Cryptographie symétrique, courbes elliptiques et échanges de clés sécurisés

Charles De Clercq
Université Paris 13

Aurélien Greuet
Université de Versailles
`http://lmv.math.cnrs.fr/annuaire/aurelien-greuet/`

1 Introduction

L'objectif de cette activité est double. Il s'agit à la fois d'introduire les problèmes bien réels de la cryptographie, mais aussi l'utilisation d'outils mathématiques pour (tenter) de résoudre ces problèmes. Nous avons fait le choix de vous faire découvrir la cryptographie avant tout par l'*expérience*, l'idée étant que pour réussir à décrypter un code, il vaut mieux parfois être observateur et attentif, plutôt qu'avoir recours à un super-calculateur !

On recommande vivement de consulter le site `http://www.bibmath.net/` ainsi que l'ouvrage *Cours de cryptographie* de Gilles Zémor si vous désirez en savoir plus sur le sujet. La partie de ce cours concernant la cryptographie symétrique est très fortement inspirée de ces références, qui proposent bien d'autres méthodes et applications cryptographiques.

Dans toute cette activité, on utilisera le logiciel de calcul formel Sage (`www.sagemath.org`) pour manipuler les différents chiffrements étudiés.

Nous allons d'abord voir ce qu'est la cryptographie ainsi que ses problématiques. Ensuite nous manipulerons les premiers exemples de chiffrements et on essayera de voir comment les casser. On cherchera alors de nouvelles méthodes de chiffrement, qui n'auront pas les faiblesses des précédentes. Puis, après un aperçu de certaines méthodes relativement récentes, nous nous intéresserons à la cryptographie asymétrique, en particulier pour l'échange de clé sécurisé. Pour cela, nous aurons besoin d'étudier les courbes elliptiques et leurs propriétés, aussi bien théoriquement que de manière pratique.

2 Qu'est-ce que la cryptographie ?

La cryptographie est l'art de transmettre des données confidentielles. Son objectif est de mettre en place des outils permettant de transmettre un message sous forme cryptée, de telle sorte que seules certaines personnes puissent retrouver le message originel.

Le message de départ, avant qu'il ne soit chiffré, est le *message en clair*. Le message obtenu après avoir chiffré le message en clair est le *cryptogramme* ou simplement le *chiffré*. Avant de rentrer dans de plus amples détails, voilà un exercice qui devrait éveiller votre curiosité. Quelle sera votre stratégie ?

Exercice 1

`zs gwubs hfoqs jcig ojowh z'owf rs qsg qvcgsg ei'sqfwjsbh gif zsg aifg zsg jouopcbrg, zsg jczsifg, dcif gs rcbbsf sbhfs sil rsg fsbgswubsasbhg gif zsg usbg ri jcwgwbous ci zsg qcidg o towfs, ei'wzg gcbh gsizg o rsqcrsf.`

`zciwg ofoucb`

Solution

Le signe tracé vous avait l'air de ces choses qu'écrivent sur les murs les vagabonds, les voleurs, pour se donner entre eux des renseignements sur les gens du voisinage ou les coups à faire, qu'ils sont seuls à décoder.

Louis Aragon

□

La suite est destinée à expliquer, pour ceux n'ayant pas trouvé, comment décrypter ce code et trouver des solutions pour éviter ce type d'attaque.

3 Problématiques de la cryptographie

3.1 Un peu d'histoire

Un des aspects essentiels de la cryptographie est donc de trouver un moyen de chiffrement (on parle de *fonction cryptographique*) aussi difficile à déjouer que possible pour les éventuels « pirates ». Comme vous l'avez (peut-être) vu, le chiffrement de l'exercice 1 n'est pas très sophistiqué. Il s'agit en fait du premier exemple d'usage de la cryptographie, le *Chiffrement de César*. Son principe est simple : on choisit un nombre, grâce auquel on décale l'alphabet. On obtient par exemple pour le nombre 4 le chiffrement suivant.

Un exemple de chiffrement de César :

A	B	C	D	E	F	G	H	I	J	K	L	M
E	F	G	H	I	J	K	L	M	N	O	P	Q

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
R	S	T	U	V	W	X	Y	Z	A	B	C	D

Exercice 2

Écrire une fonction qui prend en entrée un message sous forme d'une chaîne de caractères et un nombre entier et qui renvoie le message chiffré, après un décalage correspondant au nombre donné. En déduire une fonction qui déchiffre un message chiffré par décalage.

Pour cela (et surtout pour la suite), on peut commencer par écrire une fonction qui prend en entrée une chaîne de caractères constituée uniquement de lettres minuscules ou d'espaces et qui renvoie une liste de nombre entre 0 et 26, 0 correspondant à l'espace, 1 à la lettre A, 2 à la lettre B et ainsi de suite. De même, une fonction qui prend en entrée une liste de nombres entiers compris entre 0 et 26 et qui la convertit en chaîne de caractères pourra être très utile par la suite.

Solution

Comme indiqué, on commence par une fonction qui convertit un texte en liste de nombres. On pourrait accepter tous les caractères mais pour simplifier et parce que c'est suffisant pour comprendre les mécanismes, on ne considère que l'espace et les 26 lettres minuscules. Au cas où l'entrée ne respecte pas cette contrainte, on convertira tous les autres caractères en espaces.

On part donc d'une liste vide `numlist` et on parcourt la chaîne de caractère. On calcule le code ASCII correspondant au caractère courant. S'il est entre 97 et 122, c'est bien une minuscule, et en retranchant 96, on aura bien la correspondance $a \mapsto 1$, $b \mapsto 2$ et ainsi de suite. On rajoute donc la valeur obtenue à la liste `numlist`. Sinon, on rajoute 0 (ce qui correspond à une espace) à notre liste `numlist`.

```
def string2numlist ( phrase ) :

  numlist = []

  for pos in range(0, len(phrase)) :
      ## si 97 =< ord  =< 122 on a bien une minuscule
      if ord(phrase[pos]) in range (97, 123) :
      ## et on enleve 96 pour avoir a -> 1, b -> 2 ...
          numlist.append( ord(phrase[pos]) - 96)
      ## dans tous les autres cas, on met 0
      else :
          numlist.append( 0 )
  return (numlist)
```

Inversement, pour convertir une liste de nombres entiers compris entre 0 et 26 en chaîne de caractères (`chr(32)` correspond à une espace) :

```
def numlist2string ( numlist ) :

    phrase = ''

    for pos in range(0, len(numlist)) :
        if numlist[pos] in range(1,27) :
            phrase = phrase + chr(numlist[pos] + 96)
        else :
            phrase = phrase + chr(32)
    return (phrase)
```

Maintenant que ces fonctions peuvent être utilisées, il suffit presque d'ajouter le nombre correspondant au décalage à chaque élément de la liste pour obtenir le chiffrement par décalage. Il faut juste faire attention aux valeurs qui ne sont plus comprises, après ajout, entre 1 et 26. En effet, il est naturel de dire qu'on tourne en rond, c'est à dire que la lettre qui suit `z` est la lettre `a`. Ça ressemble très fort à des modulos, mais en mettant un simple modulo 27 (n'oublions pas que notre alphabet est constitué de 26 lettres +1 pour l'espace) alors des caractères seraient convertis en espaces et les espaces en caractères. Ce n'est pas gênant, au contraire, ça pourrait perturber un éventuel attaquant. Cependant, en général quand on fait ces chiffrements à la main, on a tendance à ne décaler que les lettres. On va donc essayer d'obtenir le même résultat.

On commence donc par convertir le message en liste de nombre à l'aide de `string2numlist`. Ensuite on va parcourir cette liste de nombre et commencer par regarder si le caractère considéré est un 0. Si c'est le cas, on ne le change pas, ce qui signifie que les espaces restent des espaces. Si le nombre courant n'est pas un 0, alors il est entre 1 et 26 et on veut lui ajouter le décalage et se retrouver avec un nombre entre 1 et 26. On a donc envie que 27 soit égal à 1 (au niveau des lettres, cela signifie qu'après `z`, on a la lettre `a`), ce qui est vrai modulo 26. Cependant, si on demande le reste modulo 26, alors 26 sera égal à 0 et donc ce qui est censé être converti en `z` sera converti en espace. Pour que 27 soit bien égal à 1 sans avoir ce problème de 26 qui devient 0, on va d'abord retrancher 1 à notre nombre de manière à avoir un nombre entre 0 et 25. On lui ajoute le décalage et on prend son reste modulo 26. On obtient ainsi un nombre entre 0 et 25 et on ajoute 1 pour compenser le -1 précédent. Ainsi, on obtient bien un nombre entre 1 et 26 correspondant au décalage voulu.

Le lecteur peu convaincu par cette argumentation pourra essayer de se convaincre que ça fonctionne avec des exemples. On aurait aussi pu s'en sortir avec une série de `if`.

```
def chiffre_decalage (message, dec) :
  mess_numlist = string2numlist (message)
  crypt_numlist = []
  for i in range(len(mess_numlist)) :
     if mess_numlist[i] <> 0:
         crypt_numlist.append((mess_numlist[i]-1+dec)%26+1)
     else :
```

```
            crypt_numlist.append(0)
    return numlist2string(crypt_numlist)
```

Évidement pour déchiffrer, il suffit de décaler dans l'autre sens, donc de faire comme si on chiffrait mais avec un signe moins devant le décalage.

```
def dechiffre_decalage (message, dec) :
    return chiffre_decalage (message, -dec)
```

□

Suétone, un écrivain Romain du I^{er} siècle évoque dans *La vie des 12 Césars* que Jules César employait déjà à cette époque le même type de chiffrement que celui de l'exercice 1. L'exercice suivant est l'occasion de réfléchir sur les stratégies mises en place lors de l'exercice 1.

Exercice 3

Pouvez-vous identifier le défaut principal du chiffrement de César ?

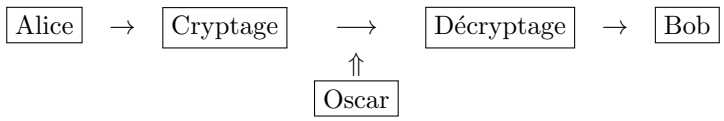
Solution

Il n'y a que 26 décalages possibles pour le chiffrement de César, autant que de lettres de l'alphabet (en comptant le décalage de 0 !). Un ordinateur ou un humain (suffisamment patient) peut très facilement tester ces 26 possibilités et est sûr de décrypter le cryptogramme. □

3.2 Et les destinataires ?

Comme nous l'avons vu, le chiffrement de César est loin d'être inviolable. Mais un autre problème bien plus général se pose : comment le destinataire peut-il décrypter le message, puisqu'il ne connaît pas a priori le chiffre de décalage ? Bien entendu, il n'est pas question de transformer le destinataire en « pirate » !

Le destinataire doit (ne serait-ce que pour des questions de temps) être capable de déchiffrer le message sans encombre. Cette situation (dite de *communication confidentielle*) est présentée le plus souvent de la façon suivante :



Deux personnes que l'on appelle, comme il est d'usage, Alice et Bob, veulent communiquer de manière confidentielle. Le *cryptanalyste* (le « pirate ») Oscar réussit a obtenir le message crypté et essaie de le décrypter. Comme nous l'avons vu plus tôt avec le chiffrement de César, pour que Bob puisse décrypter le message, il faut qu'il connaisse la « clé », c'est à dire le décalage choisi par Alice pour crypter le message en clair.

La parade à ce problème trouvée par César durant l'antiquité est originale. César rasait la tête d'un esclave et inscrivait la clé sur son crâne, puis attendait que ses cheveux repoussent avant de l'envoyer au destinataire. Cette méthode est aussi discutable humainement qu'inefficace bien entendu, et nous verrons par la suite comment contourner ce problème.

Notons que le chiffrement de César a été encore utilisé bien après l'antiquité, et même jusqu'au 20^{e} siècle. Il est encore d'ailleurs utilisé dans certains forums de discussions sur internet pour éviter les *spoilers* !

3.3 Cryptographie par permutation

Un exemple légèrement plus élaboré que le cryptage de César est le *cryptage par permutation*. Le fonctionnement est très simple et similaire au cryptage de César. Chaque lettre est remplacée par

une autre, choisie au hasard. Il n'est donc plus question de retenir que le « décalage » comme pour la méthode de César. Bien entendu, il ne faut pas choisir la même lettre pour crypter deux lettres différentes.
Voilà un exemple de cryptage par permutation.

A	B	C	D	E	F	G	H	I	J	K	L	M
M	A	P	B	O	D	R	C	Q	F	T	E	S

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
H	V	G	U	J	X	I	W	L	Z	K	Y	N

Exercice 4

Écrire une fonction qui prend en entrée un message sous forme d'une chaîne de caractères qui renvoie le message chiffré par une permutation.

En déduire la fonction qui déchiffre un message chiffré par une permutation.

Solution

Le chiffrement est relativement simple. On commence par convertir le message et la permutation en liste de nombres. On va stocker le message chiffré dans une nouvelle liste. On parcourt donc la liste correspondant au message. Si le nombre courant n'est pas un 0 (donc le caractère correspondant n'est pas une espace), on applique la permutation. On note i ce nombre. Pour ça, on rajoute à a liste correspondant au chiffré le $(i-1)$-ième élément de la liste de nombres correspondant aux permutations (la numérotation des listes commençant par 0, le i-ème élément de la liste de permutation est la lettre qui remplacera la $(i+1)$-ième lettre de l'alphabet. Donc pour avoir ce qui correspond à la i-ème lettre, on prend l'élément de la liste de permutation en position $i-1$. Si le nombre courant est un 0, on ne le change pas. On rajoute donc simplement 0 à la liste correspondant au message chiffré.

```
def chiffre_permutation (message, permutation) :
 mess_numlist = string2numlist (message)
 perm_numlist = string2numlist (permutation)

 crypt_numlist = []

 for pos in range(len(mess_numlist)) :
   if mess_numlist[pos] <> 0 :
     crypt_numlist.append(perm_numlist[mess_numlist[pos]-1])
   else :
     crypt_numlist.append(0)
 return numlist2string(crypt_numlist)
```

Pour le déchiffrement, c'est comme pour le décalage où il suffisait de décaler dans le sens opposé. Ici il suffit d'être capable de calculer la permutation inverse puisqu'il suffira de chiffrer avec la permutation inverse pour retomber sur le message en clair.

On appelle `perm_numlist` la liste de nombres correspondant à la permutation. On va utiliser la fonction `perm_numlist.index(j)` qui renvoie le plus petit `i` tel que `perm_numlist[i]=j`. En effet, si on construit une liste telle que pour tout k entier compris entre 1 et 26, le k-ième élément est `perm_numlist.index(k) + 1`, alors une la permutation correspondant va bien être l'inverse de la première.

Encore une fois, si l'explication n'est pas convaincante, il faut s'en convaincre en manipulant des exemples.

```
def dechiffre_permutation (message, permutation) :
 mess_numlist = string2numlist (message)
 perm_numlist = string2numlist (permutation)
```

```
inv_numlist=[perm_numlist.index(k)+1 for k in range(1,27)]
permutation_inverse = numlist2string (inv_numlist)

return chiffre_permutation(message, permutation_inverse)
```

□

Exercice 5

mv oukdo yfuq klikoq voq fckmluq sihfmuoq nviq yo sfqfdy tio yo yocmqmlu.

fuydo xmyo

Solution

Il entre dans toutes les actions humaines plus de hasard que de décision.

André Gide

□

Exercice 6

Combien y a-t-il de cryptages par permutation différents ?

Solution

Pour remplacer le A, on dispose des 26 lettres de l'alphabet. Pour le B il n'y a plus que 25 possibilités, et ainsi de suite. Il y a donc $26 \times 25 \times 24 \times \cdots \times 2 \times 1$ possibilités de cryptage par permutation. □

Ce nouveau type de cryptage évite un écueil : il n'est plus possible de décrypter le cryptogramme en essayant brutalement toutes les possibilités (il y en a trop !). Nous allons voir dans la prochaine partie qu'il existe cependant d'autres possibilités pour décrypter de tels messages.

4 Les limites du cryptage par permutation

Comme nous l'avons vu précédemment, le cryptage par permutation n'est pas raisonnablement attaquable en testant toutes les clés possibles (on parle dans ce cas d'une attaque *brute-force*). Comment allez-vous donc réussir à décrypter les messages suivants ?

Exercice 7

f rfdco,

beqdo qsqmqo, v'qorqdq rfd sf rdqoqugq mlao dfrrqsqd s'chrldgfubq pa blugqygq fncu pq

pqbdjrgqd au
hqoofxq. sf nldbq idagq u'qog dcqu ofuo au rqa pq hfscbq.

icqu bldpcfsqhqug,
befdsqo pq bsqdbt

Solution

Voici la lettre (la mise en page pouvait fournir cet indice) déchiffrée :

À Paris,

Chers élèves,

J'espère par la présente vous rappeler l'importance du contexte afin de décrypter un message. La force brute n'est rien sans un peu de malice.

Bien cordialement,
Charles De Clercq □

Exercice 8

`ut ruld etppcbut dtipte st it yfqst dtpkce nl'cu q'o kce klilq dtipte.`

`jtkq-mpkqifcd stqckl`

Solution

Le plus terrible secret de ce monde serait qu'il n'y ait aucun secret.

Jean-François Deniau

□

Comme le montrent les exercices précédents, il existe malgré l'énorme nombre de possibilités plusieurs façons de résoudre (avec un peu de chance et beaucoup d'essais) un cryptogramme crypté par permutation. En effet il faut savoir tenir compte, comme à l'exercice 7, des indices extérieurs au code mais pourtant bien présents dans le message. Ce détail, bien qu'il paraisse anodin à première vue, a déjà été décisif dans l'histoire de la cryptographie, comme nous le verrons lorsque nous aborderons le XX^{e} siècle et la machine *Enigma*.

Un autre indice déterminant est la répartition et la proportion des lettres dans le cryptogramme. En effet les lettres ont une fréquence d'apparition bien spécifique dans chaque langue. Par exemple en Français, la lettre apparaissant le plus fréquemment est le « E » (viennent ensuite « A », « I », « S », « T », « N », « R », « U », etc).

En observant le code crypté, il est fort probable de pouvoir identifier le cryptage de cette lettre en observant la lettre qui est la plus présente. Une étude des mots courts permet aussi de proche en proche de pointer des absurdités et (avec un peu de chance) de s'approcher d'un décryptage complet.

Bien entendu, notre chère Alice pourrait bien nous réserver une fourberie, en fournissant par exemple à Bob un message ne contenant aucun « E ». Cette méthode n'est donc pas infaillible mais s'avère parfois très performante.

5 Autres méthodes cryptographiques

5.1 Des lettres aux chiffres

Un autre moyen de cryptage est le *carré de Polybe*. Polybe (historien grec du II^{e} siècle avant J.C.) dispose les lettres de l'alphabet dans un tableau de taille 5×5. Pour crypter un message, il suffit de remplacer une lettre par ses coordonnées.

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I, J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Le carré de Polybe

La lettre « A » est donc remplacée par 11, tandis que la lettre « S » correspondra a 43 après cryptage. Remarquons que cette méthode a deux intérêts. D'une part le message crypté est plus difficile à lire car il n'est plus évident de reconnaître directement les lettres. En outre seuls les neuf symboles $1, ..., 9$ sont utilisés. Notons enfin que l'on peut compliquer encore un peu plus la situation en remplissant le tableau de Polybe dans un autre ordre que l'ordre alphabétique.

Le carré de Polybe était encore utilisé au début du XX^{e} siècle. Les nihilistes Russes (une organisation clandestine visant a renverser le Tsar) utilisaient cette méthode pour communiquer quand ils étaient emprisonnés, comme l'atteste la photographie ci-dessous des geôles de Saint Petersbourg, où les détenus communiquaient en frappant en rythme, à la manière du Morse.

Exercice 9

Trouver un moyen relativement simple pour implémenter le chiffrement et le déchiffrement avec le carré de Polybe.

Solution

Le lecteur devrait commencer à être familier avec les fonctions et les méthodes utilisées, on va donc détailler un peu moins.

Ici on commence par construire la liste `polybe`, de sorte que `polybe[i]` corresponde au chiffrement de la $i+1$-ième lettre de l'alphabet. Ce ne sera pas tout à fait le cas puisque `i` et `j` sont tous les deux envoyés sur `'24'`, il faudra donc faire attention aux décalages éventuels. Ensuite comme d'habitude, on parcourt la liste de nombre correspondant au message.

Contrairement aux cas précédents, on ne construit pas une liste correspondant au message chiffré à laquelle on ajoute un élément à chaque étape. Ici, on fait l'analogue avec des chaînes de caractères :

```
def chiffre_polybe (message) :
  polybe = [str(i+1) +  str(j+1) for i in range(5)
       for j in range(5)]
  mess_numlist = string2numlist(message)
  result = ''
  for i in range(len(mess_numlist)) :
      if mess_numlist[i] == 10 :
          result = result + polybe[8]
      elif mess_numlist[i] == 0 :
          result = result + ' '
      elif mess_numlist[i] < 10 :
          result = result + polybe[mess_numlist[i]-1]
      elif mess_numlist[i] > 10 :
          result = result + polybe[mess_numlist[i]-2]
  return result
```

Pour déchiffrer, on a envie de prendre les caractères du message chiffré deux par deux. Mais si on tombe sur une espace, il faut simplement passer au suivant, sans faire de déchiffrement. On procède donc de la manière suivante :

```
def dechiffre_polybe (message) :
  polybe = [str(i+1)+str(j+1) for i in range(5)
  for j in range(5)]
```

```
current = ''
decrypt = []
i = 0
while i < len(message):
    if message[i] <> ' ' :
        current = message[i] + message[i+1]
        i = i+2
        if ZZ(current) <= 24 :
            decrypt.append(polybe.index(current)+1)
        else :
            decrypt.append(polybe.index(current)+2)
    else :
        decrypt.append(0)
        i = i + 1

return numlist2string(decrypt)
```

□

Exercice 10

Chiffrez le titre de votre livre (ou film, ou musique, ou jeu...) préféré à l'aide du carré du carré de Polybe. Faites-le décrypter par votre voisin.

5.2 La méthode de Vigenère

Nous présentons désormais la méthode considérée comme l'aboutissement des méthodes classiques de cryptographie. Les méthodes précédentes (méthode de César, méthode par permutation, tableau de Polybe) sont bien faibles face aux attaques car il est facile d'y identifier la fréquence d'apparition des lettres. C'est au XVI^{e} siècle qu'est introduite une nouvelle méthode, la méthode de Vigenère.

Vigenère, diplomate, écrivain et historien Français publie en 1586 le *Traicté des chiffres ou Secrètes manières d'écrire*, proposant une nouvelle méthode de cryptage. L'idée est d'améliorer la méthode de César, en modifiant le décalage à chaque nouvelle lettre. L'intérêt de cette méthode est donc qu'au fur et à mesure du texte, une même lettre est cryptée de manières différentes, puisque le décalage change à chaque lettre. Plutôt que de longues explications, examinons cette méthode par un exemple.

Choisissons le message en clair « CRYPTOGRAPHIE » ainsi que la clé « TEST ». On dispose le message en clair et la clé de la manière suivante pour obtenir le cryptogramme :

C	R	Y	P	T	O	G	R	A	P	H	I	E
T	E	S	T	T	E	S	T	T	E	S	T	T

Une fois cela fait, il devient très simple de crypter notre mot, en se reportant au tableau de correspondance de Vigenère, présenté à la figure 1.

À la première lettre « C » est associée la clé « T ». La première lettre du cryptogramme est la lettre se trouvant à l'intersection de la ligne « C » et de la colonne « T », c'est à dire « V ». On recommence l'opération pour chacune des lettres afin d'obtenir le cryptogramme.

`vvqimsykttzbx`

Comme vous le remarquez, il est complètement vain d'essayer de décrypter ce message en fonction de la répartition des lettres, puisque par exemple la lettre « V » correspond dans le cryptogramme aux deux lettres « C » et « R ». Le cryptage est donc très simple avec cette méthode, mais qu'en

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

FIGURE 1 – Tableau de correspondance de Vigenère

est-il du décryptage ?

Le décryptage est lui aussi très aisé, à condition de connaître la clé qui a été utilisée pour crypter. On introduit a nouveau le cryptogramme ainsi que la clé dans un tableau comme suit :

V	V	Q	I	M	S	Y	K	T	T	Z	B	X
T	E	S	T	T	E	S	T	T	E	S	T	T

Pour retrouver la première lettre du message en clair, on regarde la colonne associée à la première lettre de la clé : « T ». Dans cette colonne on retrouve la première lettre du cryptogramme : « V ». La première lettre du message en clair n'est autre que la lettre correspondant à la ligne du « V », c'est à dire « C ».

Exercice 11

Vérifier que l'on retrouve bien le message en clair par cette méthode. Choisissez une clé et cryptez

votre livre (votre groupe de musique, votre matière...) préféré en utilisant cette clé et la méthode de Vigenère. Donnez la clé a votre voisin ainsi que le cryptogramme pour qu'il puisse retrouver le message en clair.

La méthode de Vigenère introduit donc la notion de clé, qui est essentielle. Il peut paraître inutile de crypter un message si la personne devant le décrypter ne connaît pas la clé (ce qui inexact, comme nous le verrons par la suite). Un des avantages de la méthode de Vigenère est que la clé est la même, pour crypter le message en clair et pour décrypter le cryptogramme. On parle alors de *cryptographie symétrique*.

Exercice 12

Décrypter le code suivant.

`qexhsdi di vmginiri.`

Indice : la clé est constituée de 2 *lettres*

Solution

La clé s'avère être EA et on décrypte en trouvant : Méthode de Vigenère. □

Exercice 13

Écrire une fonction qui prend en entrée un message et une clé, tout deux sous forme d'une chaîne de caractères, qui renvoie le message chiffré par chiffrement de Vigenère. Pour ne pas s'embêter et pour augmenter (très légérement, mais tout de même) la sécurité, on appliquera la clé sur tous les caractères du message, espaces comprises. On n'utilisera donc pas le tableau donné précédemment et on ajoutera naturellement les nombres correspondant aux lettres du message et de la clé pour obtenir le chiffré.

Écrire une fonction analogue pour le déchiffrement.

Solution

Comme l'énoncé l'indique, on ajoute presque simplement les nombres de la liste correspondant au message à ceux correspondant à la clé. On calcule modulo 27 car on a 26 lettres +1 pour l'espace. Mais il faut aussi faire attention aux cas où la clé est plus petite que le message et penser à répéter la clé. On va pour cela considérer les indices de la clé modulo sa taille.

```
def chiffrement ( message, cle ) :

 chiffre_numlist = []
 mess_numlist = string2numlist (message)
 cle = string2numlist (cle)

 for pos in range(0, len (mess_numlist)) :
   newlettre = (mess_numlist[pos] + cle[pos%len(cle)])%27
   chiffre_numlist.append( newlettre )

 return(numlist2string(chiffre_numlist))
```

Pour déchiffrer, il suffit de retrancher au lieu d'ajouter les nombres correspondant à la clé.

```
def dechiffrement ( message, cle ) :

 dechiffre_numlist = []
 mess_numlist = string2numlist (message)
 cle = string2numlist (cle)

 for pos in range(0, len (mess_numlist)) :
   newlettre = (mess_numlist[pos] - cle[pos%len(cle)])%27
```

```
    dechiffre_numlist.append( newlettre )
  return(numlist2string(dechiffre_numlist))
```

□

6 Aperçu de certaines méthodes du XX^{e} siècle

Toutes ces méthodes semblent bien archaïques, à l'heure de l'informatique et d'Internet. Nous allons cependant voir dans cette partie que les problématiques (et les moyens de décryptages) évoqués précédemment ont eu une grande importance, en particulier au XX^{e} siècle.

6.1 Quand la cryptographie intervient dans l'histoire

C'est au XX^{e} siècle que la cryptographie s'impose comme une science à part entière, en grande partie (comme souvent malheureusement) pour son utilité lors de la première et la seconde guerre mondiale.

Évoquons dans un premier temps un exemple frappant de l'importance prise par la cryptographie datant de 1917. La première guerre mondiale fait alors rage en Europe, et les États-Unis n'ont pas pour le moment décidé de prendre part aux hostilités. En Janvier 1917, l'Allemagne cherche à couper à tout prix les ravitaillements américains pour l'Angleterre, mais ne souhaite pas couler trop de navires Américains pour éviter leur entrée en guerre.

Afin de limiter les approvisionnements Américains, les Allemands décident d'inciter le Mexique à déclarer la guerre aux États-Unis en échange d'une aide financière et militaire. Arthur Zimmermann, ministre Allemand des affaires étrangères, envoie donc au président Mexicain un message crypté contenant les informations nécessaires ainsi que les intentions allemandes.

Le message est finalement intercepté par les services secrets britanniques, qui réussissent à décrypter le message en quelques semaines, le 22 Février 1917. Une fois transmis au président Américain, la réponse ne se fait pas attendre et le 6 Avril 1917, le président Woodstone déclare officiellement la guerre à l'Allemagne.

6.2 La seconde guerre mondiale

La cryptographie semble définitivement un enjeu majeur dès la seconde guerre mondiale. En 1940, l'Europe est ravagée par la guerre. La France et la Pologne ont capitulé et les sous-marins Allemands sèment la terreur le long des côtes Anglaises. Afin d'être toujours plus imprévisibles, ils communiquent à l'aide d'une machine mise en place par l'état-major Allemand nommée « Enigma ».

Cette machine permet de crypter n'importe quel message de manière très efficace et permet aux sous-marin de communiquer en toute impunité. L'Angleterre recrute alors de nombreux mathématiciens en vue d'essayer de décrypter les messages cryptés à l'aide d'Enigma. Parmi eux se trouve Allan Türing, mathématicien qui conçoit à ces fins le premier ordinateur, baptisé Colossus.

À l'aide d'informations glanées dans certains sous-marins (et grâce à certaines faiblesses dans le protocole Allemand), mais aussi grâce au soutien conjugué de la Pologne et des États-Unis, les Anglais parviennent finalement à décrypter ces messages et sauver nombre de navires. Cet avantage est d'autant plus décisif que les Allemands ne sauront jamais que cette prouesse a été réalisée, tandis que les alliés préparaient leurs débarquements en toute discrétion.

6.3 Une application actuelle de la cryptographie

Le système de cryptage le plus utilisé actuellement est le système RSA. Rappelons que le système RSA repose sur la difficulté à factoriser les entiers de grande taille en un produit de facteurs premiers. Nos cartes bancaires, par exemple, utilisent ce système de cryptage.

Toute carte bancaire est équipée d'une puce qui renferme une signature unique et ne pouvant être modifiée. Cette signature repose sur une clé (très) secrète à laquelle très peu de personnes ont accès. En insérant une carte dans un terminal, celui-ci vérifie que la signature a bien été générée par la clé secrète pour autoriser une transaction.

En 1998, Serge Humpich, un ingénieur Français, réussit a découvrir la clé secrète en factorisant l'entier sur lequel repose le système RSA. Il arrive ainsi à montrer qu'il est possible de fabriquer une carte bancaire fonctionnant chez les commerçants. Il est condamné en 2000 pour cette falsification et pour avoir montré qu'il était possible de contourner ce système. Une nouveau nombre premier (plus grand et donc plus difficile à factoriser) remplace depuis cette affaire l'ancien, et la nouvelle clé secrète est pour le moment inviolée.

7 Cryptographie asymétrique

Jusqu'ici on a étudié des méthodes de cryptographie à *clé secrète* : pour communiquer de manière sécurisée, deux individus doivent partager une même clé secrète. En pratique, il est important de souvent changer de clé pour garantir la sécurité des échanges. Comment s'échanger ces clés de manière simple et pratique ?

Pour répondre à cette question, on va s'intéresser à la cryptographie *asymétrique*, aussi appelée cryptographie *à clé publique*. En particulier, on va voir comment procéder à un échange de clé sécurisé. Le but est simplement de trouver un moyen pour que deux personnes aient une clé secrète commune en ayant échangé des données publiques, accessibles à tous.

Nous allons procéder à un échange de clé de type Diffie-Hellman sur des courbes elliptiques. Avant de rentrer dans la théorie, expliquons le principe à l'aide de « boites noires » : on dispose d'outils, de fonctions, dont on ne connaît pas le fonctionnement interne mais qu'on sait utiliser. On verra ensuite comment et pourquoi ça fonctionne.

On suppose qu'on dispose d'une fonction `param_publics` qui génère un point P, un nombre premier p et une liste de deux éléments qu'on appelle `courbe`, représenté par une liste de deux éléments. Ce sont les paramètres publics que doivent se partager les deux protagonistes.

Nous avons une deuxième fonction, `echange_public`, qui prend comme arguments un nombre entier a et les paramètres publics P, p et `courbe`. Elle renvoie un nouveau point, qui dépend de P, représenté lui aussi par une liste de deux éléments. On note ce point aP. Cette fonction a deux propriétés intéressantes :

- pour tous nombres entiers a et b, $a(bP) = b(aP) = (ab)P$;
- connaissant les paramètres publics et aP, on ne connaît pas de méthode spécifique pour retrouver a.

Cette deuxième propriété signifie que si on ne connaît que p, `courbe`, P et aP et qu'on veut retrouver a, il n'y a pas d'autre moyen que de calculer des kP et espérer qu'on tombe sur $k = a$. Il est donc difficile de retrouver a.

Pour utiliser les points comme des clés secrètes, on dispose d'un outil de conversion `point_en_cle` qui prend un point en entrée et renvoie une clé sous forme de caractères.

Enfin, on a une quatrième fonction `plus_un` qui prend en arguments les paramètres publics p, `courbe`, P et un point de la forme kP et renvoie $(k + 1)P$. Cette fonction pourra être utilisée pour une attaque *brute-force* destinée à retrouver a à partir de aP, en calculant tous les kP possibles et en voyant si on retombe sur aP.

Comme toujours, on appelle Alice et Bob les deux personnages qui souhaitent avoir une clé secrète commune. Un des deux va utiliser la fonction `param_publics` pour générer des paramètres et transmettre ces paramètres à l'autre ou les publier dans un annuaire.

Ensuite, Alice choisit un nombre a qu'elle garde secret. Elle calcule le point aP grâce à la fonction `echange_public` et envoie aP à Bob. Ce dernier choisit un nombre b qu'il garde secret, calcule bP et l'envoie à Alice.

Notons que d'après la première propriété, rien ne permet à quelqu'un ayant espionné les communications d'Alice et Bob de retrouver les nombres secrets a et b.

Alice va ensuite calculer $a(bP)$: elle connaît a ainsi que le point bP donc elle peut utiliser `echange_public(a, bP, p,courbe)`. Mais a étant secret, Alice est la seule personne à pouvoir le faire. De même, Bob va être le seul à pouvoir calculer $b(aP)$.

Or d'après la première propriété, $a(bP) = b(aP)$, donc Alice et Bob disposent de la même valeur. Ce sont les seuls à avoir pu calculer cette valeur commune d'après la deuxième propriété. Ils peuvent donc l'utiliser comme clé secrète.

Exercice 14

Écrire une fonction qui fait une attaque brute-force pour retrouver a ou b. Échanger une clé avec un autre groupe pendant qu'un troisième essaye de la casser. On commencera par des petites valeurs (inférieures à 1000) de a et b.

Solution

On peut ajouter une borne en argument, de sorte que le programme s'arrête si la borne est atteinte. De plus, on affiche des signes de vie assez régulièrement pour voir l'avancement du programme et être sûr qu'il ne plante pas. On utilisera `IntegerRange` plutôt que `range`, qui permet de gérer des grands nombres.

```
def bruteforce(p, courbe, P, aP, bP, borne) :
    current = P
    for i in IntegerRange(borne) :
        if i%20 == 0 :
            print 'i=',i
        if i <> 0 and i%200 == 0 :
            print 'c\'est un peu long...'
        if current == aP :
            print('a')
            return i+1
        if current == bP :
            print('b')
            return i+1
        current = plus_un(p, courbe, P, current)
```

□

7.1 Difficile ?

Lors de la description du protocole d'échange de clé, on a dit qu'il était « difficile » de retrouver a si on ne connaissait que les paramètres publics et aP. Mais que signifie exactement « difficile » lorsqu'on dit qu'un problème est difficile ?

D'abord la difficulté du problème précédent dépend essentiellement de taille du nombre premier. Essentiellement signifie ici que si on choisit systématiquement $a = 3$ comme nombre secret, effectivement le problème ne sera pas difficile. Mais si on fait des choix relativement génériques (par exemple générés aléatoirement) alors la difficulté dépend de la taille de p.

Pour visualiser ce qu'est un problème difficile, considérons le nombre premier

$$p = 78974695679943921743289887845048098475407298819350240596625818947103322 07.$$

Notons que p est un nombre de 73 chiffres, il est donc de l'ordre de 10^{72}. Notons aussi qu'il a fallu seulement $0,06$ secondes à Sage pour trouver p sur mon vieil ordinateur portable (commande `next_prime(11^(70))`), c'est donc très facile et très rapide.

On va maintenant calculer le temps que mettrait un algorithme relativement standard avec un processeur très haut de gamme pour résoudre le problème suivant : *Étant donnés le nombre premier p, un point P d'une courbe elliptique sur $\mathbb{F}_p$ et $Q = aP$, calculer a. .*

Le but étant simplement d'obtenir un ordre de grandeur, nous allons nous permettre de nombreuses approximations.

Nous allons considérer un algorithme qui n'est pas le plus rapide à ce jour, mais qui est sensiblement plus efficace que la recherche exhaustive (il s'agit d'une méthode de *baby-step/giant-step*. Cet méthode a l'avantage d'être « générique », c'est-à-dire qu'elle fonctionne pour résoudre un problème de logarithme discret sur n'importe quel groupe abstrait). Il faut à cet algorithme de l'ordre de $\sqrt{p}$ opérations pour résoudre notre problème.

On choisit comme processeur un des plus puissants à l'heure actuelle (en juin 2012), le Intel Core i7 Extreme Edition 3960X. À titre indicatif, ce processeur coûte aujourd'hui 999 euros. Il est capable de réaliser 177 730 000 000 instructions par seconde. Pour simplifier et parce que ça ne change pas l'ordre de grandeur, on va considérer qu'il peut faire 177 730 000 000 opérations par seconde. Si on veut le nombre d'opérations en une année, on calcule $365 \times 24 \times 3600 \times 177\ 730 \times 10^6 \simeq 10^{19}$. Un tel processeur peut donc faire environ 10^{19} opérations en un an.

Calculons maintenant le nombre d'opérations nécessaires pour résoudre le problème avec le p choisi. Comme il faut à notre algorithme de l'ordre de $\sqrt{p}$ opérations pour résoudre notre problème et que p est de l'ordre de 10^{72}, l'exécution va nécessiter environ 10^{36} opérations.

Donc avec un core i7 à 999 euros, il faudrait de l'ordre de $\dfrac{10^{36}}{10^{19}} \simeq 10^{17}$ années. Pour comparer, l'âge de l'univers est d'environ 15 milliards d'années, soit environ 10^{10} années. Il faudrait donc de l'ordre de $\dfrac{10^{17}}{10^{10}} \simeq 10^7$ ($\simeq$ 10 millions de) fois l'âge de l'univers pour que notre brave core i7 puisse résoudre le problème...Même si les services secrets disposent de super-calculateurs avec 10 millions de processeurs, il leur faudrait tout de même l'âge de l'univers, soit environ 15 milliards d'années, pour résoudre notre problème.

8 Courbes elliptiques : la théorie

En fait dans l'échange de clé précédent, les fonctions données manipulaient des points sur des courbes elliptiques.

8.1 C'est quoi une courbe elliptique ?

Pour nous, une courbe elliptique sera une courbe définie par une équation de la forme

$$y^2 = x^3 + ax + b,$$

où $4a^3 + 27b^2 \neq 0$ pour des raisons techniques.

Les points d'une courbe elliptique sont donc des couples (x, y), avec x et y réels qui vérifient l'équation $y^2 = x^3 + ax + b$. Une courbe elliptique n'est pas une fonction mais on dispose d'outils informatiques pour les dessiner (voir Figure 2).

D'après notre définition, une courbe elliptique est donc entièrement déterminée par deux paramètres, a et b. On peut donc « manipuler » une courbe elliptique en Sage en la représentant par une liste à deux éléments, `[a,b]`.

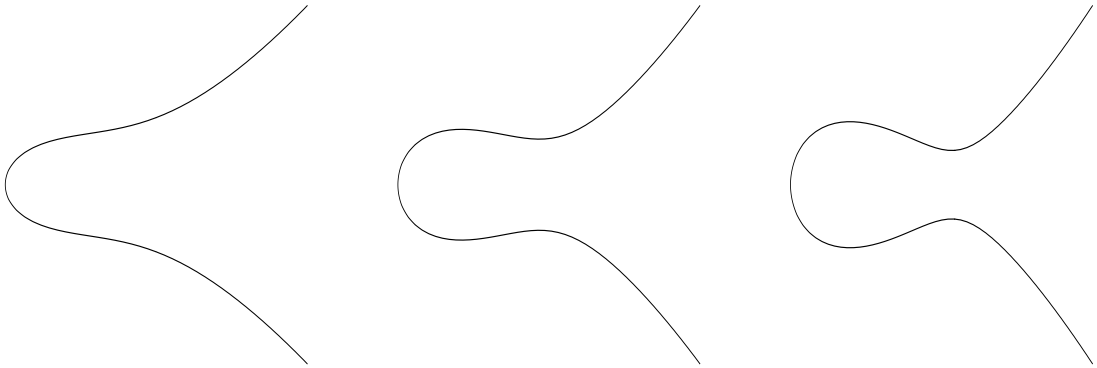
Exercice 15

Écrire une fonction Sage qui prend en argument une liste à deux éléments `[a,b]` et qui renvoie `True` si $y^2 = x^3 + ax + b$ est une courbe elliptique, `False` sinon.

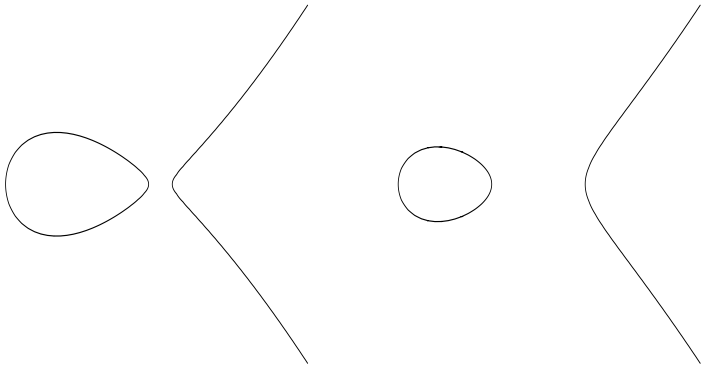
Solution

D'après la définition, il suffit de calculer $4a^3 + 27b^2$. Si c'est nul, $y^2 = x^3 + ax + b$ n'est pas l'équation d'une courbe elliptique, on renvoie donc `False`. En revanche si $4a^3 + 27b^2$ est non nul, on répond `True`.

On peut aussi rajouter une option `verbose`, qui vaut `False` par défaut. Si `verbose = False`, la fonction renvoie simplement `True` ou `False`. Par contre si `verbose = True`, en plus de renvoyer `True` ou `False`, le programme affiche l'équation de la courbe et écrit `est une courbe elliptique` ou `n'est pas une courbe elliptique`.



(a) $y^2 = x^3 + x + 2$ (b) $y^2 = x^3 - x + 2$ (c) $y^2 = x^3 - 2x + 2$



(d) $y^2 = x^3 - 2x + 1$ (e) $y^2 = x^3 - 2x$

FIGURE 2 – Quelques courbes elliptiques

```
def courbe_elliptique (courbe, verbose = False) :
   a = courbe[0];   b = courbe[1]

   if verbose :
      print 'y^2 = x^3 +',a,'x +',b

   if 4*a^3 + 27*b^2 <> 0 :
      if verbose :
          print 'est une courbe elliptique'
      return True
   else :
      if verbose :
          print 'n\'est pas une courbe elliptique'
      return False
```

On peut vérifier sur quelques exemples que la fonction renvoie bien le résultat attendu.

```
courbe = [5,2]
courbe_elliptique(courbe)
```

```
True
```

```
courbe_elliptique(courbe, verbose = True)
```

```
y^2 = x^3 + 5 x + 2
est une courbe elliptique
True
```

```
courbe_elliptique([0,0])
```

```
False
```

□

Exercice 16

Écrire une fonction qui prend en entrée un point et une courbe et renvoie `True` si le point est bien sur la courbe, `False` sinon.

Solution

```
def est_sur_la_courbe ( P, courbe) :

    if P == [infini,infini] :
        return True

    a = courbe[0];  b = courbe[1]
    xP = P[0]; yP = P[1]

    if yP^2 - xP^3 - a*xP - b == 0 :
        return True
    else :
        return False
```

```
 P = [1,1]; courbe = [-1,1]; est_sur_la_courbe (P, courbe)
```

```
True
```

```
P = [1,1]; courbe = [1,1];
est_sur_la_courbe (P, courbe)
```

```
False
```

□

8.2 Additionnons les points d'une courbe

Pour faire de la cryptographie avec les courbes elliptiques, on va définir une addition sur l'ensemble des points. Pour éviter les ambiguïtés, on notera $\oplus$ cette addition. On cherche donc, étant donnés deux points P et Q sur une courbe elliptique, à définir un troisième point R pour avoir $P \oplus Q = R$. On pourrait choisir R complètement au hasard mais on aurait peu de chance d'avoir les propriétés sympathiques des additions que l'on connaît déjà.

Exercice 17

En s'aidant d'exemples connus (les réels, les entiers relatifs), essayer de deviner quelles sont ces propriétés.

Solution

Évidement la notion de « sympathique » est relative, mais on peut s'attendre à avoir des propriétés comme

- l'associativité : si P, Q et R sont trois points sur une courbe alors $(P \oplus Q) \oplus R = P \oplus (Q \oplus R)$.
- l'existence d'un élément neutre : il existe un point $\mathcal{O}$ sur la courbe tel que pour tout point P de la courbe, $P \oplus \mathcal{O} = \mathcal{O} \oplus P = P$
- l'existence d'un opposé : pour tout point P de la courbe, il existe un unique point Q de la courbe tel que $P \oplus Q = \mathcal{O}$. On note alors $Q = -P$.

□

Nous allons voir qu'il existe une manière relativement simple pour définir une addition vérifiant les propriétés précédentes.

8.3 Le cas sympathique

Exercice 18

En général, si on prend deux points P et Q « au hasard » sur une courbe elliptique et qu'on trace la droite (PQ), combien y a-t-il de points d'intersections avec la courbe ?

Solution

Si on prend deux points pas trop particuliers sur la courbe, la droite passant par ces deux points rencontre la courbe en un troisième point R (voir Figure 3).

□

Dans un premier temps, on suppose qu'on se trouve dans le cas précédent : étant donnés deux points P et Q sur la courbe, la droite (PQ) rencontre la courbe en un unique troisième point R. On pourrait avoir envie de définir $P \oplus Q = R$. Cependant, la propriété sympathique d'*associativité* ne sera pas vérifiée comme on peut le voir sur la Figure 4.

En revanche, définissons $P \oplus Q$ comme étant le symétrique du troisième point d'intersection comme sur la Figure 5.

Alors sur un dessin (Figure 6), notre addition a l'air d'être associative. Pour s'en convaincre, on va calculer les coordonnées de $P \oplus Q$ en fonction des coordonnées de P et Q, ce qui permettra de tester sur des exemples (le cas général est difficile).

Exercice 19

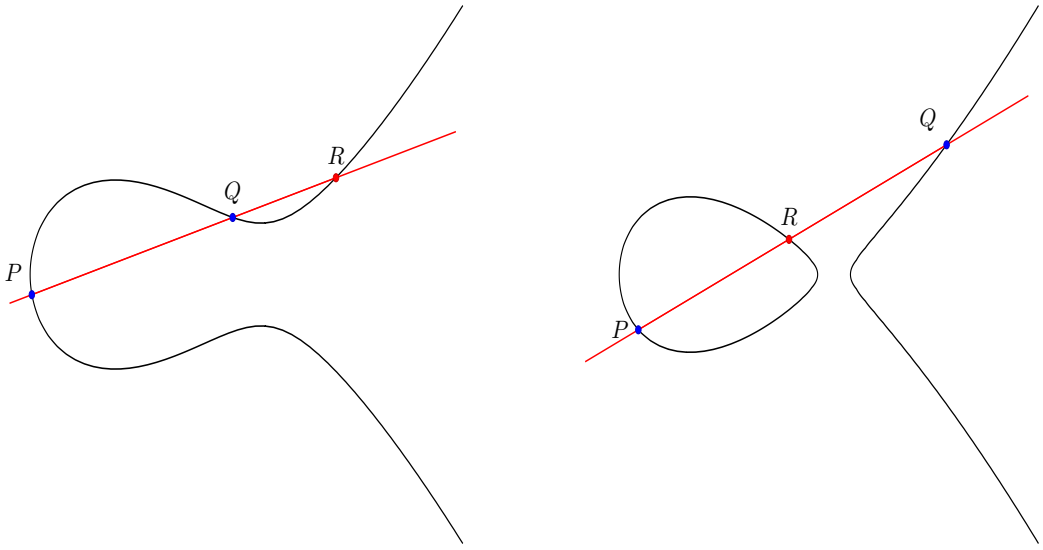


FIGURE 3 – On choisit P et Q et on trouve un troisième point R

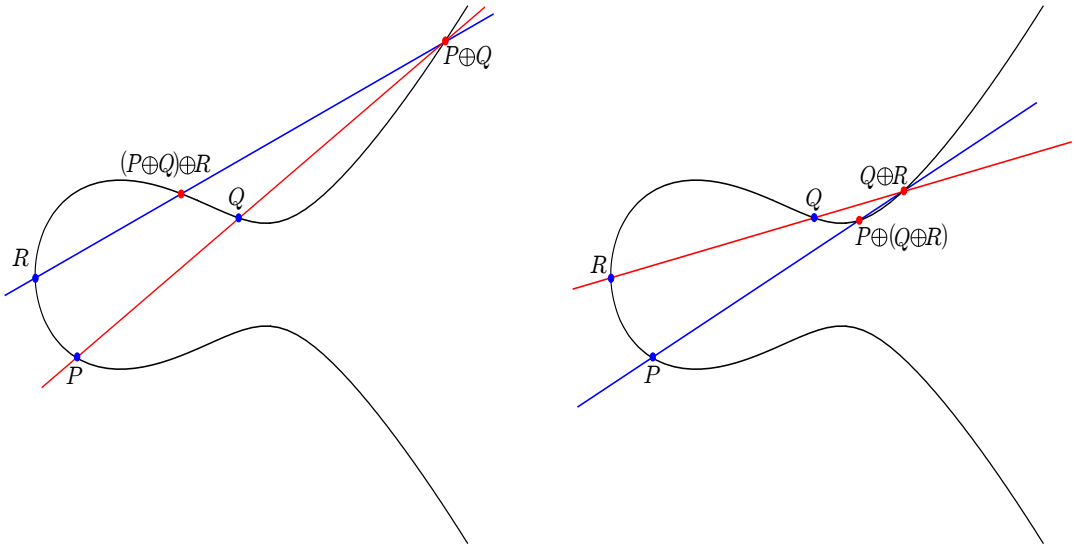


FIGURE 4 – $(P \oplus Q) \oplus R \neq P \oplus (Q \oplus R)$

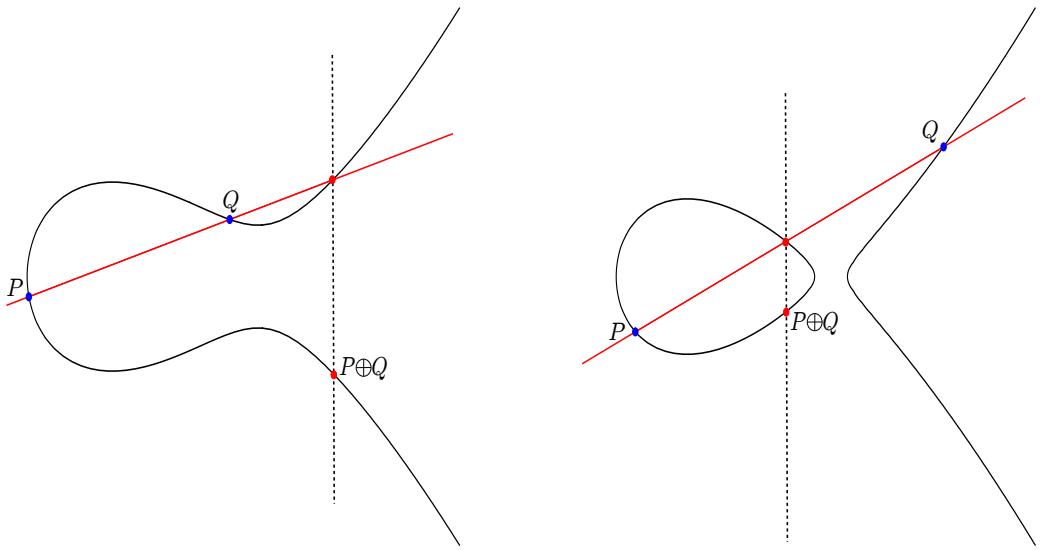


FIGURE 5 – Une meilleure définition de $P \oplus Q$

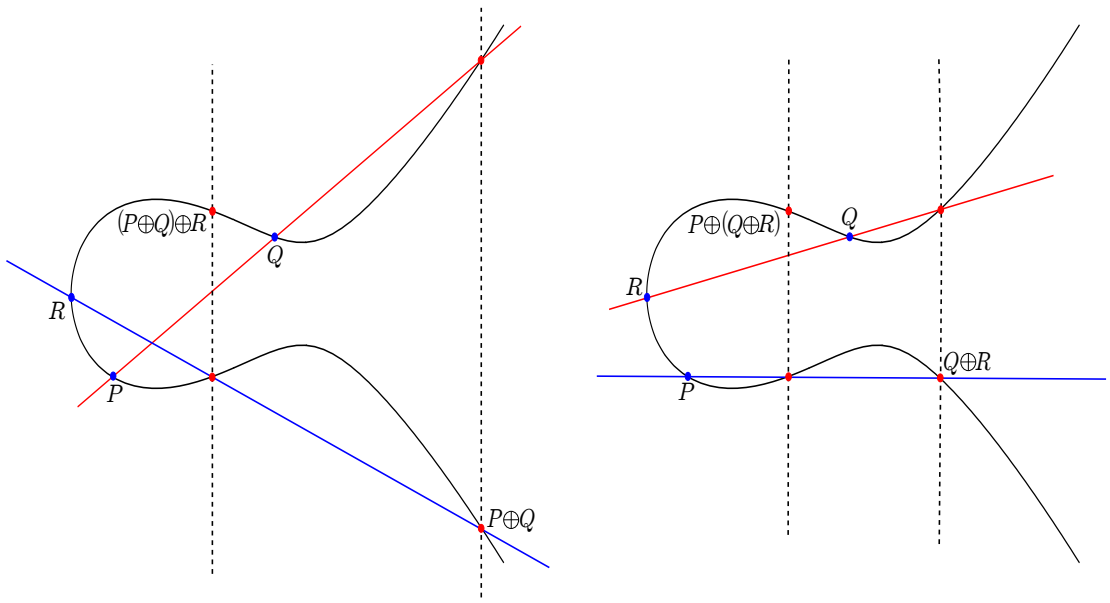


FIGURE 6 – Notre addition $\oplus$ semble être associative : Si on calcule d'abord $P \oplus Q$ et qu'on ajoute R à ce résultat, c'est-à-dire si on calcule $(P \oplus Q) \oplus R$, on trouve le même point que si on calcule $P \oplus (Q \oplus R)$, donc si fait d'abord $Q \oplus R$ puis qu'on ajoute P.

Soient P et Q deux points tels que $x_P \neq x_Q$. On note (x_P, y_P) les coordonnées de P et (x_Q, y_Q) celles de Q. Quelle est l'équation de la droite (PQ) en fonction de x_P, y_P, x_Q et y_Q ?

Solution

Comme les points P et Q sont tels que $x_P \neq x_Q$, ils ne forment pas une droite verticale, l'équation de (PQ) est donc de la forme

$$y = \lambda x + \nu.$$

Il suffit donc de déterminer λ et ν. Pour ça, on va simplement utiliser que P et Q sont sur la droite (PQ). Comme P est sur (PQ), ses coordonnées vérifient l'équation de (PQ), donc

$$y_P = \lambda x_p + \nu.$$

De même avec Q, on trouve

$$y_Q = \lambda x_Q + \nu.$$

Donc λ et ν sont solutions du système

$$\begin{cases} y_P = \lambda x_p + \nu \\ y_Q = \lambda x_Q + \nu \end{cases}$$

Pour le résoudre, on peut par exemple calculer la différence des deux lignes du système pour obtenir

$$y_Q - y_P = \lambda (x_Q - x_P).$$

Maintenant on a envie d'écrire $\lambda = \dfrac{y_Q - y_P}{x_Q - x_P}$ mais pour ça, il faut être certain que $x_P \neq x_Q$ pour ne pas diviser par 0. Or on se rappelle que dans l'énoncé, on a supposé que $x_P \neq x_Q$. On peut ainsi écrire

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}.$$

Maintenant qu'on connaît une expression de λ en fonction de x_P, y_P, x_Q et y_Q, on peut écrire, grâce à la première ligne du système, $\nu = y_P - \lambda x_P$.

Pour ne pas trop compliquer les calculs, dans un premier temps on ne remplace pas λ par son expression en fonction des différentes coordonnées mais uniquement ν. Ainsi, la droite (PQ) a pour équation $y = \lambda x + y_P - \lambda x_P = \lambda (x - x_P) + y_P$. Maintenant on remplace λ par son expression et on trouve

$$y = \frac{y_Q - y_P}{x_Q - x_P} (x - x_P) + y_P.$$

□

Pour calculer les coordonnées de $P \oplus Q$, on va devoir calculer l'intersection de la droite (PQ) et de la courbe elliptique. Pour cela, on va d'abord étudier quelques propriétés qui *a priori* n'ont aucun rapport mais qui vont nous permettre de simplifier les calculs à venir.

Exercice 20

Soit $p(x) = x^2 + ax + b$ un polynôme du second degré. On suppose qu'il a deux racines réelles α et β. Exprimer a et b en fonction de α et β.

Solution

On a supposé que p avait deux racines réelles α et β. On peut donc écrire $p(x) = (x - \alpha)(x - \beta)$. En développant cette expression, on obtient $p(x) = x^2 - \beta x - \alpha x + \alpha\beta = x^2 - (\alpha + \beta)x + \alpha\beta$. En utilisant la première expression de p, on trouve

$$p(x) = x^2 + ax + b = x^2 - (\alpha + \beta)x + \alpha\beta.$$

En identifiant, on obtient $a = -(\alpha + \beta)$ et $b = \alpha\beta$. □

Exercice 21

Cette fois on considère un polynôme de degré 3, $p(x) = x^3 + ax^2 + bx + c$. On suppose qu'il a trois racines réelles, α, β et γ. Comme précédemment, exprimer a, b et c en fonction de α, β et γ.

Solution

Comme avant, on écrit $p(x) = x^3 + ax^2 + bx + c = (x - \alpha)(x - \beta)(x - \gamma)$ et on développe le dernier membre. On trouve $p(x) = x^3 - (\alpha + \beta + \gamma)x^2 + (\alpha\beta + \alpha\gamma + \beta\gamma)x - \alpha\beta\gamma$ et en identifiant on obtient

$$\begin{aligned} a &= -(\alpha + \beta + \gamma) \\ b &= \alpha\beta + \alpha\gamma + \beta\gamma \\ c &= -\alpha\beta\gamma \end{aligned}$$

□

Exercice 22

On suppose toujours que P et Q sont deux points tels que $x_P \neq x_Q$. En utilisant l'exercice 19 et l'exercice 21, calculer les coordonnées du troisième point d'intersection de (PQ) et de la courbe. En déduire les coordonnées du point $P \oplus Q$, le symétrique du précédent par rapport à l'axe des abscisses.

Solution

On note I, de coordonnées (x_I, y_I), le troisième point d'intersection de (PQ) et de notre courbe elliptique. Cette intersection est définie par le système

$$\begin{cases} y^2 = x^3 + ax + b \\ y = \lambda x + \nu \end{cases}.$$

Dans notre cas, $\lambda = \left(\dfrac{y_Q - y_P}{x_Q - x_P}\right)$ et $\nu = y_P - \lambda x_P$, d'après l'exercice 19, mais on va faire tous les calculs en gardant λ et ν.

On reporte la deuxième ligne dans la première pour obtenir $(\lambda x + \nu)^2 = x^3 + ax + b$. On développe :

$$\lambda^2 x^2 + 2\lambda\nu x + \nu^2 = x^3 + ax + b,$$

ou encore, en arrangeant un peu les termes

$$x^3 - \lambda^2 x^2 + (a - 2\lambda\nu)x + b - \nu^2.$$

D'après l'exercice 21, le coefficient de x^2, qui vaut $-\lambda^2$, est aussi égal à l'opposé de la somme des racines de ce polynôme. Or les points P, Q et I appartiennent à l'intersection, donc leurs abscisses, x_P, x_Q et x_I sont racines de ce polynôme. On a donc $-\lambda^2 = -(x_P + x_Q + x_I)$, ce qui permet de trouver l'expression de x_I :

$$x_I = \lambda^2 - x_P - x_Q.$$

Comme le point I est sur la droite $y = \lambda x + \nu$, on trouve $y_I = \lambda x_I + \nu$, ou encore

$$y_I = \lambda(x_I - x_P) + y_P.$$

Maintenant pour obtenir le point $P \oplus Q$, il suffit de prendre le symétrique de I par rapport à l'axe des abscisses. C'est donc le point de coordonnées $(x_I, -y_I)$. □

8.4 Les cas qui demandent un peu plus d'imagination

Le cas sympathique étant résolu, on va maintenant attaquer les cas un peu moins évidents, qui vont demander un peu d'imagination. Par exemple, comment calculer $P \oplus P$? Il devient difficile de tracer la droite (PQ) lorsque P et Q sont en fait un seul et même point. Pour y remédier, on va considérer que dans ce cas, la droite considérée sera la tangente à la courbe en P. C'est raisonnable parce que si on fixe un point P sur la courbe et qu'on trace les droites (PQ) pour des points Q qui se rapprochent de P, ces droites se rapprochent de la tangente (Figure 7).

Exercice 23

Soient P et Q deux points de la courbe. On suppose que $y_P \neq 0$. On rappelle que la pente de la droite (PQ), pour $P \neq Q$, vaut $\dfrac{y_Q - y_P}{x_Q - x_P}$. En faisant « tendre » le point Q vers le point Q, déterminer la pente de la tangente à la courbe en P.

Solution

On part de $\dfrac{y_Q - y_P}{x_Q - x_P}$. Si on dit que y_Q tend vers y_P et que x_Q tend vers x_P, on obtient une forme indéterminée. On va donc manipuler un peu cette expression avant de passer à la limite. Comme nos points sont sur la courbe, ils vérifient l'équation $y^2 = x^3 + ax + b$. Si on réussi à faire apparaître y_P^2 ou y_Q^2, on pourra donc les remplacer par $x_P^3 + ax_P + b$ et $x_Q^3 + ax_Q + b$.

Pour faire apparaître y_P^2 et y_Q^2, on multiplie le numérateur et le dénominateur de l'expression de la pente par $y_Q - y_P$. On obtient

$$\frac{y_Q - y_P}{x_Q - x_P} = \frac{(y_Q - y_P)(y_Q + y_P)}{(x_Q - x_P)(y_Q + y_P)} = \frac{y_Q^2 - y_P^2}{(x_Q - x_P)(y_Q + y_P)}.$$

On remplace y_P^2 et y_Q^2 par leurs expressions en fonction de x_P et x_Q :

$$\begin{aligned}\frac{y_Q^2 - y_P^2}{(x_Q - x_P)(y_Q + y_P)} &= \frac{\left(x_Q^3 + ax_Q + b\right) - \left(x_P^3 + ax_P + b\right)}{(x_Q - x_P)(y_Q + y_P)} \\ &= \frac{\left(x_Q^3 - x_P^3\right) + a(x_Q - x_P)}{(x_Q - x_P)(y_Q + y_P)}\end{aligned}.$$

On a très envie de faire apparaître un facteur $x_Q - x_P$ dans l'expression $x_Q^3 - x_P^3$. Or on sait (ou on le retrouve par un calcul rapide) que $x_Q^3 - x_P^3 = (x_Q - x_P)\left(x_P^2 + x_P x_Q + x_Q^2\right)$. Donc

$$\begin{aligned}\frac{\left(x_Q^3 - x_P^3\right) + a(x_Q - x_P)}{(x_Q - x_P)(y_Q + y_P)} &= \frac{(x_Q - x_P)\left(x_P^2 + x_P x_Q + x_Q^2\right) + a(x_Q - x_P)}{(x_Q - x_P)(y_Q + y_P)} \\ &= \frac{x_P^2 + x_P x_Q + x_Q^2 + a}{(y_Q + y_P)}\end{aligned}.$$

Et cette fois si on fait tendre y_Q vers y_P et x_Q vers x_P, on obtient

$$\frac{x_P^2 + x_P x_P + x_P^2 + a}{(y_P + y_P)} = \frac{3x_P^2 + a}{2y_P}$$

□

Exercice 24

On suppose toujours $y_P \neq 0$. En déduire, en s'aidant de l'exercice 22, les coordonnées du point $P \oplus P$, défini comme le symétrique du second point d'intersection de la courbe avec la tangente à la courbe en P.

Solution

On commence comme dans l'exercice 22. Les coordonnées des points d'intersections $I = (x_I, y_I)$ de la courbe avec la tangente sont solutions du système

$$\begin{cases} y^2 = x^3 + ax + b \\ y = \lambda x + \nu \end{cases},$$

où cette fois λ est la pente de la tangente, trouvée à l'exercice 23. On y a aussi déterminé $\nu = y_P - \lambda x_P$ en utilisant que P était un point d'intersection de la courbe $y^2 = x^3 + ax + b$ avec la droite $y = \lambda x + \nu$. Cet argument reste vrai ici et donc ici aussi, $\nu = y_P - \lambda x_P$ (mais la valeur de λ a changé).

On reporte la deuxième ligne dans la première pour obtenir $(\lambda x + \nu)^2 = x^3 + ax + b$. Après développement et arrangement des termes, on obtient

$$x^3 - \lambda^2 x^2 + (a - 2\lambda\nu)\, x + b - \nu^2.$$

Le coefficient de x^2, qui vaut $-\lambda^2$, est aussi égal à l'opposé de la somme des racines de ce polynôme.

Dans l'exercice 22, on avait trois points d'intersections donc 3 racines pour le polynôme. Ici on ne connaît que deux points à l'intersection. En fait avec un dessin, on s'aperçoit qu'il n'y en a pas de troisième. On va donc considérer que P compte double, donc que les racines du polynôme sont x_P, x_I et encore x_P, ce qui va permettre de conclure. Ce n'est pas complètement artificiel : en effet, si on construit la tangente comme sur la figure 7, c'est à dire comme une sorte de limite de droites (PQ) quand Q tend vers P, alors à la limite le point Q va coïncider avec le point P. Comme Q était distinct de P, on peut considérer qu'à la limite, on a deux points P, ou encore que P compte double.

Les racines étant donc x_P, x_P et x_I, on trouve $-\lambda^2 = -(2x_P + x_I)$ et donc x_I :

$$\begin{aligned} x_I &= \lambda^2 - 2x_P \\ &= \left(\frac{3x_P^2 + a}{2y_P}\right)^2 - 2x_P. \end{aligned}$$

On en déduit l'ordonnée du point d'intersection, $y_I = \lambda x_I + \nu = \lambda(x_I - x_P) + y_P$. Et finalement, $P \oplus P$ a pour coordonnées $(x_I, -y_I)$. □

Dans l'exercice 22, on a supposé que P et Q étaient tels que $x_P \neq x_Q$. Dans l'exercice 23, on a supposé $P = Q$ mais que y_P était non nul.

Exercice 25

Quelle particularité a la droite considérée si les hypothèses précédentes ne sont pas vérifiées ?

Solution

Si P et Q sont distincts mais que $x_P = x_Q$, les deux points sont sur une droite verticale. De même, si P est un point de coordonnées $(x_P, 0)$, on peut se convaincre sur un dessin que la tangente à la courbe en P est verticale. □

8.5 Un point à l'infini ? ! ?

Pour définir l'addition de deux points distincts qui forment une droite verticale ou pour calculer $P \oplus P$ pour P de coordonnées $(x_P, 0)$, on va rajouter un point à la courbe, le *point à l'infini*. On décrète donc qu'en plus des points « normaux » de la courbe, on a un autre point, qu'on note $\mathcal{O}$, qui se trouve très très loin, à l'infini. On peut alors se convaincre avec un dessin (voir Figure 8) que c'est raisonnable de dire qu'une droite verticale coupe la courbe à l'infini et que les droites parallèles se coupent à l'infini (on peut faire tout ça de manière tout à fait rigoureuse et formelle mais c'est

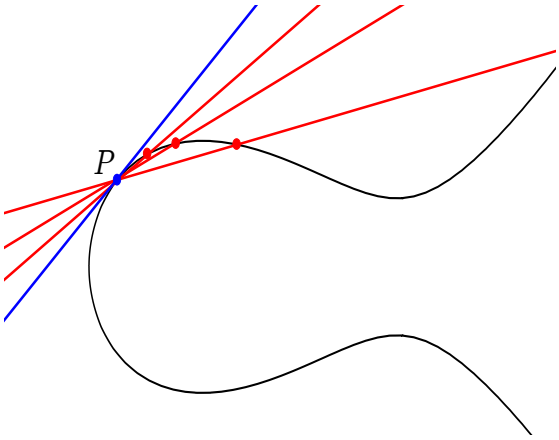


FIGURE 7 – On fixe un point P et on considère les droites passant par P et un autre point de la courbe proche de P. Plus le second point est proche de P, plus la droite est proche de la tangente à la courbe en P.

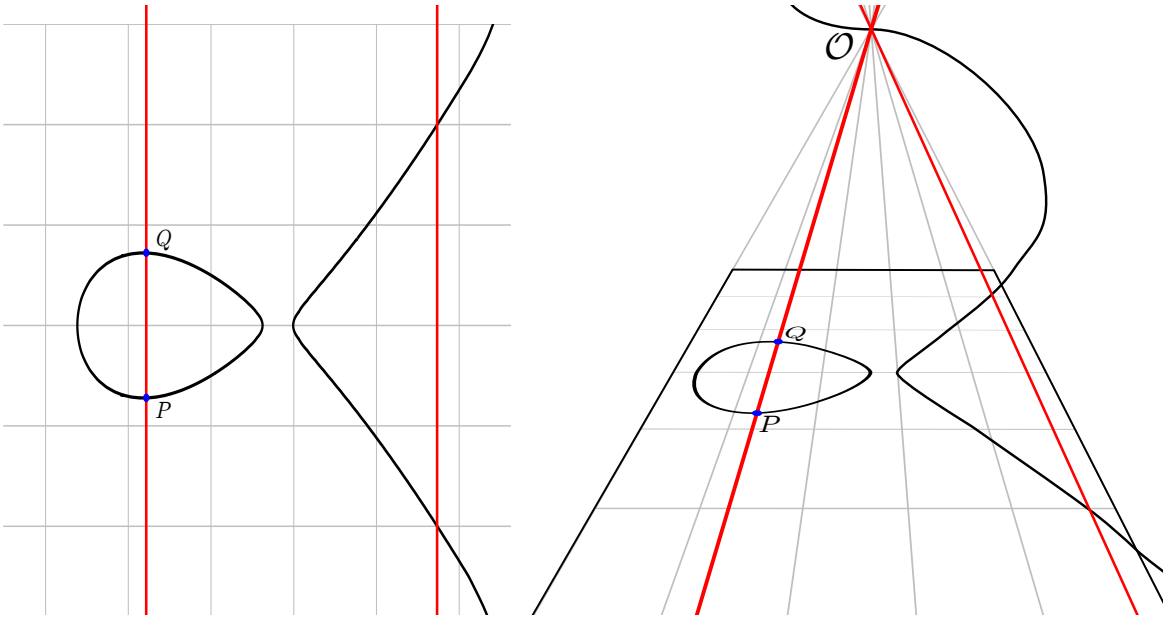


FIGURE 8 – On rajoute un point à l'infini.

beaucoup plus long). On va ensuite se convaincre que partir très très loin vers le haut ou vers le bas c'est la même chose, et donc que $\mathcal{O}$ est son propre « symétrique ».

Exercice 26

Soit P un point de la courbe. Que vaut $P \oplus \mathcal{O}$?

Solution

En s'aidant d'un dessin, par exemple la Figure 8, on voit que $P \oplus \mathcal{O} = \mathcal{O} \oplus P = P$. Le point à l'infini $\mathcal{O}$ est donc l'élément neutre de notre addition. □

Maintenant qu'on a notre élément neutre, on a envie de savoir calculer les opposés. On rappelle que l'opposé d'un point P est un point Q tel que $P \oplus Q = Q \oplus P = \mathcal{O}$. On note alors $Q = -P$.

Exercice 27

Soit P un point de la courbe, de coordonnées (x_P, y_P). Quelle sont les coordonnées de $-P$?

Solution

D'après ce qu'on a vu avant, le seul moyen de tomber sur le point à l'infini, c'est d'avoir une droite verticale. Donc $-P$ est le point de la courbe qui va former avec P une droite verticale. Si P a pour coordonnées (x_P, y_P) alors $-P$ a pour coordonnées $(x_P, -y_P)$. □

9 Les courbes elliptiques en pratique

9.1 Sommes de points

Maintenant qu'on sait comment ajouter des points, on va demander à l'ordinateur de le faire. Pour représenter le point à l'infini, on utilisera une liste de deux éléments, `[infini, infini]`.

Exercice 28

Écrire une fonction qui prend en argument deux points et une courbe elliptique (représentée par une liste de deux éléments) et qui renvoie la somme de ces deux points. On pensera à vérifier que les deux points donnés sont bien sur la courbe.

Solution

On commence par vérifier que les points sont bien sur la courbe grâce à la fonction `est_sur_la_courbe`. Puis en réutilisant la théorie vue précédemment, il suffit de bien distinguer les différents cas et de faire le calcul des coordonnées en conséquence.

```
def somme (P, Q, courbe) :

    # d'abord on regarde si P et Q sont sur la courbe
    if est_sur_la_courbe(P, courbe) == False
        raise ValueError('P n'est pas sur la courbe')
    if est_sur_la_courbe(Q, courbe) == False
        raise ValueError('Q n'est pas sur la courbe')

    # si un des points est le point a l'infini
    if P == [infini, infini] :
        return Q
    if Q == [infini, infini] :
        return P

    # sinon on donne des petits noms aux coordonnes
    xP = P[0];   yP = P[1];   xQ = Q[0];   yQ = Q[1]

    # cas qui donnent le point a l'infini
```

```
    if xP == xQ and yP == - yQ :
        return [infini, infini]

    # cas sympa ou P et Q sont distincts
    if xP <> xQ :
        pente = (yP - yQ )/(xP - xQ)
        xR = pente^2 - xP - xQ
        yR = - (yP + pente*(xR - xP))
        return [xR, yR]

    # cas P=Q avec yP=yQ<>0 (seul cas restant)
    if P == Q and yP <> 0 :
        pente = (3*xP^2 + courbe[0]) / (2*yP)
        xR = pente^2 - 2*xP
        yR = -(yP + pente*(xR - xP))
        return [xR,yR]
```

□

9.2 Calcul (plus ou moins) efficace de multiples

Maintenant qu'on sait calculer une somme, on va s'intéresser au calcul des multiples d'un point. Par définition, si k est un nombre entier positif, $k.P = \underbrace{P \oplus P \oplus \cdots \oplus P \oplus P}_{k \text{ fois}}$. Si $k = 0$, on définit $k.P = \mathcal{O}$ et si k est négatif, kP est l'opposé de $-kP$. Pour les applications qui nous intéressent en cryptographie, on va se contenter de calculer des multiples positifs.

Exercice 29

Écrire une fonction qui prend en entrée un entier strictement positif k un point P et une courbe elliptique et qui renvoie kP.

Solution

```
def multiple_naif (k, P, courbe) :

    if k == 1 :
        return P

    mult = P
    for i in range(1,k) :
        mult = somme(mult,P, courbe)
    return(mult)
```

□

Si on teste cette fonction avec quelques valeurs de k, on se rend compte que plus k est grand, plus le temps de calcul est long. Nous allons voir une seconde méthode, qui va demander moins de calculs.

L'idée vient de la remarque suivante : on considère un entier $k > 1$. Alors

- si k est pair, on note $k' = \dfrac{k}{2}$. Alors $kP = k'(2P)$. Il suffit donc de calculer $k'Q$ avec $Q = 2P$;
- si k est impair, $k-1$ est pair. On note $k'' = \dfrac{k-1}{2}$. Alors $kP = P \oplus (k''(2P))$. Il suffit donc de calculer $k''Q$ avec $Q = 2P$ et d'ajouter P au résultat.

On peut donc faire un calcul *récursif*.

Par exemple, pour calculer $13P$:

- $13P = P \oplus (12P)$. On fait donc une addition. Mais on ne connaît pas $12P$, il faut donc le calculer.

– $12P = 6.(2P)$. On note alors $Q = 2P = P \oplus P$, ce qui fait une addition à calculer, et on cherche à calculer $6Q$.
– $6Q = 3.(2Q)$. On calcule $R = 2Q = Q \oplus Q$, donc une addition, et on cherche à calculer $3R$.
– $3R = R \oplus (2R)$. On fait donc encore une addition et il reste à calculer $2R$.
– $2R = R \oplus R$, donc une dernière addition.

Finalement, on voit qu'on est capable de calculer $13P$ avec seulement 5 additions, alors qu'avec l'algorithme naïf, il en aurait fallu 12. En effet, dans cet exemple on a évité des calculs inutiles, en l'occurrence $3P$, $5P$, $6P$, $7P$, $9P$, $10P$ et $11P$.

Exercice 30

Écrire une fonction récursive de calcul efficace de multiple.

Solution

Il suffit de traduire l'explication précédente.

```
def multiple (k, P, courbe) :
  if k == 1 :
      return P

  elif k%2 == 0 :
      return multiple (k/2, somme(P, P, courbe), courbe)
  elif k%2 == 1 :

      return somme (
         multiple ((k-1)/2, somme(P, P, courbe), courbe)
         P,
         courbe)
```

□

9.3 Sur des corps finis ?

On a maintenant la possibilité de calculer rapidement les multiples d'un point sur une courbe elliptique. Mais si on calcule par exemple des multiples de l'ordre de 100 ou 1000, on remarque que la taille des coordonnées explose.

Un autre problème important est d'être capable de calculer un point sur une courbe, ce qui n'est pas évident en général mais absolument indispensable pour calculer des multiples (et donc faire de la cryptographie).

On peut remarquer que dans tous les calculs de sommes ou de multiples de points sur une courbe elliptique, les seules opérations utilisées sont du type « addition » ou « multiplication » (soustraire a c'est pareil qu'ajouter $-a$, c'est-à-dire ajouter l'opposé de a, donc c'est de type « addition ». De même, diviser par a, c'est multiplier par l'inverse de a, donc c'est de type « multiplication »). Il n'y a donc aucune opération « compliquée » comme extraire une racine carrée, calculer un logarithme, etc.

Donc si on dispose d'un ensemble sur lequel on peut mettre une addition et une multiplication qui lui confère une structure de corps, on peut utiliser toutes nos formules. En particulier, on a vu dans une autre activité que les corps finis étaient des corps (si on ne l'a pas vu, il suffit d'imaginer qu'il existe des ensembles de nombres qu'on peut multiplier, diviser (sauf par 0), additionner et soustraire, qui ont toutes les propriétés algébriques sympathiques des rationnels ou des réels, mais il n'y a qu'un nombre fini d'éléments dans ces ensembles. On peut aussi, en attendant les classes prépa ou la licence, consulter wikipédia).

Alors pourquoi remplacer les réels par les corps finis ? Essentiellement parce que la taille des nombres considérés reste bornée. En effet, dans le corps $\mathbb{F}_p$ à p éléments, après une réduction modulo p, grâce à une division euclidienne (ce qui est très rapide pour un ordinateur), on peut manipuler des nombres entre 0 et p. On s'affranchit donc du problème de la taille des nombres rencontrés sur les réels.

Pour dire à Sage qu'on veut travailler sur des corps finis, on procède de la manière suivante. On commence par prendre un nombre premier pour la taille du corps. Pour ça, on peut utiliser la commande `next_prime`. Elle prend en argument un entier k et renvoie le premier nombre premier strictement plus grand que k.

```
p = next_prime(8**40);p
```

```
1329227995784915872903807060280345027
```

Ensuite on donne un nom, par exemple K, au corps fini à p éléments que nous allons considérer. Pour Sage, le corps fini à p élément s'appelle `GF(p)` (pour l'anglais *Galois Field*, littéralement corps de Galois, mais qui signifie bien corps fini).

```
K = GF(p);K
```

```
Finite Field of size 1329227995784915872903807060280\
345027
```

Et à partir de maintenant, quand on manipule des courbes ou les coordonnées de points, on précise qu'on prend les éléments dans K :

```
courbe = [K(1),K(-1)];P = [K(1),K(1)];
print courbe, P
```

```
[1, 1329227995784915872903807060280345026] [1, 1]
```

On remarque que Sage a remplacé -1 par 1329227995784915872903807060280345026. En effet, ce dernier nombre est égal $p-1$, donc dans le corps à p éléments, où $p=0$, il est bien égal à -1. Pour ne pas confondre les nombres entiers et les éléments du corps, on peut utiliser la commande `base_ring` qui nous dit où vit l'élément.

```
x = K(42)
print x
print base_ring(x)
print base_ring(42)
```

```
42
Finite Field of size 1329227995784915872903807060280\
345027
Integer Ring
```

Ici, x vaut 42 dans le corps fini. Même s'il est noté 42 comme le 42 entier qu'on connaît bien, Sage sait qu'il vit dans le corps fini K. Et il se rappelle quand même que 42 sans autre précision est un entier (un élément de l'anneau des entiers relatifs, *Integer Ring* en anglais).

9.4 Et comment on trouve un point sur une courbe ?

On a réglé le problème de la taille des coefficients, mais il reste le problème de trouver des points sur la courbe. Pour ça, on sait faire des choses simples quand p est congru à 3 mod 4, ce qu'on suppose à partir de maintenant. En effet, on sait facilement extraire, si elle existe, une racine carrée. Plus précisément, soit a un élément dans notre corps fini. S'il existe un élément b tel que $a=b^2$ (ce qu'on sait facilement tester), alors on sait facilement calculer b ou $-b$, donc un élément qui élevé au carré donne a (en d'autres termes, une racine carrée).

Exercice 31

Comment utiliser l'extraction de racine carrée pour trouver un point sur la courbe ?

Solution

On peut agir de la manière suivante : on choisit un élément x au hasard dans le corps fini et on calcule x^3+ax+b. Il se trouve que si $x\neq 0$, on a environ une chance sur deux que ce soit un carré. Si c'en est un, on le garde, sinon on prend un nouveau x jusqu'à ce que x^3+ax+b soit un carré. Quand on a

bien un carré, on en prend une racine carrée et on l'appelle y. Alors $y^2 = x^3 + ax + b$ et donc le point (x, y) est un point de la courbe. □

Pour savoir si un élément $k \neq 0$ d'un corps fini à p élément est un carré, il faut calculer $k^{(p-1)/2}$. Si ça vaut 1 alors k est un carré, sinon ça vaut -1 et ce n'est pas un carré.

Si un élément k est un carré, on peut en trouver une racine carrée en calculant $k^{(p+1)/4}$.

Exercice 32

En utilisant les affirmations précédentes, écrire une fonction qui prend en argument un nombre premier p, une courbe elliptique sur le corps fini à p élément et qui renvoie un point de cette courbe.

Solution

On va commencer par tester si le nombre premier p est bien congru à 3 modulo 4. Dans le cas contraire, on renvoie une erreur. Sinon, on fait exactement ce qui est indiqué plus haut : on donne des valeurs aléatoires à `xP` grâce à la fonction `K.random_element()`, où `K` a été défini comme étant le corps fini à p éléments. Puis on regarde si `xP^3+axP+b` est un carré. Si c'est le cas, on en prend une racine carrée, on l'appelle `yP` et le point `[xP, yP]` est sur la courbe. Sinon, on redonne une valeur aléatoire à `xP` et on recommence, jusqu'à ce que `xP^3+axP+b` soit un carré.

```
def trouve_point(p, courbe) :
    # si p <> 3 mod 4, on renvoie une erreur
    if p%4 <> 3 :
        raise ValueError('trouve_point : p <> 3 modulo 4')

    a = courbe[0];   b = courbe[1]

    K = GF(p)
    pas_carre = True
    while pas_carre :
        xP = K.random_element()
        yPcarre = xP^3 + a*xP + b
        if yPcarre^((p-1)/2) == K(1):
            pas_carre = False
        else :
            pas_carre = True

    yP = (yPcarre)^((p+1)/4)

    return [xP, yP]
```

□

9.5 Synthèse

On a maintenant tous les outils à disposition pour écrire les « boîtes noires » utilisées au début de la partie 7.

Exercice 33

Écrire les fonctions suivantes :

- `param_publics` qui génère un point P, un nombre premier p congru à 3 modulo 4 et une courbe elliptique `courbe`, représentée par une liste de deux éléments ;
- `echange_public`, qui prend comme arguments un nombre entier a et les paramètres publics P, p et `courbe` et renvoie le multiple aP de P (calculé efficacement) ;
- `point_en_cle` qui prend un point en entrée et renvoie une clé sous forme de caractères ;

– `plus_un` qui prend en arguments les paramètres publics p, `courbe`, P et un point de la forme kP et renvoie $(k+1)P$.

Solution

Commençons par `param_publics`. Écrivons une fonction (sans argument) qui va générer un nombre premier p avec `next_prime`. Si $p \equiv 3 \mod 4$ c'est terminé, on renvoie p. Sinon, on va regarder le nombre premier qui suit p, en appelant `next_prime(p + 2)`.

Pour générer des nombres premiers, on choisit de prendre `next_prime` d'un nombre de la forme $9^a 8^b$ où a est un entier aléatoire entre 25 et 30 et b un entier aléatoire entre 10 et 15. C'est presque un choix complètement arbitraire. En effet, on aurait pu se contenter de choisir `next_prime` de 10^a avec a aléatoire. Mais dans ce cas, on aurait des nombres premiers d'une forme particulière (un 1 suivi de beaucoup de 0 suivi d'un 1, d'un 3 , d'un 7 ou encore d'un 9, ce qui n'est jamais bon en cryptographie). Par contre la taille des entiers a et b est arbitraire et on peut décider de prendre plus grand.

```
def genere_premier () :
  n = 9^ZZ.random_element(25,30)*8^ZZ.random_element(10,15)
  p = next_prime (n)
  while p%4 != 3 :
      p = next_prime (p + 2)
  return p
```

Pour générer la courbe, on va encore utiliser des nombres aléatoires. Ce ne sont plus des nombres entiers mais des éléments du corps fini $\mathbb{F}_p$. On choisit donc deux valeurs aléatoires a et b dans $\mathbb{F}_p$ et on regarde si la liste `[a,b]` correspond bien à une courbe elliptique, grâce à notre fonction `courbe_elliptique`. Si c'est le cas c'est terminé, on renvoie `[a,b]`, sinon on recommence (on sait qu'avec un grand nombre premier p et a et b aléatoires, la probabilité que la courbe $y^2 = x^3 + ax + b$ soit elliptique est très élevée).

```
def genere_courbe (p) :
    a = GF(p).random_element()
    b = GF(p).random_element()
    while courbe_elliptique([a,b]) == False :
        a = GF(p).random_element()
        b = GF(p).random_element()
    return [a,b]
```

Pour trouver un point sur la courbe, on dispose de la fonction `trouve_point` codée précédemment.

Il reste maintenant à regrouper toutes ces fonctions. Pour plus de lisibilité, en plus de renvoyer les paramètres, on va les afficher de manière agréable.

```
def param_public () :
   p = genere_premier ()
   courbe = genere_courbe (p)
   P = trouve_point ( courbe, p )

   print 'nombre premier :', p
   print 'point :', P
   print 'courbe : y^2 = x^3 +', courbe[0],'x + ', courbe[1]

   return p, P, courbe
```

Si tout va bien, le lecteur attentif a compris que les fonctions `echange_public` et `plus_un` ne sont rien d'autre que le calcul de multiple et le calcul de somme.

```
def echange_public (a, P, p, courbe) :
    return multiple(a, P, courbe)
```

```
def plus_un (p, courbe, P, kP) :
    return somme(P, kP, courbe)
```

Pour finir, on va transformer un point (en pratique ce sera le point abP commun à Alice et Bob, qui correspond à une clé secrète) en une clé composée de caractères.

Pour cela, on considère la première coordonnée du point qu'on va transformer en nombre entier. En effet, même si cette première coordonnée ressemble à un nombre entier, il ne faut pas oublier que c'est un élément du corps fini $\mathbb{F}_p$. La fonction `ZZ` permet de considérer l'entier qui a la même écriture. Ensuite on convertit cet entier en liste de nombres, chacun compris entre 0 et 27 à l'aide de `.digits(27)` (cette fonction renvoie la liste des digits de l'écriture du nombre en base 27). On fait la même chose pour la seconde coordonnée. Puis on concatène les deux listes et on les convertit en chaîne de caractères, grâce à notre fonction `numlist2string`.

```
def point_en_cle (point) :
    liste_a = ZZ(point[0]).digits(27)
    liste_b = ZZ(point[1]).digits(27)
    return numlist2string(a + b)
```

□

Voilà, la partie sur les courbes elliptiques est terminée, on a maintenant compris comment fonctionnait l'échange de clé sécurisé et en plus on a été capable de le programmer.

9.6 Pour aller (encore) plus loin

On a donc vu plusieurs aspects de la cryptographie. Les exemples de cryptosystèmes à clé secrète étudiés ne sont plus utilisés actuellement pour des données sensibles. En revanche, ceux utilisés aujourd'hui, pour des applications civiles mais aussi militaires, sont basés sur l'utilisation d'une clé secrète comme dans le chiffrement de Vigenère. Les différentes opérations permettant de chiffrer un message sont beaucoup plus complexes mais le principe est bien là. Le lecteur intéressé pourra taper « schéma de Feistel », « DES » (pour Data Encryption Standard), « AES » (pour Advanced Encryption Standard, actuellement approuvé par la NSA pour les informations top secrètes) dans son moteur de recherche préféré pour de plus amples informations.

Pour l'échange de clé sécurisé, le principe que nous avons étudié est l'échange de type Diffie-Hellman. Vous l'utilisez tous les jours si vous allez sur Internet. L'usage des courbes elliptiques n'est pas le plus répandu mais est au cœur des recherches actuelles en cryptographie asymétrique. En effet, les courbes elliptiques permettent d'avoir une sécurité équivalente à RSA avec des tailles de clés sensiblement plus petites, ce qui peut être très intéressant lorsque la mémoire est limitée, par exemple sur une carte à puce.

Pour finir et pour ne pas oublier que souvent, en sortant des sentiers battus, on peut découvrir des moyens de casser des cryptosystèmes très sécurisés, on conseil au lecteur de faire des recherches (sur Internet par exemple) sur les attaques par canal auxiliaire, où l'on voit qu'on peut retrouver une clé en analysant la consommation ou les émanations électromagnétiques du matériel.