

↔ **Introduction**

• **Cryptographie, cryptanalyse, cryptologie**

La **cryptographie** étudie les principes, méthodes et techniques mathématiques liées aux aspects de sécurité de l'information et répond aux besoins suivants :

- intégrité des données transmises : vérifier que les données n'ont pas été altérées (que ce soit frauduleusement ou accidentellement) ;
- contrôle d'accès : authentifier les utilisateurs afin de limiter l'accès aux données ;
- confidentialité : rendre les données incompréhensibles aux personnes non autorisées ;
- identification : authentifier l'émetteur et le destinataire des données ;
- non répudiation : ne pas permettre à l'émetteur de nier qu'il a effectivement émis les données ni au destinataire qu'il les a effectivement reçues.

Elle vise donc à fournir des techniques permettant l'échange d'informations entre un émetteur et un destinataire dans un environnement a priori non sûr (par exemple Internet) sans que des intrus puissent s'immiscer dans cet échange, que ce soit pour y lire des informations, soit pour en modifier ou en transmettre.

En résumé, la cryptographie est donc l'art ou la science de garder secret les messages.

La **cryptanalyse** s'intéresse à la sécurité des procédés de chiffrement élaborés dans le cadre de la cryptographie. On essaie d'y casser les fonctions cryptographiques par des méthodes autres que la force brute qui consiste, pour un algorithme donné, à essayer systématiquement toutes les clés. Il s'agit donc là de rechercher les faiblesses mathématiques de l'algorithme.

La **cryptologie** recouvre les domaines de la cryptographie et de la cryptanalyse.

• **Terminologie**

Dans le jargon cryptographique, étant donné un message (*plain-text* ou *clear-text*), l'opération de **chiffrement** (ou **cryptage**) consiste à produire un message (*cipher-text*/chiffre-texte) cryptant ce message original et visant à le rendre illisible.

L'opération inverse consistant à reconstruire le message original à partir du message chiffré est le **déchiffrement** (ou **décryptage**).

Un chiffre est constitué d'une méthode de cryptage et d'une méthode de décryptage. Si quelques algorithmes reposent sur le secret des algorithmes, les algorithmes modernes illustrent le principe de Kerckhoffs : la sécurité d'un système de chiffrement n'est pas fondée sur le secret de la procédure qu'il utilise mais sur le paramètre qu'il utilise dans sa mise en oeuvre et qui en est la **clé**.

Pour illustrer la différence des deux approches, prenons l'exemple de l'algorithme de chiffrement **ROT13** (méthode de César) pour un alphabet de 26 caractères (typiquement les lettres A-Z) qui décale simplement les lettres d'un texte de 13 positions. La lettre A est chiffrée en N qui est elle-même chiffrée par A :

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ  
NOPQRSTUVWXYZABCDEFGHIJKLM
```

Chaque lettre est invariablement remplacée par la même : il s'agit d'un chiffrement par *substitution monoalphabétique*. Si deux entités décident de communiquer en chiffrant les messages par cette méthode, le secret entre ces deux entités est que le chiffrage est réalisé au moyen de cet algorithme.

Avec la seconde approche, l'algorithme **ROT13** n'est qu'un cas particulier d'un algorithme général paramétré par une permutation de l'alphabet utilisé. La technique d'encryptage consiste alors à substituer à un caractère du message son image par la permutation. La technique est connue mais le secret détenu par les deux entités communiquant entre elles est la permutation : c'est cette permutation qui constitue la clé.

• **Cryptage symétrique/cryptage asymétrique**

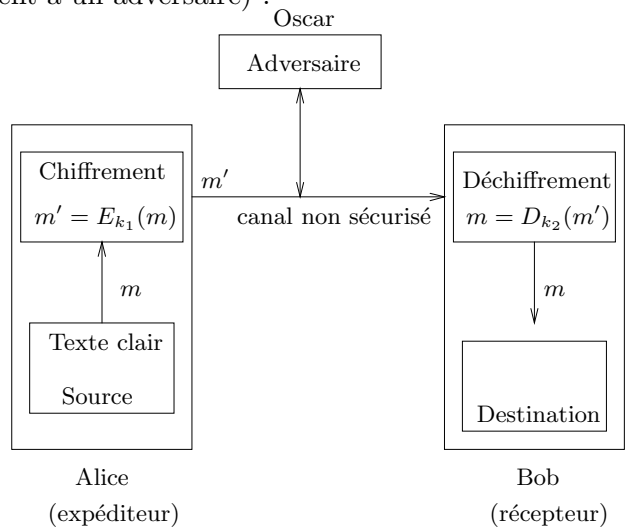
- Dans les **algorithmes symétriques**, la même clé (**clé secrète**) est utilisée pour le chiffrement et le déchiffrement (ou la clé de déchiffrement est différente mais est calculable facilement à partir de la clé de chiffrement). La clé doit donc être échangée et connue des deux entités.

Le standard de chiffrement **D.E.S.** (*Data Encryption Standard*, 1975) en est un exemple. La clé y est constituée de 64 bits, donc une succession de 64 chiffres 0 ou 1.

Nous verrons plus loin des exemples de transformations permettant la construction du message chiffré à partir du message en clair et de la clé.

- Dans les **algorithmes asymétriques**, les clés de chiffrement et de déchiffrement sont différentes. Une **clé publique** est utilisée pour le chiffrement : cette clé est largement diffusée. Par contre le déchiffrement est réalisé au moyen d'une **cle privée** que seul le destinataire est supposé connaître. Un exemple en est le standard de chiffrement **R.S.A.** (du nom de leurs auteurs, *Rivest, Shamir, Adleman*, 1977).

Le schéma suivant résume ce principe général et Alice, Bob et Oscar y désigne de manière symbolique un émetteur, un destinataire et un adversaire (essayant de déjouer la sécurité dans l'échange entre Alice et Bob), E_{k_1} correspond à l'algorithme d'encryptage (ou de chiffrement) avec la clé k_1 et D_{k_2} désigne l'algorithme de déchiffrement avec la clé k_2 ($k_1 = k_2$ dans un algorithme symétrique). Un canal sécurisé est un canal où un adversaire n'a pas la possibilité de lire, de modifier ou d'effacer (il est sécuritaire s'il est inaccessible physiquement à un adversaire) :



↔ **Exemples introductifs**

Imaginons qu'on veuille coder un message constitué de lettres majuscules, d'espaces (représentés symboliquement par_) et de ' (l'apostrophe).

Une première technique triviale pour chiffrer un message consiste à réaliser une permutation des caractères le constituant, comme par exemple :

```
ABCDEFGHIJKLMNQRSTUWXYZ_'
INFO'PWNRCBUKQVZJKEGL_MATHS
```

Le chiffage du texte

```
LES_SANGLOTS_LONGS_DES_VIOLONS_DE_L'AUTOMNE_BERCENT_MON_COEUR_D'UNE_LANGUEUR_MONOTONE
```

conduit au message chiffré suivant

```
U'KHKIDWUQEKHUQDWKHO'KHLRQUQDKHN'JF'DEHKQDHFQ'GJHOSGD'HUIDWG'GJHKQDQEQD'
```

Le programme suivant réalise ce chiffrement : lors de l'appel, le premier paramètre correspond à l'alphabet utilisé, le deuxième définit la permutation et le troisième donne le message à coder. Aucun contrôle n'a été fait pour s'assurer qu'on a bien une permutation et que le message ne contient que des caractères de l'alphabet.

```

--> cat Chiffre1.java
class Chiffre1 {
    public static void main(String[] args) {
        int i, pos;
        if(args.length < 3) {
            System.err.println("Erreur parametres");
            System.exit(0); }
        StringBuffer message = new StringBuffer();
        for(i = 0; i < args[2].length(); i++) {
            pos = args[0].indexOf(args[2].charAt(i));
            message = message.append(args[1].charAt(pos));
        }
        System.out.println(message);
    }
}
--> java Chiffre1 "ABCDEFGHIJKLMNOPQRSTUVWXYZ_" "INFO'PWNRCBUKQVZJKEGL_MATHS" "LES_SANGLOTS_LONGS_
DES_VIOLONS_DE_L'AUTOMNE_BERCENT_MON_COEUR_D'UNE_LANGUEUR_MONOTONE"
U'KHKIDWUQEKHUQDWKHO'KHLRQUQDKHN'JF'DEHKQDHFQ'GJHOSGD'HUIDWG'GJHKQDQEQD'
-->

```

Avec un tel chiffrement, une lettre est cryptée de la même manière partout dans le message chiffré, ce qui facilite le cassage en se basant sur l'analyse des fréquences classiques des lettres dans les différentes langues.

Une amélioration décisive de ce type de chiffrement a été proposée par Vigenère au 16^{ème} siècle. Le chiffrement consiste à réaliser un décalage d'un certain nombre de positions pour les caractères successifs constituant le message. Cela présente l'avantage qu'une même lettre n'est pas chiffrée de manière uniforme dans tout le texte. La clé est constituée par la donnée des décalages réalisés. Si la clé n'a pas la longueur du texte, on peut par exemple envisager que la séquence soit utilisée circulairement.

Ainsi, avec la clé

```
5 7 2 4 7 9 1 3 8 9 2 4 10 1 3
```

le premier caractère (et de manière générale le $(15 * n + 1)$ -ème) est décalé de 5 positions : par exemple un A devient ainsi un F. Le second caractère (de manière générale le $(15 * n + 2)$ -ème) est décalé de 7 positions : un A devient ainsi un H. Et ainsi de suite.

Pour l'extrait du texte de Verlaine, cela conduit au message chiffré suivant :

```
QLUCZJOJTXVWIMRSNUCKNTBRRQPYOVDKGCISIBX'XORO'EJYEIUA'PWWAGYFXWFFD'WFBTJPKCFXWFOUSUXURVN
```

Cette technique de chiffrement a résisté près de 3 siècles à la cryptanalyse. Il n'a été cassé que vers 1850 par Babbage (parallèlement avec Kasiski).

On pourra consulter <http://www.apprendre-en-ligne.net/crypto/vigenere/decodevig.html>.

Un tel chiffrement peut être réalisé par exemple avec le programme suivant (le programme est appelé avec deux paramètres, le premier étant l'alphabet et le second le message à chiffrer) :

```

--> cat Chiffre2.java
class Chiffre2 {
    public static void main(String[] args) {
        int i, // pour parcourir le message
            pos, // position du caractère dans l'alphabet
            decal = 0, // pour savoir de combien de positions on décale;
        int[] decalage = {5, 7, 2, 4, 7, 9, 1, 3, 8, 9, 2, 4, 10, 1, 3};
        if(args.length < 2) {
            System.err.println("Erreur parametres");
            System.exit(0);
        }
        // pour construire le message crypté
        StringBuffer message = new StringBuffer();
        for(i = 0; i < args[1].length(); i++) {
            // position dans l'alphabet du caractère courant du message en clair
            pos = args[0].indexOf(args[1].charAt(i));

```

```

    // position du prochain caractère à ajouter au message chiffré
    pos = (pos + decalage[decal]) % args[0].length();
    // on rajoute le caractère au message chiffré
    message = message.append(args[0].charAt(pos));
    // on avance circulairement dans decalage
    decal = (decal + 1) % decalage.length;
}
System.out.println(message);
}
}
--> java Chiffre2 "ABCDEFGHJKLMNOPQRSTUVWXYZ_" "LES_SANGLOTS_LONGS_DES_VIOLONS_DE_L'AUTOMNE_BERCENT_
MON_COEUR_D'UNE_LANGUEUR_MONOTONE"
QLUCZJOJTXVWIMRSNUCKNTBBRQPYOVDKGCSIBX'XORO'EJYEIUA'PWWAGYFXWFFD'WFBTJPKCFXWFOUXURVN
-->

```

↔ Chiffrement par flots ou par blocs

• Chiffrement par flot

Dans un tel chiffrement, le message est découpé en blocs réguliers de petite taille : il s'agit d'un bit ou d'un octet. Un exemple en est le chiffrement de Vigenère que nous avons présenté.

Le cryptage est réalisé au moyen d'une clé qui, pour garantir la sûreté, doit satisfaire les conditions suivantes (satisfaites en particulier par le système de chiffrement élaboré par Vernam pour la communication via le téléphone rouge entre la Maison Blanche et le Kremlin) :

- la clé doit être de taille égale au nombre de blocs (bits ou octets) à crypter. Ainsi, si le message M est composé des n blocs M_1, M_2, \dots, M_n ($M = M_1M_2\dots M_n$), la clé K doit être elle-même constituée de n blocs, c'est-à-dire s'écrit $K = k_1k_2\dots k_n$;
- les composants de la clé (les k_i) doivent être choisis de façon totalement aléatoire ;
- la clé ne doit servir qu'une seule fois.

Le chiffrement d'un message M avec la clé K est réalisé en appliquant composant par composant, c'est-à-dire entre m_i et k_i , une loi $*$, ce qui conduit à sa forme chiffrée $m_i * c_i$.

Il est important que la loi $*$ soit une loi de groupe, c'est-à-dire satisfasse la propriété qu'étant donné un couple (a, b) , il existe un et un seul x tel que $a = b * x$.

Un exemple simple de fonction utilisée pour le cryptage est, pour un bloc d'un seul bit, de réaliser un *OU exclusif* (*XOR*) entre chacun des bits du message et le bit correspondant de la clé (rappelons que cette opération souvent notée \oplus est telle que $0 \oplus 0 = 1 \oplus 1 = 0$ et $0 \oplus 1 = 1 \oplus 0 = 1$).

Ainsi, si le message à transmettre est 1100011, une clé aléatoire de même longueur sera choisie, par exemple 0100110, ce qui conduira au chiffrement

$$1100011 \oplus 0100110 = 1000101$$

A la réception du message, le déchiffrement est réalisé par l'opération inverse du *XOR*, c'est-à-dire le *XOR* lui-même :

$$1000101 \oplus 0100110 = 1100011$$

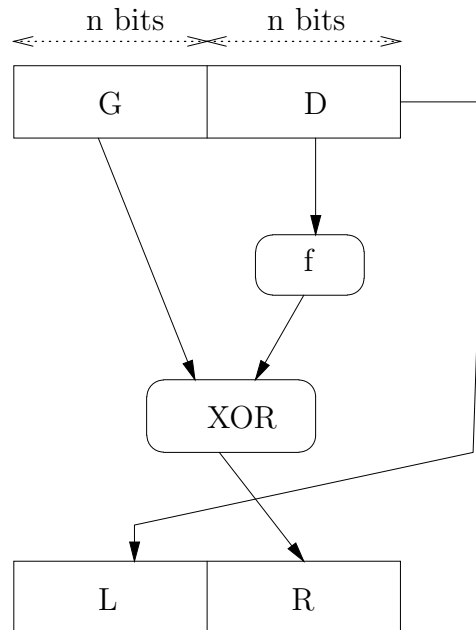
Ce système apporte une grande sécurité mais nécessite d'une part de générer des clés de taille gigantesque, ce qui nécessite une grande puissance de calcul et pose par ailleurs le problème de la transmission de ces clés. Cela a entraîné l'élaboration de solutions alternatives.

• Chiffrement par blocs

Plutôt que d'utiliser une clé immense à usage unique de la taille du message à chiffrer, on va utiliser une clé plus petite et les algorithmes utiliseront cette clé de manière apparemment complexe.

Après avoir été numérisé (par exemple en utilisant les codes ASCII des caractères le composant ou tout autre codage conduisant à une suite de 0 et de 1), le message est découpé en blocs de longueur donnée, généralement une puissance de 2 comprise entre 32 et 512 bits. Ce qui différencie ce type de chiffrement du chiffrement par flot est la taille du bloc. Chaque bloc va être codé séparément au moyen d'une fonction bijective produisant un code paraissant aléatoire (le chiffrement d'un bloc peut être réalisé dans différents modes, mais cela dépasse le stade introductif auquel nous nous plaçons).

Nous allons simplement présenter le principe général proposé dans les années 1950 par Feistel :



La fonction f est supposée transformer une chaîne de n bits en une autre de même longueur. L'algorithme de chiffrement va traiter des blocs de $2n$ bits selon le schéma précédent.

Le bloc est donc découpé en une partie gauche G et une partie droite D de n bits.

L'image du bloc (G, D) est le bloc (L, R) dans lequel $L = D$ et $R = G \oplus f(D)$.

Cette transformation est bijective. A partir du couple (L, R) , il est facile de retrouver le couple (G, D) : $D = L$ et $G = R \oplus f(L)$ ($R \oplus f(L) = G \oplus f(D) \oplus f(L) = G \oplus f(L) \oplus f(L) = G \oplus 0 = G$).

La partie droite D du message initial n'a pas été transformée, elle a juste été déplacée.

D'où l'idée d'itérer le processus un certain nombre de fois (tours ou rondes).

↔ **L'algorithme de chiffrement DES** (*Data Encryption Standard*)

Il a été élaboré chez IBM puis largement adopté à partir de 1976 et a été le plus utilisé durant les années 1980 et 1990.

Il s'agit d'un algorithme de chiffrement symétrique utilisant une clé secrète de 64 bits dans laquelle les bits 8, 16, 24, 32, 40, 48, 56 et 64 sont des bits de parité. Le bit 8 sert ainsi à assurer que le nombre de bits valant 1 dans les bits 1 à 8 est impair : si les bits 1 à 7 ont comme valeur 1001100, le huitième bit aura comme valeur 0.

L'algorithme consiste en un réseau de Feistel à 16 tours : le message à chiffrer est découpé en blocs de 64 bits, chacun d'eux étant divisé en sous-blocs de 32 bits. Il a été cassé en 1998 et a été remplacé en 2000 par l'AES (*Advanced Encryption Standard*) dont les grandes lignes sont :

- algorithme de type symétrique comme le DES ;
- chiffrement par blocs (également comme le DES) ;
- différentes combinaisons de couples longueur de clé/longueur de bloc (128-128, 192-128 ou 256-128 bits) ;
- l'algorithme crypte des blocs de 128 bits de données et utilise pour cela :
 - (a) une fonction non linéaire de substitution d'octet à partir d'une table de substitution ;
 - (b) une opération de décalages sur des lignes ;
 - (c) un brouillage de colonnes ;
 - (d) une addition par XOR ;
 - (e) 10, 12 ou 14 tours selon que la clé est de taille 128, 192 ou 256.

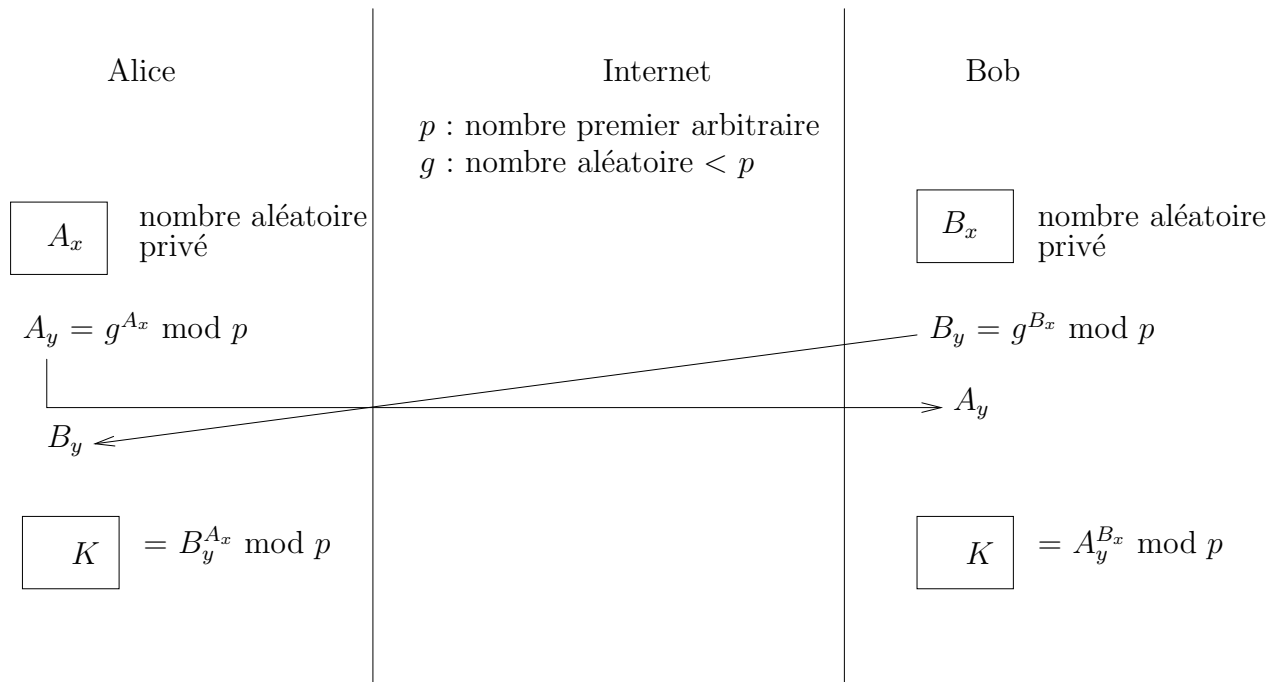
↔ **L'algorithme de Diffie et Hellman**

L'objectif de cet algorithme est de fournir à deux entités Alice et Bob un moyen de se mettre d'accord sur un nombre utilisable comme clé de chiffrement sans qu'une troisième entité puisse découvrir ce nombre en à partir des messages échangés par Alice et Bob.

• Principe général

Un nombre premier p arbitraire est choisi ainsi qu'un entier aléatoire g inférieur à p : ces deux nombres seront publics et donc connus a priori de tous.

Le schéma suivant décrit le protocole de construction du secret K partagé entre Alice et Bob :



Comme on le voit, Alice calcule en fait $(g^{A_x})^{B_x}$ et Bob calcule $(g^{B_x})^{A_x}$. Les opérations sont réalisées modulo le nombre premier p , c'est-à-dire sur l'ensemble de nombres $Z_p = \{0, 1, \dots, p-1\}$. Cet ensemble a la structure de corps fini et on a $(g^a)^b = (g^b)^a$.

Alice et Bob calculent donc le même secret K .

La sécurité du mécanisme repose sur la difficulté (à l'heure actuelle) de calculer, dans un corps fini, le logarithme discret, c'est-à-dire d'y inverser l'exponentiation et donc d'y déterminer la valeur de a à partir de celle de g^a .

Cependant le point faible du mécanisme est qu'il ne permet pas l'authentification des deux entités Bob et Alice : les échanges ne portent pas de signature. Il est donc possible à un adversaire d'intercepter les valeurs échangées entre Alice et Bob et leur substituer des valeurs qu'il aura calculées.

• Un exemple

Un exemple d'utilisation du protocole de Diffie-Hellman est donné avec de petits nombres. Il va de soi qu'une utilisation normale s'appuie sur des nombres plus grands (typiquement un nombre premier de plus de 200 chiffres et des nombres A_x et B_x d'au moins 100 chiffres).

- Alice choisit le nombre premier $p = 317$ et le nombre $g = 7$. Ces deux nombres sont transmis à Bob ;
- Alice choisit le nombre $A_x = 11$ et calcule $A_y = g^{A_x} \bmod p$, c'est-à-dire $7^{11} \bmod 317$.

n	1	2	3	4	5	6	7	8	9	10	11
$7^n \bmod 317$	7	49	26	182	6	42	294	156	141	36	252

Alice obtient donc la valeur $A_y = 252$ qu'elle transmet à Bob ;

- Bob choisit le nombre $B_x = 14$ et calcule $B_y = g^{B_x} \bmod p$, c'est-à-dire $7^{14} \bmod 317$.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$7^n \bmod 317$	7	49	26	182	6	42	294	156	141	36	252	179	302	212

Bob obtient la valeur $B_y = 212$ qu'il transmet à Alice ;

- Alice calcule la valeur $K = 212^{11} \bmod 317$:

n	1	2	3	4	5	6	7	8	9	10	11
$212^n \bmod 317$	212	247	59	145	308	311	313	103	280	81	54

Alice obtient la valeur 54 ;

- o Bob calcule la valeur $K = 252^{14} \bmod 317$:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$252^n \bmod 317$	252	104	214	38	66	148	207	176	289	235	258	31	204	54

Bob obtient également la valeur 54.

↪ Cryptographie à clé publique : RSA

• Fondements et bases mathématiques

La sécurité de ce système de chiffrement à clé publique, proposé en 1977 par Rivest, Shamir et Adleman repose sur la difficulté de factoriser de grands nombres : il est impossible, étant donné un entier n obtenu comme produit de deux grands nombres premiers (d'au moins 100 chiffres), de retrouver en un temps raisonnable ces deux nombres premiers.

Ainsi, étant donné le nombre de 155 chiffres suivant, le problème posé est de retrouver sa décomposition comme produit de deux nombres premiers :

10941738641570527421809707332204035761200373294544920599091384213147634998428893478471799725789126733249
76257528997818337

c'est-à-dire retrouver les deux nombres

102639592829741105772054196573991675900716567808038066803341933521790711307779

et

1066034883801268454820927220360012878679207958575989291522270608237193062808643

Il fait par ailleurs appel à l'arithmétique modulaire et repose sur le théorème d'Euler : nous rappelons tout d'abord le concept d'*arithmétique modulaire* et donnons, sans démonstrations, les résultats fondamentaux sur lesquels repose le chiffrement RSA :

- o étant donné un nombre entier naturel n , on réalise toutes les opérations modulo n , c'est-à-dire qu'un nombre entier est identifié par le reste de division euclidienne (entière) par n . Cela revient donc à ne considérer que l'ensemble $Z_n = \{0, 1, \dots, n - 1\}$;
- o étant donné un nombre entier naturel n , on note $\phi(n)$ (fonction phi d'Euler ou indicateur d'Euler), le nombre d'entiers compris entre 1 et n qui sont premiers avec n (c'est -à-dire dont le PGCD avec n est égal à 1) ;
- o un nombre premier p est premier avec tous les nombres strictement plus petits que lui et donc p premier $\Rightarrow \phi(p) = p - 1$;
- o p et q premiers $\Rightarrow \phi(p \times q) = \phi(p) \times \phi(q) = (p - 1) \times (q - 1)$. Par exemple $\phi(35) = 24$ car 35 est le produit des deux nombres premiers 5 et 7 et donc $\phi(35) = 4 \times 6$;
- o le théorème d'Euler : n entier naturel et a nombre premier avec $n \Rightarrow a^{\phi(n)} \equiv 1 \pmod n$;
- o si un nombre x est premier avec n , il existe un nombre y tel que $x \times y \equiv 1 \pmod n$. Le nombre y est appelé inverse de x et est noté x^{-1} (ce nombre peut être calculé par l'algorithme d'Euclide étendu présenté en annexe) ;
- o le petit théorème de Fermat est un corollaire du précédent dans le cas où n est premier : n entier naturel premier et $a < n \Rightarrow a^{n-1} \equiv 1 \pmod n$ (ou encore $a^n \equiv a$).

• Le choix des clés

L'entité Alice, qui souhaite recevoir des messages chiffrés, va choisir deux grands nombres entiers premiers (aussi grands que possible, d'au moins 100 chiffres) distincts.

Elle calcule leur produit $n = p \times q$ et choisit un entier e premier avec $(p - 1) \times (q - 1)$.

La clé publique RSA d'Alice, qui sera publiée, par exemple, dans un annuaire, est le couple (n, e) .

Cette clé sera utilisé par Bob lorsqu'il enverra un message à Alice pour crypter ce message.

Les messages cryptés seront déchiffrés par Alice au moyen d'une clé différente calculée à partir des entiers p , q et e qu'elle est la seule à connaître. Les seules informations qui circulent et que peut posséder une autre entité sont en effet n (sans connaître p et q) et e .

Cette clé est un couple (n, d) dans lequel l'entier d est tel que $ed = 1 \pmod{(p - 1) \times (q - 1)}$: un tel entier d existe puisque e est premier avec $(p - 1) \times (q - 1)$ (d est l'inverse e^{-1} de e modulo $(p - 1) \times (q - 1)$).

Autrement dit, $e.d - 1$ est un multiple de $(p - 1) \times (q - 1)$.

L'algorithme d'Euclide (voir en annexe) permet à Alice de construire d à partir des valeurs p , q et e .

A titre d'exemple, travaillons avec deux petits nombres premiers : $p = 37$ et $q = 41$.

On obtient $n = 37 \times 41 = 1517$. i

Choisissons par exemple $e = 91$, nombre qui est premier avec $36 * 40 = 1440$.

La clé publique qui sera publiée et utilisée pour chiffrer les messages est $(1517, 91)$.

En ce qui concerne la clé privée, l'inverse de 91 est 1171. La clé privée associée est donc $(1517, 1171)$

• Le chiffrement

Reprenons le message que nous avons déjà utilisé en exemple :

LES_SANGLOTS_LONGS_DES_VIOLONS_DE_L'AUTOMNE_BERCENT_MON_COEUR_D'UNE_LANGUEUR_MONOTONE

Commençons par le numériser. Nous remplaçons chacun des 85 caractères qui le composent par son code ASCII. Ces codes ASCII ont tous 2 caractères décimaux : 39 pour ' , 65 à 90 pour les lettres A à Z et 95 pour _.

Cela conduit à un message formé de $85 * 2 = 170$ chiffres et on a rajouté un chiffre 0 en tête pour avoir une longueur multiple de 3 (en l'occurrence 171) :

0766983958365787176798483957679787183956869839586737976797883956869957639658584797778699566698267697884
95777978956779698582956839857869957665787185698582957779787984797869

Le message ainsi obtenu va être crypté au moyen de la clé publique $(1517, 91)$: cela signifie en particulier qu'on va donc travailler avec de l'arithmétique modulo 1517 et qu'il faut donc constituer des blocs correspondant à des nombres inférieurs à 1517. On peut crypter des blocs de taille 3 (correspondant à des nombres compris entre 0 et 999 et donc tous inférieurs à 1517). Il ne faut pas crypter des blocs de taille 2 car dans ce cas on ferait simplement un chiffrement par substitutions facilement attaquable par analyse de fréquences.

On obtient la suite de blocs suivants à chiffrer :

076 698 395 836 578 717 679 848 395 767 978 718 395 686 983 958 673 797 679 788 395 686
995 763 965 858 479 777 869 956 669 826 769 788 495 777 978 956 779 698 582 956 839 857
869 957 665 787 185 698 582 957 779 787 984 797 869

Un bloc de valeur x sera crypté par $x^{91} \bmod 1517$ (la clé publique et $(1517, 91)$ est la valeur du modulo, produit des deux entiers premiers choisis).

Ainsi le premier bloc sera crypté $76^{91} \bmod 1517$ sous forme d'un bloc de 4 chiffres (les valeurs possibles sont comprises entre 0 et 1516), soit 0997

Le chiffrement des 57 blocs donnent le message crypté suivant de 228 caractères (les blocs séparés les différents blocs de 4 caractères par un espace mais ces espaces ne figurent pas dans le message crypté) :

0997 0042 1283 0385 0865 0800 1171 0293 1283 0138 0423 0799 1283 1090 0983 0070 1487 0387 1171
0196 1283 1090 0181 1050 1483 1080 1482 0740 0648 0117 1224 1381 0933 0196 1503 0740 0423 0117
0738 0042 0730 0117 0321 0734 0648 0079 0073 0935 1332 0042 0730 0079 0738 0935 0533 0387 0648

Le destinataire du message va procéder à son déchiffrement au moyen de sa clé privée, à savoir 1171. Il va constituer un message numérique en déchiffrant chacun des blocs de 4 chiffres.

Un tel bloc de 4 chiffres de valeur y sera déchiffré par la formule $y^{1171} \bmod 1517$. Ainsi le déchiffrement du premier bloc (de valeur 0997) donnera lieu au calcul de $997^{1171} \bmod 1517$, c'est-à-dire 76, puis le second bloc 42 consuira à $42^{1171} \bmod 1517$ c'est-à-dire 698, soit finalement la suite de valeurs suivantes

76 698 395 836 578 717 679 848 395 767 978 718 395 686 983 958 673 797 679 788 395 686 995 763
965 858 479 777 869 956 669 826 769 788 495 777 978 956 779 698 582 956 839 857 869 957 665 787
185 698 582 957 779 787 984 797 869

qui sous forme de message numérisé sans espaces donne :

7669839583657871767984839576797871839568698395867379767978839568699576396585847977786995666982676978849
5777978956779698582956839857869957665787185698582957779787984797869

Il ne reste plus qu'à reconstituer les codes sur 2 chiffres décimaux :

76[L] 69[E] 83[S] 95[_] 83[S] 65[A] 78[N] 71[G] 76[L] 79[O] 84[T] 83[S] 95[_] 76[L] 79[O] 78[N] 71[G]
83[S] 95[_] 68[D] 69[E] 83[S] 95[_] 86[V] 73[I] 79[O] 76[L] 79[O] 78[N] 83[S] 95[_] 68[D] 69[E] 95[_]
76[L] 39['] 65[A] 85[U] 84[T] 79[O] 77[M] 78[N] 69[E] 95[_] 66[B] 69[E] 82[R] 67[C] 69[E] 78[N] 84[T]
95[_] 77[M] 79[O] 78[N] 95[_] 67[C] 79[O] 69[E] 85[U] 82[R] 95[_] 68[D] 39['] 85[U] 78[N] 69[E] 95[_]
76[L] 65[A] 78[N] 71[G] 85[U] 69[E] 85[U] 82[R] 95[_] 77[M] 79[O] 78[N] 79[O] 84[T] 79[O] 78[N] 69[E]

↔ Signature

• Principe général

Il s'agit de fournir un moyen permettant à un destinataire de vérifier qu'un message provient effectivement de l'expéditeur attendu et non d'un intrus.

Pour ce faire, chacune des deux entités utilisera une clé privée et une clé publique. L'émetteur d'un message constituera une *empreinte* (ou *condensat*) de ce message. Pour construire une telle empreinte on utilise une *fonction de hachage*. L'objectif est d'obtenir un résumé significatif du message à partir duquel elle est construite.

Le mécanisme général correspond à la séquence suivante :

1. l'expéditeur calcule l'empreinte du message ;
2. l'expéditeur chiffre l'empreinte avec sa clé privée ;
3. l'expéditeur ajoute l'empreinte ainsi cryptée au message en clair et chiffre l'ensemble ainsi obtenu (message + empreinte cryptée) avec la clé publique du destinataire ;
4. l'expéditeur envoie le message au destinataire ;
5. le destinataire déchiffre le message avec sa clé privée ;
6. le destinataire déchiffre l'empreinte extraite du message avec la clé publique de l'expéditeur ;
7. le destinataire calcule l'empreinte du texte qu'il a décrypté en utilisant la même fonction que celle qu'a utilisée l'expéditeur du message et compare les deux empreintes qui doivent être identiques.

• Un exemple

Nous allons reprendre le texte déjà utilisé et en constituer une empreinte simple en ne conservant que les 3 premiers caractères et les 3 derniers caractères du message : l'empreinte ainsi obtenue est donc **LESONE**. Nous supposons que l'émetteur en est Bob et qu'il possède :

- la clé publique d'Alice, c'est-à-dire (1517, 91) ;
- une clé privée : (1073, 71). Cette clé a été calculée à partir des nombres premiers 29 et 37, et donc $n = 1073$. Le nombre $e = 71$ (premier avec $1008 = 28 \times 36$) a été choisi, ce qui conduit à la valeur $d = 71$ utilisée dans la clé privée ($71 * 71 \equiv 1 \pmod{1008}$). On se trouve dans un cas où la clé privée et la clé publique sont identiques (ce qui n'est pas souhaitable, mais nous ferons avec) et elles sont égales à (1073, 71).

De son côté, Alice, la destinatrice du message, possède :

- la clé publique de Bob, c'est-à-dire (1073, 71) ;
- sa clé privée, c'est-à-dire (1517, 1171).

La forme numérique de l'empreinte (suite des codes ASCII des caractères) donne (les espaces servent simplement à visualiser les codes) :

76 69 83 79 78 69

que nous regroupons par blocs de 3 chiffres en vue de leur chiffrement (cela donne donc des nombres inférieurs à 1073) :

766 983 797 869

Le chiffrement de l'empreinte par Bob en utilisant sa clé privée donne une suite de 4 nombres écrits avec 4 chiffres :

0713 0844 0827 0405

En vue de chiffrement avec la clé publique d'Alice, des nombres de 3 chiffres (donc inférieurs à 1517) sont constitués :

713 084 408 270 405

Le chiffrement de cette séquence est réalisé avec la clé publique d'Alice est réalisé en même temps que le chiffrement du message (tel qu'il est décrit précédemment et nous n'y revenons pas) :

0713 0121 0371 0973 1112

Alice va recevoir dans le message crypté ce double cryptage de l'empreinte construite par Bob à partir du message originel. Elle le décrypte avec sa clé privée, ce qui lui donne un message en clair et une empreinte encore cryptée :

713 084 408 270 405

Le décryptage avec la clé publique de Bob permet à Alice de reconstituer l'empreinte que Bob avait construite avec la fonction de hachage :

766 983 797 869

ou sous forme de codes de caractères :

76[L] 69[E] 83[S] 79[0] 78[N] 69[E]

Le dernier travail qu'Alice doit réaliser est le calcul de l'empreinte du message qu'elle a obtenu par le décryptage, puis elle doit vérifier que cette empreinte est la même que celle qu'elle a reçue de Bob sous forme doublement cryptée.

↪ **Annexes**

• Algorithme d'Euclide

Il permet le calcul du PGCD (plus grand commun diviseur) de deux nombres par la méthode suivante :

- $\text{pgcd}(n, 0) = n$ pour tout entier $n > 0$;
- si a et b sont des entiers > 0 , $\text{pgcd}(a, b) = \text{pgcd}(b, a \% b)$ où $a \% b$ est le reste de la division euclidienne de a par b .

Ainsi, pour les deux nombres 91 et 1440 que nous avons utilisés pour illustrer le chiffrement RSA, l'algorithme d'Euclide permet de calculer leur PGCD qui est égal à 1 et donc de vérifier qu'ils sont premiers entre eux .

Le déroulement de l'algorithme pour ces deux nombres donne :

- $\text{pgcd}(91, 1440) = \text{pgcd}(1440, 91)$ car $91 = 1440 * 0 + 91$;
- $\text{pgcd}(1440, 91) = \text{pgcd}(91, 75)$ car $1440 = 91 * 15 + 75$;
- $\text{pgcd}(91, 75) = \text{pgcd}(75, 16)$ car $91 = 75 * 1 + 16$;
- $\text{pgcd}(75, 16) = \text{pgcd}(16, 11)$ car $75 = 16 * 4 + 11$;
- $\text{pgcd}(16, 11) = \text{pgcd}(11, 5)$ car $16 = 11 * 1 + 5$;
- $\text{pgcd}(11, 5) = \text{pgcd}(5, 1)$ car $11 = 5 * 2 + 1$;
- $\text{pgcd}(5, 1) = \text{pgcd}(1, 0)$ car $5 = 1 * 5 + 1$;
- $\text{pgcd}(1, 0) = 1$.

Cet algorithme s'écrit sous forme récursive de manière simple en Java :

```
static int pgcd(int a, int b) {  
    if(b == 0) return a;  
    return pgcd(b, a % b);  
}
```

• Théorème de Bezout et algorithme d'Euclide étendu

Le théorème de Bezout assure que le PGCD n de deux entiers a et b est une combinaison linéaire à coefficients entiers de a et b et donc qu'il existe des entiers u et v tels que $d = a \times u + b \times v$.

L'algorithme d'Euclide peut être modifié (algorithme d'Euclide étendu) pour déterminer ces deux coefficients u et v .

Le principe est simple : on exprime tout d'abord a et b comme combinaison linéaire de a et b :

- $a = a \times (1) + b \times (0)$
- $b = a \times (0) + b \times (1)$

Puis on exprime les reste successifs des divisions calculés par l'algorithme d'Euclide, comme combinaison linéaire de a et b . Par exemple de l'égalité $a = b \times q_1 + r_1$, on déduit $r_1 = a - b \times q_1$ et en reportant les expressions de a et b , on trouve une expression linéaire de r_1 en fonction de a et b .

En continuant, on obtient $b = r_1 \times q_2 + r_2$, d'où on tire $r_2 = b - r_1 \times q_2$. En remplaçant b et r_1 par leurs expressions en fonction de a et b on obtiendra une expression de r_2 en fonction de a et b .

En itérant ce processus, on obtiendra une expression du PGCD de a et b sous forme de combinaison linéaire de a et b .

Appliquons ce principe aux deux entiers 1440 et 91 :

- $1440 = 1440 \times (1) + 91 \times (0)$
- $91 = 1440 \times (0) + 91 \times (1)$
- $75 = 1440 \times (1) - 91 \times (15) = 1440 \times (1) + 91 \times (-15)$
- $16 = 91 \times (1) - 75 \times (1) = 91 \times (1) - 1440 \times (1) - 91 \times (-15) = 1440 \times (-1) + 91 \times (16)$
- $11 = 75 \times (1) - 16 \times (4) = 1440 \times (1) + 91 \times (-15) - 1440 \times (-4) - 91 \times (64) = 1440 \times (5) + 91 \times (-79)$
- $5 = 16 \times (1) - 11 \times (1) = 1440 \times (-1) + 91 \times (16) - 1440 \times (5) - 91 \times (-79) = 1440 \times (-6) + 91 \times (95)$
- $1 = 11 \times (1) - 5 \times (2) = 1440 \times (5) + 91 \times (-79) - 1440 \times (-12) - 91 \times (190) = 1440 \times (17) + 91 \times (-269)$

En travaillant modulo 1440, la valeur -269 est égale à 1171 ($1440 - 269$), valeur que nous avons adoptée pour la clé privée associée à la clé publique $(1517, 91)$.

Cette méthode de calcul des coefficients u et v dans l'expression linéaire du PGCS de a et b est implantée dans le code Java suivant :

```
--> cat Bezout.java
class Couple {
    int c1, c2;
    Couple(int a, int b) {c1 = a; c2 = b;}
    // redéfinition de la fonction toString pour imprimer un couple de manière agréable
    public String toString() { return("(" + c1 + "," + c2 + ")"); }
}
class Bezout {
    static Couple bezout(int a, int b) {
        int q, r;
        Couple cpl1 = new Couple (1, 0); Couple cpl2 = new Couple (0, 1); Couple tmp;
        System.out.println(a + " " + b);
        while (b != 0) {
            q = a / b; r = a % b; a = b; b = r;
            tmp = new Couple(cpl1.c1 - q * cpl2.c1, cpl1.c2 - q * cpl2.c2);
            cpl1 = cpl2; cpl2 = tmp;
            System.out.println(cpl1 + " " + cpl2); System.out.println("=====");
            System.out.println(a + " " + b);
        }
        return cpl1;
    }
    public static void main(String[] args){
        System.out.println(bezout(1440, 91));
    }
}
--> java Bezout
1440 91
(0,1) (1,-15)
=====
91 75
(1,-15) (-1,16)
=====
75 16
(-1,16) (5,-79)
=====
16 11
(5,-79) (-6,95)
=====
11 5
(-6,95) (17,-269)
=====
5 1
(17,-269) (-91,1440)
=====
1 0
(17,-269)
-->
```

• Du code Java pour le chiffrement RSA

```
--> cat Cle.java
class Couple {
    int c1, c2;
    Couple(int a, int b) {c1 = a; c2 = b;}
    // redéfinition de la fonction toString pour imprimer un couple de manière agréable
    public String toString() { return("(" + c1 + "," + c2 +")"); }
}

class Cle {
    int valeur;
    int modulo;
    // constructeur d'une clé pour un modulo et une valeur donnée
    Cle(int valeur, int modulo){ this.valeur = valeur; this.modulo = modulo;}
    public String toString() { return("(" + valeur + "," + modulo +")"); }

    static Couple bezout(int a, int b) {
        int q, r;
        Couple cpl1 = new Couple (1, 0); Couple cpl2 = new Couple (0, 1);
        Couple tmp;
        while (b != 0) {
            q = a / b;
            r = a % b;
            a = b;
            b = r;
            tmp = new Couple(cpl1.c1 - q * cpl2.c1, cpl1.c2 - q * cpl2.c2);
            cpl1 = cpl2;
            cpl2 = tmp;
        }
        return cpl1;
    }

    // constructon d'une clé publique et d'une clé privée à partir
    // de deux nombres premiers p et q et un nombre e premier avec (p-1).(q-1)
    static Cle[] creerCles(int p, int q, int e) {
        Cle[] keys = new Cle[2]; // le tableau des deux clés
        keys[0] = new Cle(e, p*q); // la clé publique
        int d = bezout((p-1)*(q-1), e).c2;
        if(d < 0)
            d = (p-1)*(q-1) + d;
        keys[1] = new Cle(d, p*q);
        return keys;
    }
}

--> cat RSA.java
class RSA {
    // impression d'un tableau de bytes
    static void imprimer(byte[] tab){
        for(int i = 0; i <tab.length; i++) System.out.print(tab[i]);
        System.out.println();
    }
}
```

```

// tranformation d'un message (suite de caractères) en une suite de chiffres :
//     les caractères du message sont supposés tous codés sur 2 chiffres décimaux
//     le tableau de chiffres construit doit avoir une taille multiple de n
static byte[] numeriser(String message, int n) {
    int lg = 2 * message.length();
    int pos = 0;
    if(lg % n != 0) {pos = n - lg % n; lg += pos;}
    byte[] tab = new byte[lg];
    for(int i = 0; i < message.length(); i++) {
        tab[pos++] = (byte) (message.charAt(i) / 10);
        tab[pos++] = (byte) (message.charAt(i) % 10);
    }
    return tab;
}
// transforme une suite de bytes contenant des chiffres en une chaîne de caractères
// correspondant aux codes ASCII supposés être de deux chiffres décimaux
static StringBuffer construireMsg(byte[] tab) {
    StringBuffer message = new StringBuffer();
    for(int i = 0; i < tab.length / 2; i++)
        message.append((char)(10 * tab[2 * i] + tab[2 * i + 1]));
    return message;
}
// fonction qui crypte le nombre bloc avec la clé key
static long crypter(long bloc, Cle key){
    long res = 1L;
    for (int i = 0; i < key.valeur; i++)
        res = (res * bloc) % key.modulo;
    return res;
}
// chiffre un tableau découpé en blocs de taille lg1 avec la clé key.
// Chaque bloc de taille lg1 sera encrypté en un bloc taille lg2
static byte[] chiffrer(byte[]t, int lg1, int lg2, Cle key) {
    int n = t.length / lg1; // nombre de blocs de taille lg1
    long bloc; // bloc exprimé comme nombre
    int pos;
    // allocation du message chiffré
    long chiffre;
    byte[] msgChiffre = new byte[n * lg2];
    for(int i = 0; i < n; i++) {
        bloc = 0;
        for(int j = 0; j < lg1; j++) bloc = bloc * 10 + t[lg1 * i + j];
        chiffre = crypter(bloc, key);
        pos = lg2 *(i + 1) -1;
        for(int j = 0; j < lg2; j++) {
            msgChiffre[pos--] = (byte)(chiffre % 10);
            chiffre /= 10;
        }
    }
    return msgChiffre;
}
}

```

```

public static void main(String[] args){
    if(args.length == 0){
        System.err.println("Paramètre absent");
        System.exit(2);
    }
    Cle[] keys = Cle.creerCles(37, 41, 91);
    System.out.println("=====");
    System.out.println("Cle publique : " + keys[0]);
    System.out.println("Cle privée : " + keys[1]);
    System.out.println("=====");
    System.out.println("Message en clair :");
    System.out.println(" " + args[0]);
    byte[] msgNum = numeriser(args[0], 3); // suite des chiffres du message numérisé
    System.out.println("Message numérisé :");
    System.out.print(" ");
    imprimer(msgNum);
    byte[] chiffre = chiffrer(msgNum, 3, 4, keys[0]);
    System.out.println("Message chiffré :");
    System.out.print(" ");
    imprimer(chiffre);
    byte[] dechiffre = chiffrer(chiffre, 4, 3, keys[1]);
    System.out.println("Message déchiffré sous forme numérique :");
    System.out.print(" ");
    imprimer(dechiffre);
    System.out.println("Message déchiffré :");
    System.out.print(" ");
    System.out.println(construireMsg(dechiffre));
}
}
--> java RSA LES_SANGLOTS_LONGS_DES_VIOLONS_DE_L'AUTOMNE_BERCENT_MON_COEUR_D'UNE_LANGUEUR_
MONOTONE
=====
Cle publique : (91,1517)
Cle privée : (1171,1517)
=====
Message en clair :
    LES_SANGLOTS_LONGS_DES_VIOLONS_DE_LAUTOMNE_BERCENT_MON_COEUR_DUNE_LANGUEUR_MONOTONE
Message numérisé :
    00766983958365787176798483957679787183956869839586737976797883956869957665858479777
8699566698267697884957779789567796985829568857869957665787185698582957779787984797869
Message chiffré :
    10061224032115080657046101380533032106130387046103210505004212831422037901380423032
1050507360613085907480387077807360418004212500820088400790738078910850462127206880505
07340648007900730935133200420730007907380935053303870648
Message déchiffré sous forme numérique :
    00766983958365787176798483957679787183956869839586737976797883956869957665858479777
8699566698267697884957779789567796985829568857869957665787185698582957779787984797869
Message déchiffré :
    LES_SANGLOTS_LONGS_DES_VIOLONS_DE_LAUTOMNE_BERCENT_MON_COEUR_DUNE_LANGUEUR_MONOTONE
-->

```