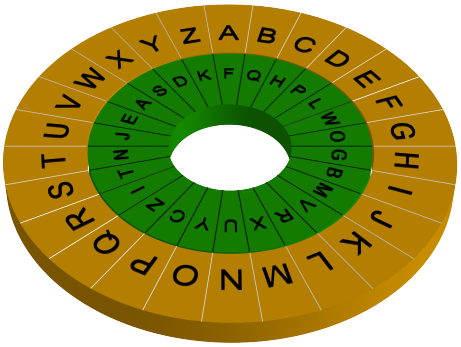
4. **Deuxième lettre.** L'opérateur tape la deuxième lettre du message **A**, la machine affiche la correspondance, c'est de nouveau **W**.
5. **Rotation.** L'anneau intérieur tourne de 1/26ème de tour, maintenant le **A** extérieur et fixe est en face du **Q**, le **B** en face du **R**, le **C** en face du **U**,...



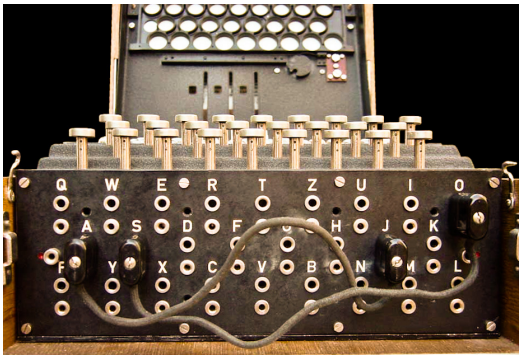
6. **Troisième lettre.** L'opérateur tape la troisième lettre du message **C**, la machine affiche la correspondance **U**.
7. **Rotation.** L'anneau intérieur effectue sa rotation.
8. **Message chiffré.** Le message crypté est donc "**WWU**"

Cette méthode de chiffrement est identique à un chiffrement de type Vigenère pour une clé de longueur 26. Il y a 26 clés différents à disposition avec un seul anneau intérieur et identifiées par lettre de la position initiale : **G**, **W**, **Q**... **T** correspondant aux alphabets : "**GWQ...PT**", "**WQR...TG**", "**QRU...GW**"...
En fait, la machine Enigma était beaucoup plus sophistiquée, il n'y avait pas un mais plusieurs anneaux intérieurs. Par exemple pour deux anneaux intérieurs comme sur la figure : **B** s'envoie sur **W**, qui s'envoie sur **A** ; la lettre **B** est cryptée en **A**. Ensuite l'anneau intérieur numéro 1 effectue 1/26ème de tour. La lettre **A** s'envoie sur **W**, qui s'envoie sur **A** ; la lettre **A** est cryptée en **A**. Lorsque l'anneau intérieur numéro 1 a fait une rotation complète (26 lettres ont été tapées) alors l'anneau intérieur numéro 2 effectue 1/26ème de tour. C'est comme sur un compteur kilométrique, lorsque le chiffre des kilomètres parcourt $0, 1, 2, 3, \ldots, 9$, alors au kilomètre suivant, le chiffre des kilomètres est 0 et celui des dizaines de kilomètres est augmenté d'une unité.





S'il y a trois anneaux, lorsque l'anneau intérieur 2 a fait une rotation complète, l'anneau intérieur 3 tourne de 1/26ème de tour. Il y a alors 26^3 clés différentes facilement identifiables par les trois lettres des positions initiales des anneaux. Il fallait donc pour utiliser cette machine, d'abord choisir les disques (nos anneaux intérieurs) les placer dans un certain ordre, fixer la position initiale de chaque disque. Ce système était rendu largement plus complexe avec l'ajout de correspondances par fichage entre les lettres du clavier (voir photo). Le nombre de clés possibles dépassait plusieurs milliards de milliards !



3.3. La ronde des chiffres : DES

La machine Enigma génère mécaniquement un alphabet différent à chaque caractère crypté, tentant de se rapprocher d'un chiffrement parfait. Nous allons voir une autre méthode, cette fois numérique : le DES. Le DES (*Data Encryption Standard*) est un protocole de chiffrement par blocs. Il a été, entre 1977 et 2001, le standard de chiffrement pour les organisations du gouvernement des États-Unis et par extension pour un grand nombre de pays dans le monde.

Commençons par rappeler que l'objectif est de générer une clé aléatoire de grande longueur. Pour ne pas avoir à retenir l'intégralité de cette longue clé, on va la générer de façon pseudo-aléatoire à partir d'une petite clé.
Voyons un exemple élémentaire de suite pseudo-aléatoire.
Soit (u_n) la suite définie par la donnée de (a, b) et de u_0 et la relation de récurrence

$$u_{n+1} \equiv a \times u_n + b \pmod{26}.$$

Par exemple pour $a = 2$, $b = 5$ et $u_0 = 6$, alors les premiers termes de la suites sont :

$$6 \quad 17 \quad 13 \quad 5 \quad 15 \quad 9 \quad 23 \quad 25 \quad 3 \quad 11 \quad 1 \quad 7 \quad 19 \quad 17 \quad 13 \quad 5$$

Les trois nombres (a, b, u_0) représentent la clé principale et la suite des $(u_n)_{n\in\mathbb{N}}$ les clés secondaires.
Avantages : à partir d'une clé principale courte (ici trois nombres) on a généré une longue liste de clés secondaires. Inconvénients : la liste n'est pas si aléatoire que cela, elle se répète ici avec une période de longueur 12 : $17, 13, 5, \ldots, 17, 13, 5, \ldots$

Le système DES est une version sophistiquée de ce processus : à partir d'une clé courte et d'opérations élémentaires on crypte un message. Comme lors de l'étude de la machine Enigma, nous allons présenter une version très simplifiée de ce protocole afin d'en expliquer les étapes élémentaires.
Pour changer, nous allons travailler modulo 10. Lorsque l'on travaille par blocs, les additions se font *bit* par *bit*. Par exemple : $[1\ 2\ 3\ 4] \oplus [7\ 8\ 9\ 0] = [8\ 0\ 2\ 4]$ car $(1 + 7 \equiv 8 \pmod{10}, 2 + 8 \equiv 0 \pmod{10}, \ldots)$
Notre message est coupé en blocs, pour nos explications ce seront des blocs de longueur 8. La clé est de longueur 4.
Voici le message (un seul bloc) : $M = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8]$ et voici la clé : $C = [3\ 1\ 3\ 2]$.

Étape 0. Initialisation. On note $M_0 = M$ et on découpe M en une partie gauche et une partie droite

$$M_0 = [G_0 \parallel D_0] = [1\ 2\ 3\ 4 \parallel 5\ 6\ 7\ 8]$$

Étape 1. Premier tour. On pose

$$M_1 = [D_0 \parallel C \oplus \sigma(G_0)]$$

où σ est une permutation circulaire.
On effectue donc trois opérations pour passer de M_0 à M_1 :

1. On échange la partie droite et la partie gauche de M_0 :
$$M_0 \longmapsto [5\ 6\ 7\ 8 \parallel 1\ 2\ 3\ 4]$$
2. Sur la nouvelle partie droite, on permute circulairement les nombres :
$$\longmapsto [5\ 6\ 7\ 8 \parallel 2\ 3\ 4\ 1]$$
3. Puis on ajoute la clé secrète C à droite (ici $C = [3\ 1\ 3\ 2]$) :
$$\longmapsto [5\ 6\ 7\ 8 \parallel 5\ 4\ 7\ 3] = M_1$$

On va recommencer le même processus. Cela revient à appliquer la formule de récurrence, qui partant de $M_i = [G_i \parallel D_i]$, définit

$$M_{i+1} = [D_i \parallel C \oplus \sigma(G_i)]$$

Étape 2. Deuxième tour. On part de $M_1 = [5\ 6\ 7\ 8 \parallel 5\ 4\ 7\ 3]$.

1. On échange la partie droite et la partie gauche de M_0 :
$$M_0 \longmapsto [5\ 4\ 7\ 3 \parallel 5\ 6\ 7\ 8]$$
2. Sur la nouvelle partie droite, on permute circulairement les nombres.
$$\longmapsto [5\ 4\ 7\ 3 \parallel 6\ 7\ 8\ 5]$$
3. Puis on ajoute la clé secrète C à droite.
$$\longmapsto [5\ 4\ 7\ 3 \parallel 9\ 8\ 1\ 7] = M_2$$

On peut décider de s'arrêter après ce tour et renvoyer le message crypté $X = M_2 = [5\ 4\ 7\ 3\ 9\ 8\ 1\ 7]$.
Comme chaque opération élémentaire est inversible, on applique un protocole inverse pour déchiffrer.
Dans le vrai protocole du DES, les blocs sont de taille 64 *bits*, il y a plus de manipulations sur le message et les étapes mentionnées ci-dessus sont effectuées 16 fois (on parle de tours). À chaque tour, une clé différente est utilisée. Il existe donc un préambule à ce protocole : générer 16 clés secondaires (de longueur 48 *bits*) à partir de la clé principale, ce qui se fait selon le principe de la suite pseudo-aléatoire (u_n) expliquée plus haut.

4. La cryptographie à clé publique

Les Grecs pour envoyer des messages secrets rasaient la tête du messager, tatouaient le message sur son crâne et attendaient que les cheveux repoussent avant d'envoyer le messager effectuer sa mission !
Il est clair que ce principe repose uniquement sur le secret de la méthode.

4.1. Le principe de Kerckhoffs

Cette méthode rudimentaire va à l'encontre du principe de Kerckhoffs. Le principe de Kerckhoffs s'énonce ainsi :

«La sécurité d'un système de chiffrement ne doit reposer que sur le secret de la clé.»

Cela se résume aussi par :

«L'ennemi peut avoir connaissance du système de chiffrement.»

Voici le texte original d'Auguste Kerckhoffs de 1883 «La cryptographie militaire» paru dans le *Journal des sciences militaires*.
Il traite notamment des enjeux de sécurité lors des correspondances :

> *«Il faut distinguer entre un système d'écriture chiffré, imaginé pour un échange momentané de lettres entre quelques personnes isolées, et une méthode de cryptographie destinée à régler pour un temps illimité la correspondance des différents chefs d'armée entre eux.»*

Le principe fondamental est le suivant :

> *«Dans le second cas, [...] il faut que* **le système n'exige pas le secret**, *et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi.»*

Ce principe est novateur dans la mesure où intuitivement il semble opportun de dissimuler le maximum de choses possibles : clé et système de chiffrement utilisés. Mais l'objectif visé par Kerckhoffs est plus académique, il pense qu'un système dépendant d'un secret mais dont le mécanisme est connu de tous sera testé, attaqué, étudié, et finalement utilisé s'il s'avère intéressant et robuste.

4.2. Factorisations des entiers

Quels outils mathématiques répondent au principe de Kerckoffs ?
Un premier exemple est la toute simple multiplication ! En effet si je vous demande combien font 5×7, vous répondez 35. Si je vous demande de factoriser 35 vous répondez 5×7. Cependant ces deux questions ne sont pas du même ordre de difficulté. Si je vous demande de factoriser 1591, vous aller devoir faire plusieurs tentatives, alors que si je vous avais directement demandé de calculer 37×43 cela ne pose pas de problème.
Pour des entiers de plusieurs centaines de chiffres le problème de factorisation ne peut être résolu en un temps raisonnable, même pour un ordinateur. C'est ce problème asymétrique qui est à la base de la cryptographie RSA (que nous détaillerons plus tard) : connaître p et q apporte plus d'information utilisable que $p \times q$. Même si en théorie à partir de $p \times q$ on peut retrouver p et q, en pratique ce sera impossible.

Formalisons ceci avec la notion de complexité. La ***complexité*** est le temps de calculs (ou le nombre d'opérations élémentaires) nécessaire pour effectuer une opération.
Commençons par la complexité de l'addition : disons que calculer la somme de deux chiffres (par exemple $6+8$) soit de complexité 1 (par exemple 1 seconde pour un humain, 1 milliseconde pour un ordinateur). Pour calculer la somme de deux entiers à n chiffres, la complexité est d'ordre n (exemple : $1234+2323$, il faut faire 4 additions de chiffres, donc environ 4 secondes pour un humain).
La multiplication de deux entiers à n chiffres est de complexité d'ordre n^2. Par exemple pour multiplier 1234 par 2323 il faut faire 16 multiplications de chiffres (chaque chiffre de 1234 est à multiplier par chaque chiffre de 2323).
Par contre la meilleure méthode de factorisation connue est de complexité d'ordre $\exp(4n^{\frac{1}{3}})$ (c'est moins que $\exp(n)$, mais plus que n^d pour tout d, lorsque n tend vers $+\infty$).
Voici un tableau pour avoir une idée de la difficulté croissante pour multiplier et factoriser des nombres à n chiffres :

n	multiplication	factorisation
3	9	320
4	16	572
5	25	934
10	100	5 528
50	2 500	2 510 835
100	10 000	115 681 968
200	40 000	14 423 748 780

4.3. Fonctions à sens unique

Il existe bien d'autres situations mathématiques asymétriques : les ***fonctions à sens unique***. En d'autres termes, étant donnée une fonction f, il est possible connaissant x de calculer «facilement» $f(x)$; mais connaissant un élément de l'ensemble image de f, il est «difficile» ou impossible de trouver son antécédent.
Dans le cadre de la cryptographie, posséder une fonction à sens unique qui joue le rôle de chiffrement n'a que peu de sens. En effet, il est indispensable de trouver un moyen efficace afin de pouvoir déchiffrer les messages chiffrés. On parle alors de ***fonction à sens unique avec trappe secrète***.

Prenons par exemple le cas de la fonction f suivante :

$$f : x \longmapsto x^3 \pmod{100}.$$

- Connaissant x, trouver $y = f(x)$ est facile, cela nécessite deux multiplications et deux divisions.
- Connaissant y image par f d'un élément x ($y = f(x)$), retrouver x est difficile.

Tentons de résoudre le problème suivant : trouver x tel que $x^3 \equiv 11 \pmod{100}$.
On peut pour cela :

- soit faire une recherche exhaustive, c'est-à-dire essayer successivement 1, 2, 3, ..., 99, on trouve alors :

$$71^3 = 357\,911 \equiv 11 \pmod{100},$$

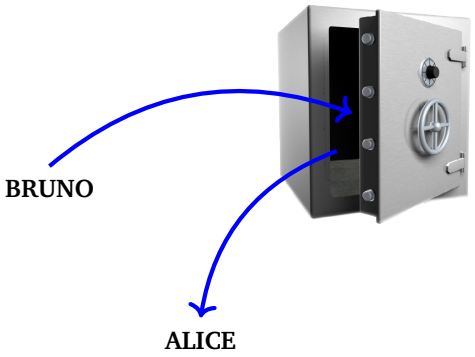
- soit utiliser la trappe secrète : $y \longmapsto y^7 \pmod{100}$ qui fournit directement le résultat !

$$11^7 = 19\,487\,171 \equiv 71 \pmod{100}.$$

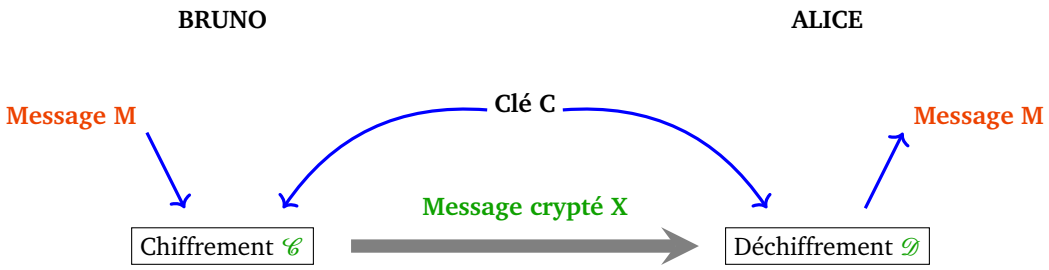
La morale est la suivante : le problème est dur à résoudre, sauf pour ceux qui connaissent la trappe secrète. (Attention, dans le cas de cet exemple, la fonction f n'est pas bijective.)

4.4. Chiffrement à clé secrète

Petit retour en arrière. Les protocoles étudiés dans les chapitres précédents étaient des ***chiffrements à clé secrète***. De façon imagée, tout se passe comme si Bruno pouvaient déposer son message dans un coffre fort pour Alice, Alice et Bruno étant les seuls à posséder la clé du coffre.

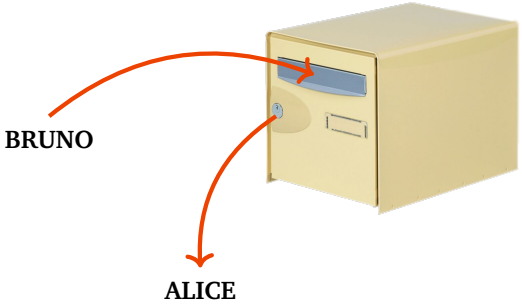


En effet, jusqu'ici, les deux interlocuteurs se partageaient une même clé qui servait à chiffrer (et déchiffrer) les messages. Cela pose bien sûr un problème majeur : Alice et Bruno doivent d'abord se communiquer la clé.

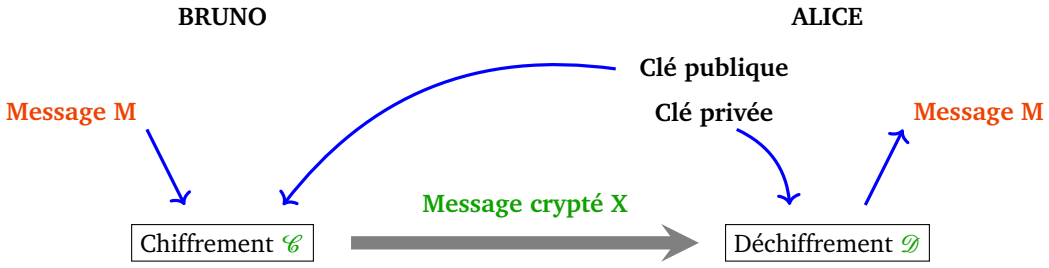


4.5. Chiffrement à clé publique

Les fonctions à sens unique à trappe donnent naissance à des protocoles de chiffrement à clé publique. L'association «clé» et «publique» peut paraître incongrue, mais il signifie que le principe de chiffrement est accessible à tous mais que le déchiffrement nécessite une clé qu'il faut bien sûr garder secrète.



De façon imagée, si Bruno veut envoyer un message à Alice, il dépose son message dans la boîte aux lettres d'Alice, seule Alice pourra ouvrir sa boîte et consulter le message. Ici la clé publique est symbolisée par la boîte aux lettre, tout le monde peut y déposer un message, la clé qui ouvre la boîte aux lettres est la clé privée d'Alice, que Alice doit conserver à l'abri.



En prenant appui sur l'exemple précédent, si le message initial est 71 et que la fonction f de chiffrement est connue de tous, le message transmis est 11 et le déchiffrement sera rapide si la trappe secrète 7 est connue du destinataire.
Les paramètres d'un protocole de ***chiffrement à clé publique*** sont donc :

- les fonctions de chiffrement et de déchiffrement : $\mathscr{C}$ et $\mathscr{D}$,
- la clé publique du destinataire qui va permettre de paramétrer la fonction $\mathscr{C}$,
- la clé privée du destinataire qui va permettre de paramétrer la fonction $\mathscr{D}$.

Dans le cadre de notre exemple Bruno souhaite envoyer un message à Alice, ces éléments sont :

- $\mathscr{C} : x \longmapsto x^? \pmod{100}$ et $\mathscr{D} : x \longmapsto x^? \pmod{100}$,
- **3** : la clé publique d'Alice qui permet de définir complètement la fonction de chiffrement :

$$\mathscr{C} : x \longmapsto x^3 \pmod{100},$$

- **7** : la clé privée d'Alice qui permet de définir complètement la fonction de déchiffrement :

$$\mathscr{D} : x \longmapsto x^7 \pmod{100}.$$

Dans la pratique, un chiffrement à clé publique nécessite plus de calculs et est donc assez lent, plus lent qu'un chiffrement à clé privée. Afin de gagner en rapidité, un protocole hybride peut être mis en place de la façon suivante :

- à l'aide d'un protocole de chiffrement à clé publique, Alice et Bruno échangent une clé,
- Alice et Bruno utilise cette clé dans un protocole de chiffrement à clé secrète.

5. L'arithmétique pour RSA

Pour un entier n, sachant qu'il est le produit de deux nombres premiers, il est difficile de retrouver les facteurs p et q tels que $n = pq$. Le principe du chiffrement RSA, chiffrement à clé publique, repose sur cette difficulté.
Dans cette partie nous mettons en place les outils mathématiques nécessaires pour le calcul des clés publique et privée ainsi que les procédés de chiffrement et déchiffrement RSA.

5.1. Le petit théorème de Fermat amélioré

Nous connaissons le petit théorème de Fermat

Théorème 1 (Petit théorème de Fermat)**.**
Si p est un nombre premier et $a \in \mathbb{Z}$ alors

$$\boxed{a^p \equiv a \pmod{p}}$$

et sa variante :

Corollaire 1.
Si p ne divise pas a alors

$$\boxed{a^{p-1} \equiv 1 \pmod{p}}$$

Nous allons voir une version améliorée de ce théorème dans le cas qui nous intéresse :

Théorème 2 (Petit théorème de Fermat amélioré).
Soient p et q deux nombres premiers distincts et soit $n = pq$. Pour tout $a \in \mathbb{Z}$ tel que $\text{pgcd}(a, n) = 1$ *alors :*

$$\boxed{a^{(p-1)(q-1)} \equiv 1 \pmod{n}}$$

On note $\varphi(n) = (p-1)(q-1)$, la ***fonction d'Euler***. L'hypothèse $\text{pgcd}(a, n) = 1$ équivaut ici à ce que a ne soit divisible ni par p, ni par q. Par exemple pour $p = 5$, $q = 7$, $n = 35$ et $\varphi(n) = 4 \cdot 6 = 24$. Alors pour $a = 1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18, \ldots$ on a bien $a^{24} \equiv 1 \pmod{35}$.

Démonstration. Notons $c = a^{(p-1)(q-1)}$. Calculons c modulo p :

$$c \equiv a^{(p-1)(q-1)} \equiv (a^{(p-1)})^{q-1} \equiv 1^{q-1} \equiv 1 \pmod{p}$$

où l'on appliquer le petit théorème de Fermat : $a^{p-1} \equiv 1 \pmod{p}$, car p ne divise pas a.
Calculons ce même c mais cette fois modulo q :

$$c \equiv a^{(p-1)(q-1)} \equiv (a^{(q-1)})^{p-1} \equiv 1^{p-1} \equiv 1 \pmod{q}$$

où l'on appliquer le petit théorème de Fermat : $a^{q-1} \equiv 1 \pmod{q}$, car q ne divise pas a.
Conclusion partielle : $c \equiv 1 \pmod{p}$ et $c \equiv 1 \pmod{q}$.

Nous allons en déduire que $c \equiv 1 \pmod{pq}$.
Comme $c \equiv 1 \pmod{p}$ alors il existe $\alpha \in \mathbb{Z}$ tel que $c = 1 + \alpha p$; comme $c \equiv 1 \pmod{q}$ alors il existe $\beta \in \mathbb{Z}$ tel que $c = 1 + \beta q$. Donc $c - 1 = \alpha p = \beta q$. De l'égalité $\alpha p = \beta q$, on tire que $p | \beta q$.
Comme p et q sont premiers entre eux (car ce sont des nombres premiers distincts) alors par le lemme de Gauss on en déduit que $p | \beta$. Il existe donc $\beta' \in \mathbb{Z}$ tel que $\beta = \beta' p$.
Ainsi $c = 1 + \beta q = 1 + \beta' pq$. Ce qui fait que $c \equiv 1 \pmod{pq}$, c'est exactement dire $a^{(p-1)(q-1)} \equiv 1 \pmod{n}$. □

5.2. L'algorithme d'Euclide étendu

Nous avons déjà étudié l'algorithme d'Euclide qui repose sur le principe que $\text{pgcd}(a, b) = \text{pgcd}(b, a \pmod{b})$.
Voici sa mise en œuvre informatique.

Code 7 (*euclide.py (1)*).

```
def euclide(a,b):
    while b !=0 :
        a , b = b , a % b
    return a
```

On profite que `Python` assure les affectations simultanées, ce qui pour nous correspond aux suites

$$\begin{cases} a_{i+1} & = & b_i \\ b_{i+1} & \equiv & a_i \pmod{b_i} \end{cases}$$

initialisée par $a_0 = a$, $b_0 = b$.
Nous avons vu aussi comment « remonter » l'algorithme d'Euclide à la main pour obtenir les coefficients de Bézout u, v tels que $au + bv = \text{pgcd}(a, b)$. Cependant il nous faut une méthode plus automatique pour obtenir ces coefficients, c'est l'***algorithme d'Euclide étendu***.
On définit deux suites (x_i), (y_i) qui vont aboutir aux coefficients de Bézout.
L'initialisation est :

$$x_0 = 1 \qquad x_1 = 0 \qquad y_0 = 0 \qquad y_1 = 1$$

et la formule de récurrence pour $i \geqslant 1$:

$$x_{i+1} = x_{i-1} - q_i x_i \qquad y_{i+1} = y_{i-1} - q_i y_i$$

où q_i est le quotient de la division euclidienne de a_i par b_i.

Code 8 (*euclide.py (2)*).

```
def euclide_etendu(a,b):
    x = 1 ; xx = 0
    y = 0 ; yy = 1
```

```
    while b != 0 :
        q = a // b
        a , b = b , a % b
        xx , x = x - q*xx , xx
        yy , y = y - q*yy , yy
    return (a,x,y)
```

Cet algorithme renvoie d'abord le pgcd, puis les coefficients u, v tels que $au + bv = \text{pgcd}(a, b)$.

5.3. Inverse modulo n

Soit $a \in \mathbb{Z}$, on dit que $x \in \mathbb{Z}$ est un ***inverse de a modulo n*** si $\boxed{ax \equiv 1 \pmod{n}.}$
Trouver un inverse de a modulo n est donc un cas particulier de l'équation $ax \equiv b \pmod{n}$.

Proposition 1.
- *a admet un inverse modulo n si et seulement si a et n sont premiers entre eux.*
- *Si $au + nv = 1$ alors u est un inverse de a modulo n.*

En d'autres termes, trouver un inverse de a modulo n revient à calculer les coefficients de Bézout associés à la paire (a, n).

Démonstration. La preuve est essentiellement une reformulation du théorème de Bézout :

$$\begin{aligned} \text{pgcd}(a,n) = 1 &\iff \exists u, v \in \mathbb{Z} \quad au + nv = 1 \\ &\iff \exists u \in \mathbb{Z} \quad au \equiv 1 \pmod{n} \end{aligned}$$

□

Voici le code :

Code 9 (*euclide.py (3)*).
```
def inverse(a,n):
    c,u,v = euclide_etendu(a,n)     # pgcd et coeff. de Bézout
    if c != 1 :             # Si pgcd différent de 1 renvoie 0
        return 0
    else :
        return u % n                        # Renvoie l'inverse
```

5.4. L'exponentiation rapide

Nous aurons besoin de calculer rapidement des puissances modulo n. Pour cela il existe une méthode beaucoup plus efficace que de calculer d'abord a^k puis de le réduire modulo n. Il faut garder à l'esprit que les entiers que l'on va manipuler ont des dizaines voire des centaines de chiffres.
Voyons la technique sur l'exemple de $5^{11} \pmod{14}$. L'idée est de seulement calculer $5, 5^2, 5^4, 5^8 \ldots$ et de réduire modulo n à chaque fois. Pour cela on remarque que $11 = 8 + 2 + 1$ donc

$$5^{11} = 5^8 \times 5^2 \times 5^1.$$

Calculons donc les $5^{2^i} \pmod{14}$:

$$\begin{aligned} 5 &\equiv 5 \pmod{14} \\ 5^2 &\equiv 25 \equiv 11 \pmod{14} \\ 5^4 &\equiv 5^2 \times 5^2 \equiv 11 \times 11 \equiv 121 \equiv 9 \pmod{14} \\ 5^8 &\equiv 5^4 \times 5^4 \equiv 9 \times 9 \equiv 81 \equiv 11 \pmod{14} \end{aligned}$$

à chaque étape est effectuée une multiplication modulaire. Conséquence :

$$5^{11} \equiv 5^8 \times 5^2 \times 5^1 \equiv 11 \times 11 \times 5 \equiv 11 \times 55 \equiv 11 \times 13 \equiv 143 \equiv 3 \pmod{14}.$$

Nous obtenons donc un calcul de $5^{11} \pmod{14}$ en 5 opérations au lieu de 10 si on avait fait $5 \times 5 \times 5 \cdots$.

Voici une formulation générale de la méthode. On écrit le développement de l'exposant k en base 2 : $(k_\ell, \dots, k_2, k_1, k_0)$ avec $k_i \in \{0,1\}$ de sorte que

$$k = \sum_{i=0}^{\ell} k_i 2^i.$$

On obtient alors

$$x^k = x^{\sum_{i=0}^{\ell} k_i 2^i} = \prod_{i=0}^{\ell} (x^{2^i})^{k_i}.$$

Par exemple 11 en base 2 s'écrit $(1,0,1,1)$, donc, comme on l'a vu :

$$5^{11} = (5^{2^3})^1 \times (5^{2^2})^0 \times (5^{2^1})^1 \times (5^{2^0})^1.$$

Voici un autre exemple : calculons $17^{154} \pmod{100}$. Tout d'abord on décompose l'exposant $k = 154$ en base 2 : $154 = 128 + 16 + 8 + 2 = 2^7 + 2^4 + 2^3 + 2^1$, il s'écrit donc en base 2 : $(1,0,0,1,1,0,1,0)$.
Ensuite on calcule $17, 17^2, 17^4, 17^8, \dots, 17^{128}$ modulo 100.

$$\begin{aligned}
17 &\equiv 17 \pmod{100}\\
17^2 &\equiv 17 \times 17 \equiv 289 \equiv 89 \pmod{100}\\
17^4 &\equiv 17^2 \times 17^2 \equiv 89 \times 89 \equiv 7921 \equiv 21 \pmod{100}\\
17^8 &\equiv 17^4 \times 17^4 \equiv 21 \times 21 \equiv 441 \equiv 41 \pmod{100}\\
17^{16} &\equiv 17^8 \times 17^8 \equiv 41 \times 41 \equiv 1681 \equiv 81 \pmod{100}\\
17^{32} &\equiv 17^{16} \times 17^{16} \equiv 81 \times 81 \equiv 6561 \equiv 61 \pmod{100}\\
17^{64} &\equiv 17^{32} \times 17^{32} \equiv 61 \times 61 \equiv 3721 \equiv 21 \pmod{100}\\
17^{128} &\equiv 17^{64} \times 17^{64} \equiv 21 \times 21 \equiv 441 \equiv 41 \pmod{100}
\end{aligned}$$

Il ne reste qu'à rassembler :

$$17^{154} \equiv 17^{128} \times 17^{16} \times 17^8 \times 17^2 \equiv 41 \times 81 \times 41 \times 89 \equiv 3321 \times 3649 \equiv 21 \times 49 \equiv 1029 \equiv 29 \pmod{100}$$

On en déduit un algorithme pour le calcul rapide des puissances.

Code 10 (*puissance.py*).

```
def puissance(x,k,n):
    puiss = 1                       # Résultat
    while (k>0):
        if k % 2 != 0 :             # Si k est impair (i.e. k_i=1)
            puiss = (puiss*x) % n
        x = x*x % n                 # Vaut x, x^2, x^4,...
        k = k // 2
    return(puiss)
```

En fait Python sait faire l'exponentiation rapide : `pow(x,k,n)` pour le calcul de a^k modulo n, il faut donc éviter `(x ** k) % n` qui n'est pas adapté.

6. Le chiffrement RSA

Voici le but ultime de ce cours : la chiffrement RSA. Il est temps de relire l'introduction du chapitre « Arithmétique » pour s'apercevoir que nous sommes prêts !

Pour crypter un message on commence par le transformer en un –ou plusieurs– nombres. Les processus de chiffrement et déchiffrement font appel à plusieurs notions :

- *On choisit deux* **nombres premiers** *p et q que l'on garde secrets et on pose $n = p \times q$. Le principe étant que même connaissant n il est très difficile de retrouver p et q (qui sont des nombres ayant des centaines de chiffres).*
- *La clé secrète et la clé publique se calculent à l'aide de l'***algorithme d'Euclide** *et des* **coefficients de Bézout**.
- *Les calculs de cryptage se feront* **modulo** *n.*
- *Le déchiffrement fonctionne grâce à une variante du* **petit théorème de Fermat**.

Dans cette section, c'est Bruno qui veut envoyer un message secret à Alice. La processus se décompose ainsi :

1. Alice prépare une clé publique et une clé privée,
2. Bruno utilise la clé publique d'Alice pour crypter son message,
3. Alice reçoit le message crypté et le déchiffre grâce à sa clé privée.

6.1. Calcul de la clé publique et de la clé privée

Choix de deux nombres premiers

Alice effectue, une fois pour toute, les opérations suivantes (en secret) :

- elle choisit deux nombres premiers distincts p et q (dans la pratique ce sont de très grand nombres, jusqu'à des centaines de chiffres),
- Elle calcule $n = p \times q$,
- Elle calcule $\varphi(n) = (p-1) \times (q-1)$.

Exemple 1.

- $p = 5$ et $q = 17$
- $n = p \times q = 85$
- $\varphi(n) = (p-1) \times (q-1) = 64$

Vous noterez que le calcul de $\varphi(n)$ n'est possible que si la décomposition de n sous la forme $p \times q$ est connue. D'où le caractère secret de $\varphi(n)$ même si n est connu de tous.

Exemple 2.

- $p = 101$ et $q = 103$
- $n = p \times q = 10\,403$
- $\varphi(n) = (p-1) \times (q-1) = 10\,200$

Choix d'un exposant et calcul de son inverse

Alice continue :

- elle choisit un exposant e tel que $\text{pgcd}(e, \varphi(n)) = 1$,
- elle calcule l'inverse d de e module $\varphi(n)$: $d \times e \equiv 1 \pmod{\varphi(n)}$. Ce calcul se fait par l'algorithme d'Euclide étendu.

Exemple 1.

- Alice choisit par exemple $e = 5$ et on a bien $\text{pgcd}(e, \varphi(n)) = \text{pgcd}(5, 64) = 1$,
- Alice applique l'algorithme d'Euclide étendu pour calculer les coefficients de Bézout correspondant à $\text{pgcd}(e, \varphi(n)) = 1$. Elle trouve $5 \times 13 + 64 \times (-1) = 1$. Donc $5 \times 13 \equiv 1 \pmod{64}$ et l'inverse de e modulo $\varphi(n)$ est $d = 13$.

Exemple 2.

- Alice choisit par exemple $e = 7$ et on a bien $\text{pgcd}(e, \varphi(n)) = \text{pgcd}(7, 10\,200) = 1$,
- L'algorithme d'Euclide étendu pour $\text{pgcd}(e, \varphi(n)) = 1$ donne $7 \times (-1457) + 10\,200 \times 1 = 1$. Mais $-1457 \equiv 8743 \pmod{\varphi(n)}$, donc pour $d = 8743$ on a $d \times e \equiv 1 \pmod{\varphi(n)}$.

Clé publique

La ***clé publique*** d'Alice est constituée des deux nombres :

$$\boxed{n \text{ et } e}$$

Et comme son nom l'indique Alice communique sa clé publique au monde entier.

Exemple 1. $n = 85$ et $e = 5$

Exemple 2. $n = 10\,403$ et $e = 7$

Clé privée

Alice garde pour elle sa ***clé privée*** :

$$\boxed{d}$$

Alice détruit en secret p, q et $\varphi(n)$ qui ne sont plus utiles. Elle conserve secrètement sa clé privée.

Exemple 1. $d = 13$

Exemple 2. $d = 8743$

6.2. Chiffrement du message

Bruno veut envoyer un message secret à Alice. Il se débrouille pour que son message soit un entier (quitte à découper son texte en bloc et à transformer chaque bloc en un entier).

Message

Le message est un entier m, tel que $0 \leqslant m < n$.

Exemple 1. Bruno veut envoyer le message $m = 10$.

Exemple 2. Bruno veut envoyer le message $m = 1234$.

Message chiffré

Bruno récupère la clé publique d'Alice : n et e avec laquelle il calcule, à l'aide de l'algorithme d'exponentiation rapide, le message chiffré :

$$\boxed{x \equiv m^e \pmod{n}}$$

Il transmet ce message x à Alice

Exemple 1. $m = 10$, $n = 85$ et $e = 5$ donc

$$x \equiv m^e \pmod{n} \equiv 10^5 \pmod{85}$$

On peut ici faire les calculs à la main :

$$\begin{aligned} 10^2 &\equiv 100 \equiv 15 \pmod{85} \\ 10^4 &\equiv (10^2)^2 \equiv 15^2 \equiv 225 \equiv 55 \pmod{85} \\ x &\equiv 10^5 \equiv 10^4 \times 10 \equiv 55 \times 10 \equiv 550 \equiv 40 \pmod{85} \end{aligned}$$

Le message chiffré est donc $x = 40$.

Exemple 2. $m = 1234$, $n = 10\,403$ et $e = 7$ donc

$$x \equiv m^e \pmod{n} \equiv 1234^7 \pmod{10\,403}$$

On utilise l'ordinateur pour obtenir que $x = 10\,378$.

6.3. Déchiffrement du message

Alice reçoit le message x chiffré par Bruno, elle le décrypte à l'aide de sa clé privée d, par l'opération :

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

qui utilise également l'algorithme d'exponentiation rapide.
Nous allons prouver dans le lemme 1, que par cette opération Alice retrouve bien le message original m de Bruno.

Exemple 1. $c = 40$, $d = 13$, $n = 85$ donc

$$x^d \equiv (40)^{13} \pmod{85}.$$

Calculons à la main $40^{13} \equiv \pmod{85}$ on note que $13 = 8 + 4 + 1$, donc $40^{13} = 40^8 \times 40^4 \times 40$.

$$\begin{aligned} 40^2 &\equiv 1600 \equiv 70 \pmod{85} \\ 40^4 &\equiv (40^2)^2 \equiv 70^2 \equiv 4900 \equiv 55 \pmod{85} \\ 40^8 &\equiv (40^4)^2 \equiv 55^2 \equiv 3025 \equiv 50 \pmod{85} \end{aligned}$$

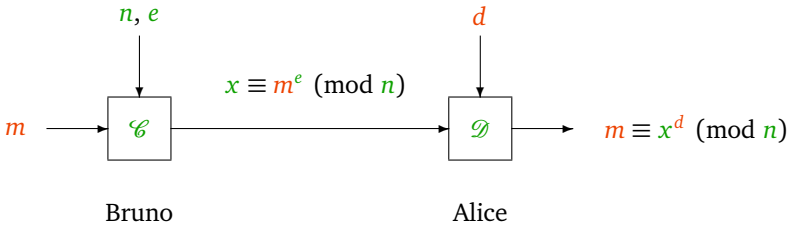
Donc

$$x^d \equiv 40^{13} \equiv 40^8 \times 40^4 \times 40 \equiv 50 \times 55 \times 40 \equiv 10 \pmod{85}$$

qui est bien le message m de Bruno.

Exemple 2. $c = 10\,378$, $d = 8743$, $n = 10\,403$. On calcule par ordinateur $x^d \equiv (10\,378)^{8743} \pmod{10\,403}$ qui vaut exactement le message original de Bruno $m = 1234$.

6.4. Schéma



Clés d'Alice :
- publique : n, e
- privée : d

6.5. Lemme de déchiffrement

Le principe de déchiffrement repose sur le petit théorème de Fermat amélioré.

Lemme 1.
Soit d l'inverse de e modulo $\varphi(n)$.

$$\boxed{\textit{Si } x \equiv m^e \pmod{n} \textit{ alors } m \equiv x^d \pmod{n}.}$$

Ce lemme prouve bien que le message original m de Bruno, chiffré par clé publique d'Alice (e, n) en le message x, peut-être retrouvé par Alice à l'aide de sa clé secrète d.

Démonstration.
- Que d soit l'inverse de e modulo $\varphi(n)$ signifie $d \cdot e \equiv 1 \pmod{\varphi(n)}$. Autrement dit, il existe $k \in \mathbb{Z}$ tel que $d \cdot e = 1 + k \cdot \varphi(n)$.
- On rappelle que par le petit théorème de Fermat généralisé : lorsque m et n sont premiers entre eux

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

- **Premier cas** $\text{pgcd}(m, n) = 1$.
  Notons $c \equiv m^e \pmod{n}$ et calculons x^d :

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

- **Deuxième cas** $\text{pgcd}(m,n) \neq 1$.
  Comme n est le produit des deux nombres premiers p et q et que m est strictement plus petit que n alors si m et n ne sont pas premiers entre eux cela implique que p divise m ou bien q divise m (mais pas les deux en même temps). Faisons l'hypothèse $\text{pgcd}(m,n) = p$ et $\text{pgcd}(m,q) = 1$, le cas $\text{pgcd}(m,n) = q$ et $\text{pgcd}(m,p) = 1$ se traiterait de la même manière.
  Étudions $(m^e)^d$ à la fois modulo p et modulo q à l'image de ce que nous avons fait dans la preuve du théorème de Fermat amélioré.
  — modulo p : $m \equiv 0 \pmod{p}$ et $(m^e)^d \equiv 0 \pmod{p}$ donc $(m^e)^d \equiv m \pmod{p}$,
  — modulo q : $(m^e)^d \equiv m \times (m^{\varphi(n)})^k \equiv m \times (m^{q-1})^{(p-1)k} \equiv m \pmod{q}$.
  Comme p et q sont deux nombres premiers distincts, ils sont premiers entre eux et on peut écrire comme dans la preuve du petit théorème de Fermat amélioré que

$$(m^e)^d \equiv m \pmod{n}$$

□

6.6. Algorithmes

La mise en œuvre est maintenant très simple. Alice choisit deux nombres premiers p et q et un exposant e.
Voici le calcul de la clé secrète :

Code 11 (*rsa.py (1)*).
```
def cle_privee(p,q,e) :
    n = p * q
    phi = (p-1)*(q-1)
    c,d,dd = euclide_etendu(e,phi)          # Pgcd et coeff de Bézout
    return(d % phi)                         # Bon représentant
```

Le chiffrement d'un message m est possible par tout le monde, connaissant la clé publique (n,e).

Code 12 (*rsa.py (2)*).
```
def codage_rsa(m,n,e):
    return pow(m,e,n)
```

Seule Alice peut déchiffrer le message crypté x, à l'aide de sa clé privée d.

Code 13 (*rsa.py (3)*).
```
def decodage_rsa(x,n,d):
    return pow(x,d,n)
```

Pour continuer...

Bibliographie commentée :

1. **Histoire des codes secrets** de Simon Singh, Le livre de Poche.
   Les codes secrets racontés comme un roman policier. Passionnant. Idéal pour les plus littéraires.
2. **Comprendre les codes secrets** de Pierre Vigoureux, édition Ellipses.
   Un petit livre très clair et très bien écrit, qui présente un panorama complet de la cryptographie sans rentrer dans les détails mathématiques. Idéal pour les esprits logiques.
3. **Codage et cryptographie** de Joan Gómez, édition Le Monde – Images des mathématiques. Un autre petit livre très clair et très bien, un peu de maths, des explications claires et des encarts historiques intéressants.
4. **Introduction à la cryptographie** de Johannes Buchmann, édition Dunod.
   Un livre d'un niveau avancé (troisième année de licence) pour comprendre les méthodes mathématiques de la cryptographie moderne. Idéal pour unifier les points de vue des mathématiques avec l'informatique.

5. **Algèbre - Première année** de Liret et Martinais, édition Dunod.
Livre qui recouvre tout le programme d'algèbre de la première année, très bien adapté aux étudiants des l'université. Pas de cryptographie.

Auteurs du chapitre Arnaud Bodin, François Recher