SÉCURISER LES COMMUNICATIONS AU SEIN D'UNE VILLE :

LA CRYPTOGRAPHIE ASYMÉTRIQUE

À L'AIDE DE COURBES ELLIPTIQUES

ZAIDAN OSCAR

17673

PLAN DU TIPE

I/ Théorie

- Cryptographie symétrique et asymétrique
- Courbe elliptique
- Courbe elliptique sur $\mathbb{Z}/p\mathbb{Z}$
- Crypto système El-Gamal

II/ Mise en pratique

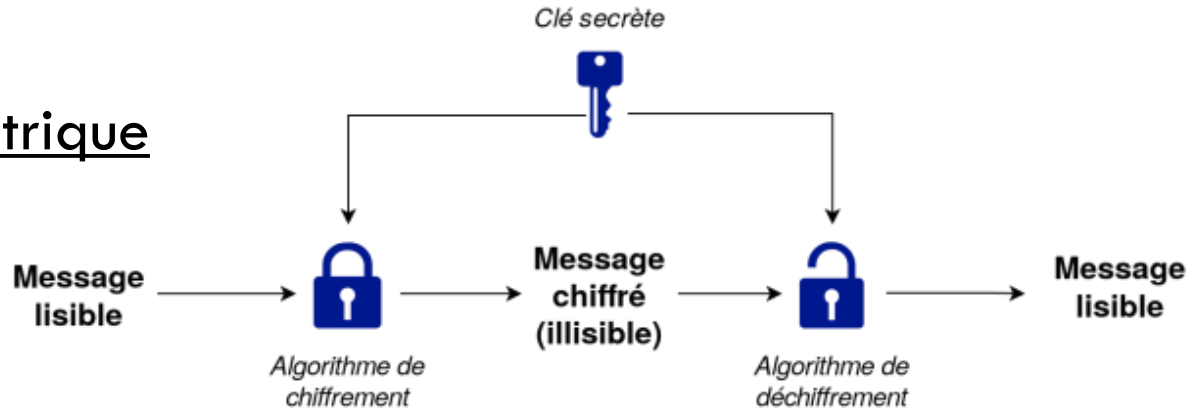
- Implémentation des courbes
- Implémentation de El-Gamal

III/ Conclusion

- Analyse sécurité
- Dans la pratique
- Comparaison au système RSA

I/ CRYPTOGRAPHIE SYMÉTRIQUE ET ASYMÉTRIQUE

Symétrique



Asymétrique

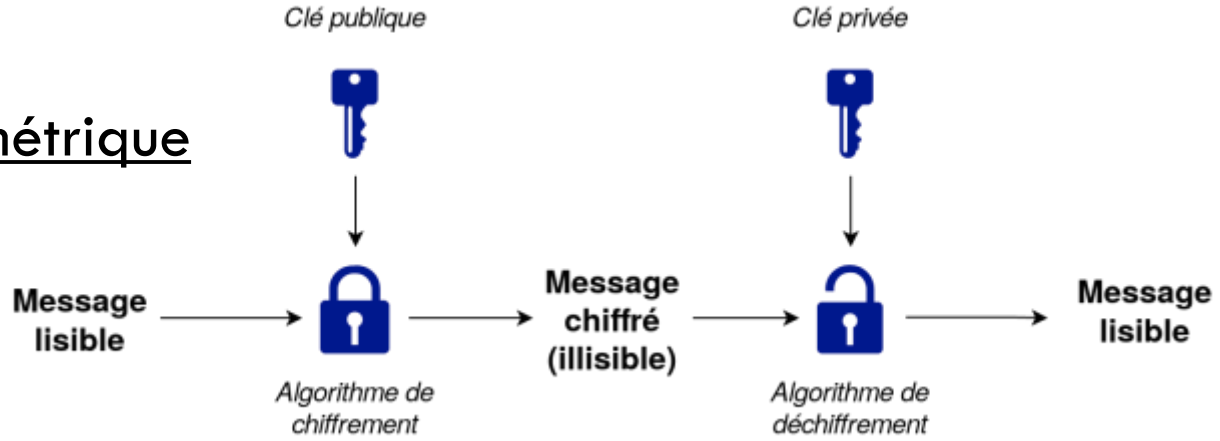
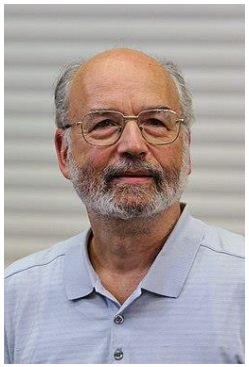


image : librecours.net

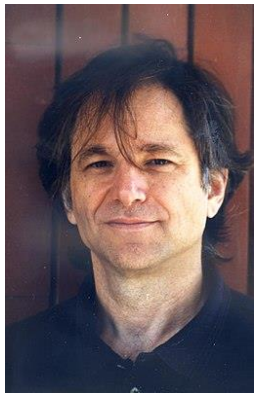
I/ CRYPTOGRAPHIE ASYMÉTRIQUE

RSA : Rivest, Shamir Adleman.

Rivest



Shamir



Adleman

El Gamal : sur courbe elliptique

Taher El Gamal

image : wikipedia.org

I/ COURBES ELLIPTIQUES

Equation de Weierstrass :

Soit K un corps

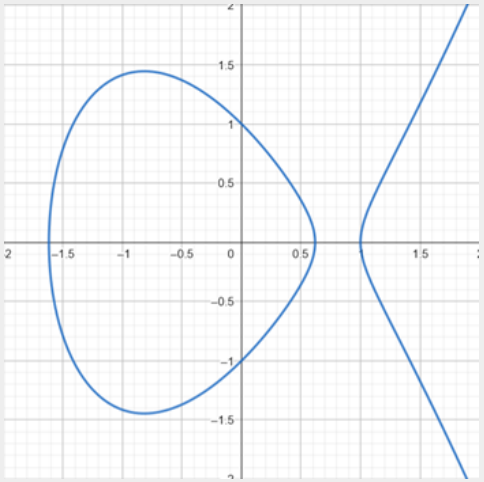
$$E : y^2 = x^3 + Ax + B \qquad A \in K, B \in K$$

Lisse si $\Delta = -16\,(\,4A^3 + 27B^2\,) \neq 0$

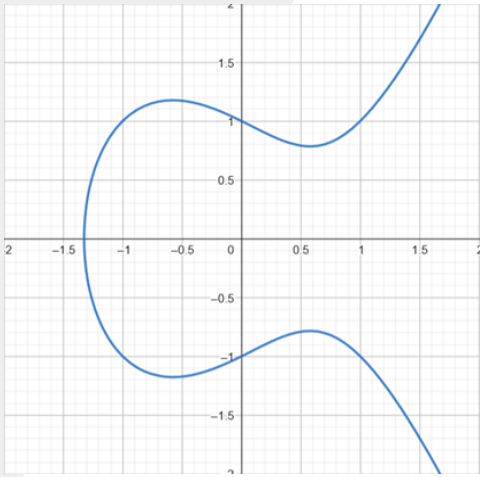
I/ COURBES ELLIPTIQUES

Courbe Elliptique : $y^2 = x^3 + ax + b$

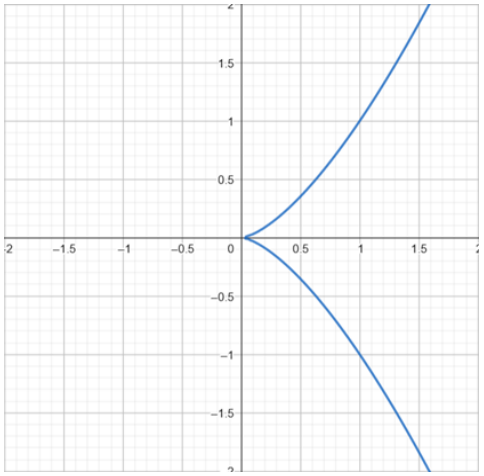
$\Delta = -16\,(\,4a^3 + 27b^2\,)$



$\Delta > 0\ (a = -1, b = 2)$



$\Delta < 0\ (a = -1, b = 1)$



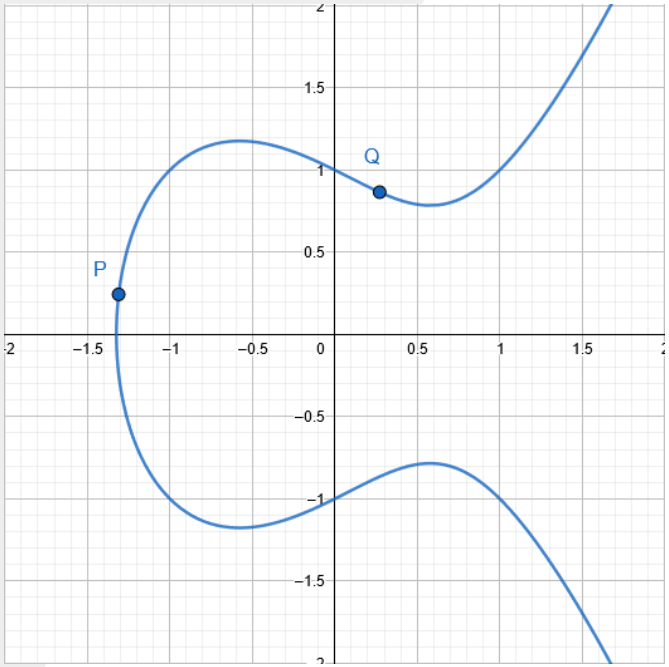
$\Delta = 0\ (a = 0, b = 0)$

courbes : générées sur GeoGebra

I/ COURBES ELLIPTIQUES

STRUCTURE DE GROUPE

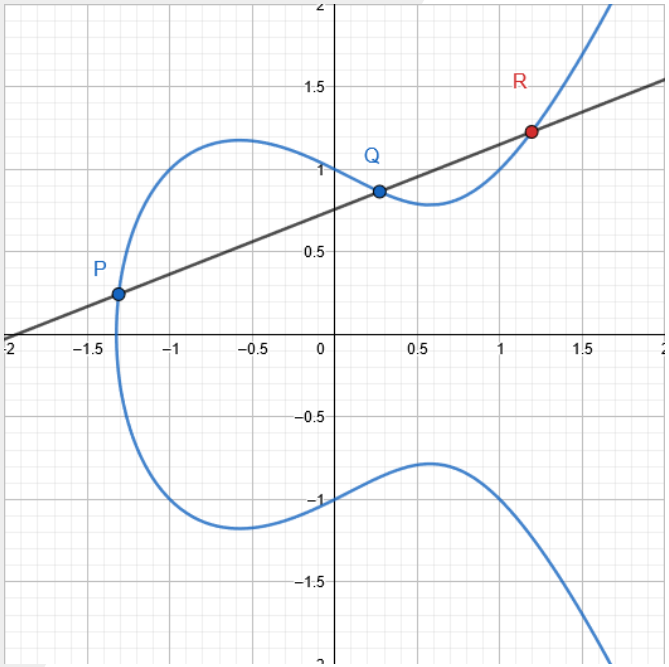
2 Points distincts



I/ COURBES ELLIPTIQUES

STRUCTURE DE GROUPE

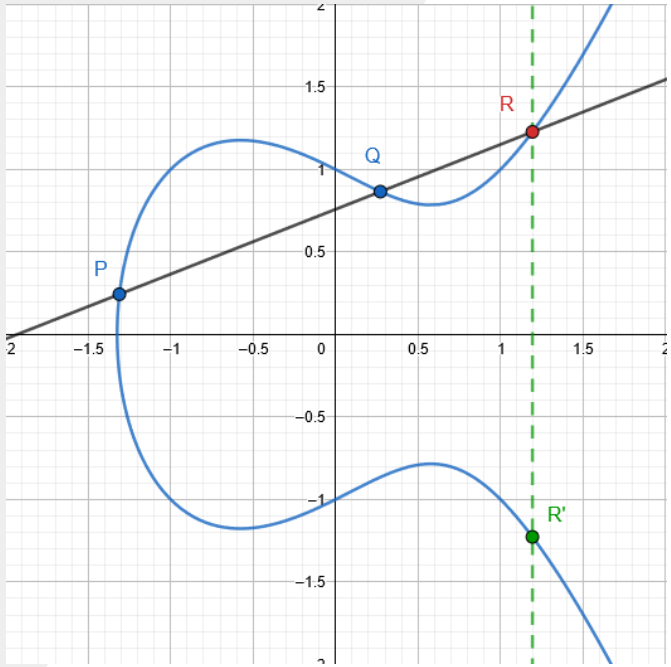
2 Points distincts



I/ COURBES ELLIPTIQUES

STRUCTURE DE GROUPE

2 Points distincts

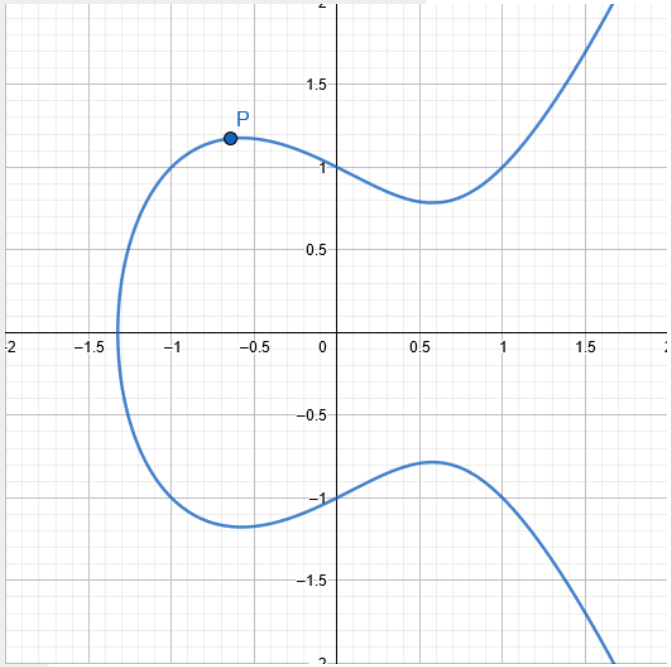


P + Q = R’

I/ COURBES ELLIPTIQUES

STRUCTURE DE GROUPE

Point à l'infini



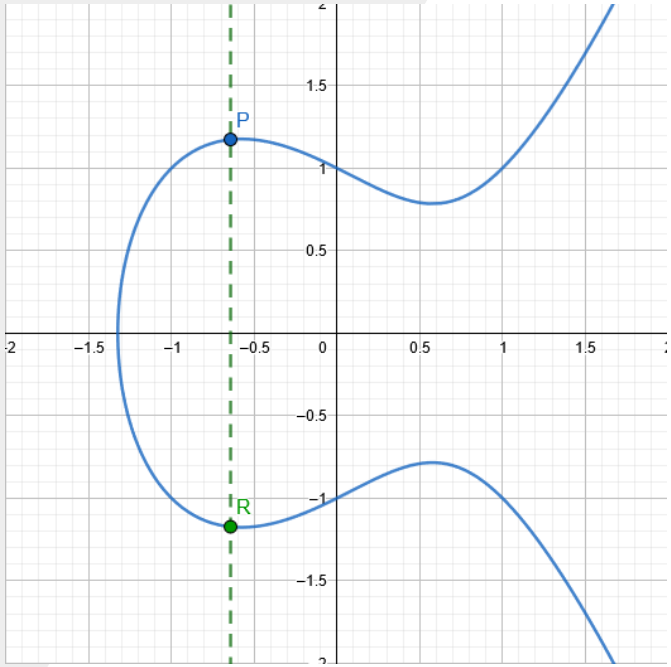
I/ COURBES ELLIPTIQUES

STRUCTURE DE GROUPE

Point à l'infini

$P + O = P$

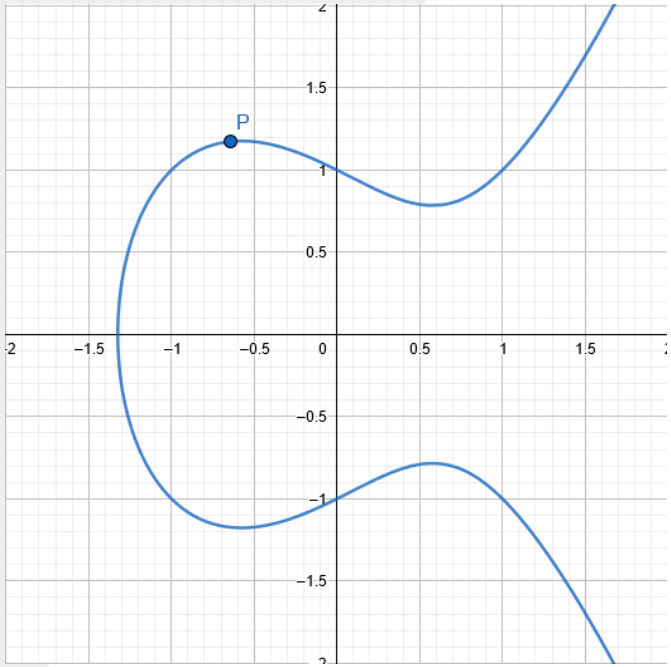
$-P = R$



I/ COURBES ELLIPTIQUES

STRUCTURE DE GROUPE

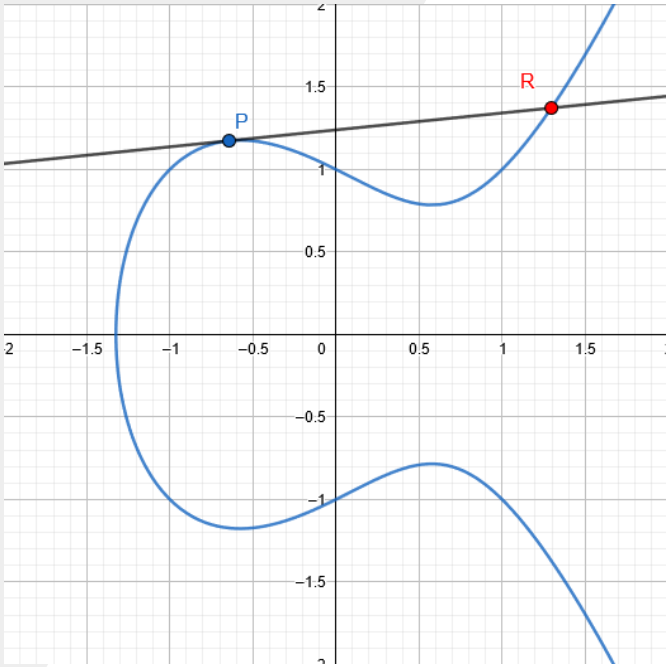
Double d'un point



I/ COURBES ELLIPTIQUES

STRUCTURE DE GROUPE

Double d'un point

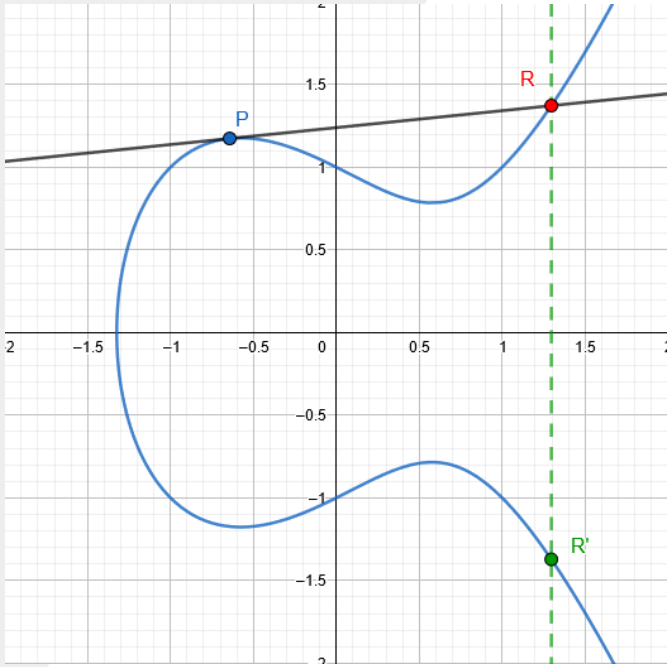


I/ COURBES ELLIPTIQUES

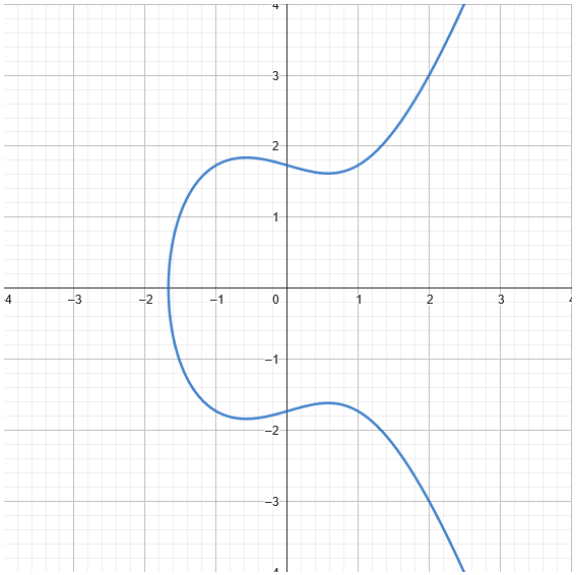
STRUCTURE DE GROUPE

Double d'un point

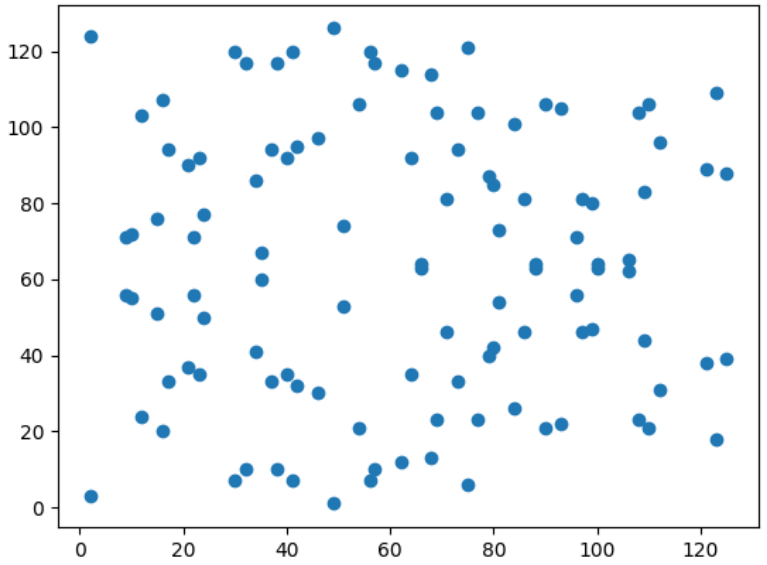
$P + P = R'$



I/ COURBES SUR $\mathbb{Z}/p\,\mathbb{Z}$



$y^2 = x^3 - x + 3$

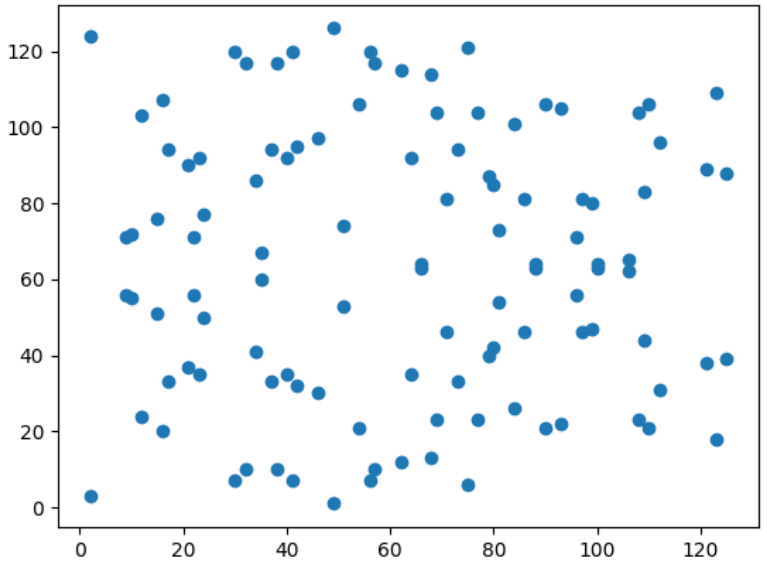


$y^2 = x^3 - x + 3$ sur $\mathbb{Z}/127\,\mathbb{Z}$

I/ COURBES SUR $\mathbb{Z}/p\,\mathbb{Z}$

Soit $a, b \in \mathbb{N}$, $p \in \mathbb{P}$ et $p \equiv 3[4]$

$$C = \{ (x, y) \epsilon (\mathbb{Z}/p\,\mathbb{Z})^2 \mid y^2 = x^3 + ax + b \}$$



$y^2 = x^3 - x + 3$ sur $\mathbb{Z}/127\,\mathbb{Z}$

I/ COURBES SUR $\mathbb{Z}/p\,\mathbb{Z}$

Génération des points : trouver les racines carrées dans $\mathbb{Z}/p\mathbb{Z}$

Symbole de Legendre

$$\begin{cases} \left(\frac{x}{p}\right) = 0 \; si \; x|p \\ \left(\frac{x}{p}\right) = 1 \; si \; \exists k \in \mathbb{N}^*, k^2 \equiv x \; [p] \\ \left(\frac{x}{p}\right) = -1 \; sinon \end{cases}$$

$$\left(\frac{x}{p}\right) \equiv x^{\frac{p-1}{2}} \; [p]$$

Inverse modulaire :
(si x premier avec p)

- $\exists u, v \in \mathbb{Z}, \; ux + vp = 1$
- $x^{-1} = u \; , \; ux \equiv 1[p]$

I/ COURBES SUR $\mathbb{Z}/p\,\mathbb{Z}$

Génération des points : trouver les racines carrées dans $\mathbb{Z}/p\mathbb{Z}$

$$\left(\frac{x}{p}\right) \equiv x^{\frac{p-1}{2}}\ [p]$$

Si $p \equiv 3[4]$ alors $\frac{p-1}{2}$ est impair

Bézout :

$$\exists u, v\ \in \mathbb{Z}\,, 2u + \frac{p-1}{2}v = 1$$

Soit x tel que $\left(\frac{x}{p}\right) = 1$

Posons $y = x^u$

Alors

$$y^2 \equiv x^{2u}\ [p]$$

$$y^2 \equiv x^{1-\frac{p-1}{2}v}\ [p]$$

$$y^2 \equiv x^1 x^{\left(\frac{p-1}{2}\right)^{-v}}\ [p]$$

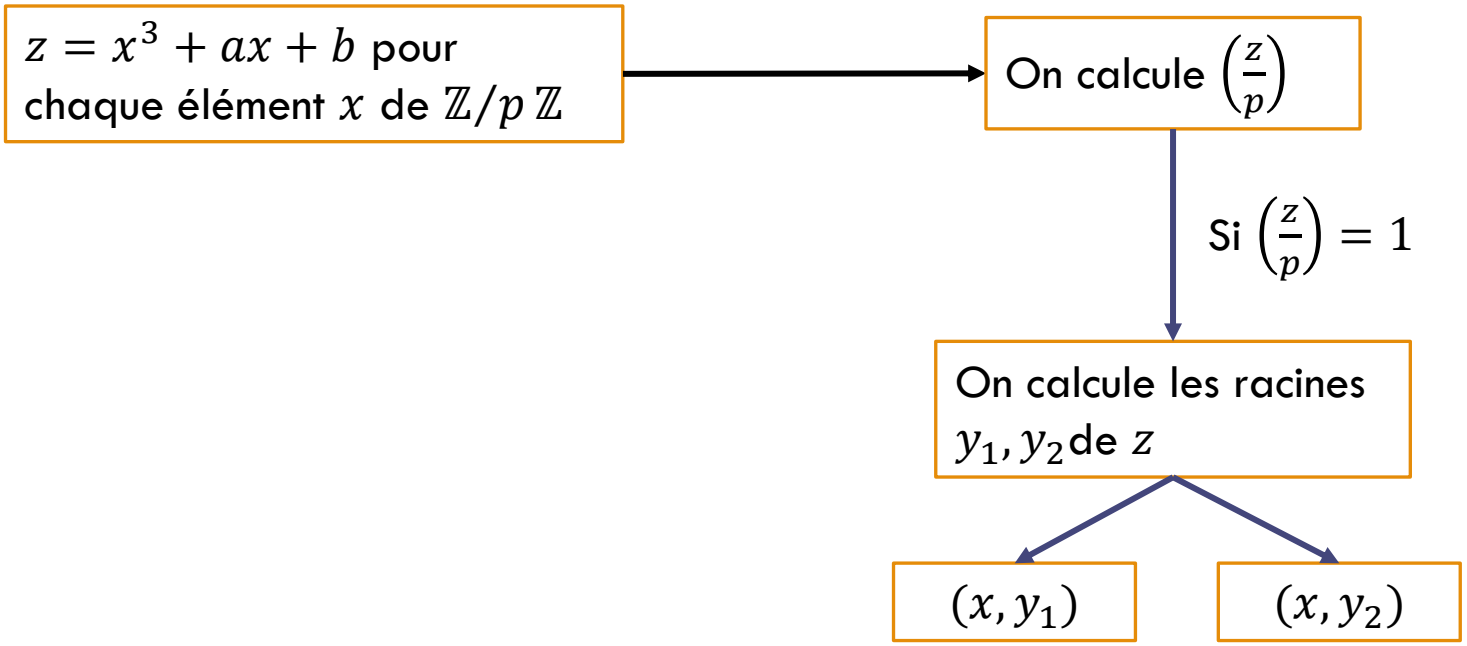
$$\boxed{y^2 \equiv x\ [p]}$$

I/ COURBES SUR $\mathbb{Z}/p\,\mathbb{Z}$

Génération des points : trouver les points de la courbe

$$y^2 = x^3 + ax + b\ [p]$$

Obtention des points de la courbe :



I/ UTILISATION : CRYPTO SYSTÈME EL-GAMAL

Clé publique : Courbe elliptique $\mathcal{E}$ sur $\mathbb{Z}/p\mathbb{Z}$, un point G d'ordre n un point $P = qG$

Clé privée : un entier $q, 1 \leq q < n$

Message : un point $M \in < G >$

Chiffrement

- Choix d'un entier $0 \leq k < n$
- Calcul de
  - $C_1 = kG$
  - $C_2 = M + kP$
- Message : (C_1, C_2)

Déchiffrement

- Calcul de $qC_1 = qkG = kP$
- Alors $C_2 - kP = M$

II/ IMPLÉMENTATION : COURBES ELLIPTIQUES

Objets

- Point
- Courbe

Fonctions

- Trouver des points
- Addition
- Multiplication par un entier

II/ IMPLÉMENTATION : COURBES ELLIPTIQUES

En pratique : en Python

Class Point

```
infinity_point = Point(True) # Point à l'infini

my_point = Point(False, 5, 2) # Point de coordonnées (5, 2)
```

- Est infini
- Coordonnées

Class Curve

```
"""Courbe d'équation y^2 = x^3 - x + 3 mod 127"""
my_curve = Curve(-1, 3, 127)
```

- Trouver des points
- Addition
- Multiplication par un entier

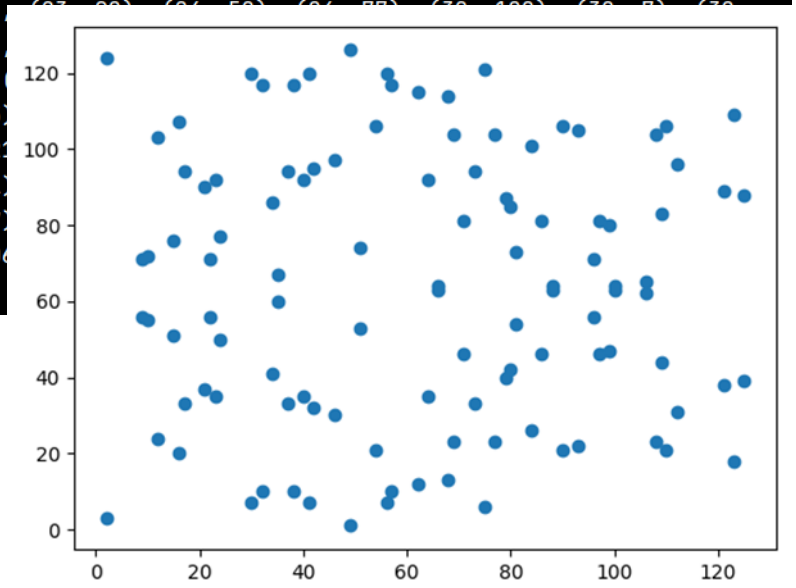
II/ IMPLÉMENTATION : COURBES ELLIPTIQUES

Génération des points

```
>>> my_curve = Curve(-1, 3, 127)
```

```
>>> my_curve.generate_points()
[(2, 124), (2, 3), (9, 71), (9, 56), (10, 72), (10, 55), (12, 103), (12, 24), (15, 76), (15, 51), (16, 107), (16, 20), (17,
 94), (17, 33), (21, 37), (21, 90), (22, 71), (22, 56), (23, 35)
117), (32, 10), (34, 41), (34, 86), (35, 60), (35, 67), (37, 94)
120), (41, 7), (42, 32), (42, 95), (46, 30), (46, 97), (49, 1),
120), (56, 7), (57, 117), (57, 10), (62, 115), (62, 12), (64, 35
 104), (69, 23), (71, 81), (71, 46), (73, 94), (73, 33), (75, 12
 42), (80, 85), (81, 73), (81, 54), (84, 26), (84, 101), (86, 81
 22), (93, 105), (96, 71), (96, 56), (97, 81), (97, 46), (99, 47
(108, 104), (108, 23), (109, 44), (109, 83), (110, 21), (110, 10
 (123, 109), (125, 88), (125, 39)]
```

```
>>> my_curve.show()
```



$y^2 = x^3 - x + 3$ sur $\mathbb{Z}/127\,\mathbb{Z}$

II/ IMPLÉMENTATION : COURBES ELLIPTIQUES

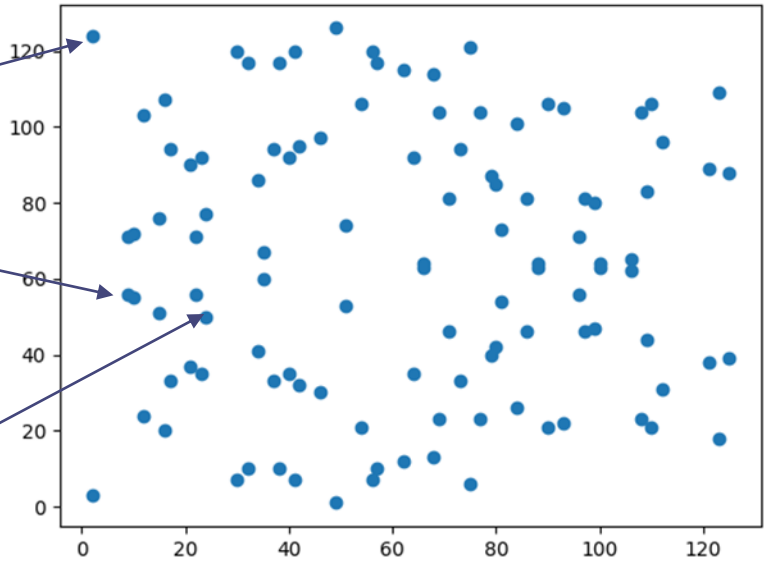
Addition

```
>>> my_curve = Curve(-1, 3, 127)
```

```
>>> p1 = Point(False, 2, 124)
>>> p2 = Point(False, 57, 10)
```

```
>>> my_curve.belongs_to_curve(p1)
True
>>> my_curve.belongs_to_curve(p2)
True
```

```
>>> my_curve.add(p1, p2)
(22, 56)
```



$y^2 = x^3 - x + 3$ sur $\mathbb{Z}/127\,\mathbb{Z}$

II/ IMPLÉMENTATION : COURBES ELLIPTIQUES

Multiplication par un entier

```
>>> my_curve = Curve(-1, 3, 127)
```

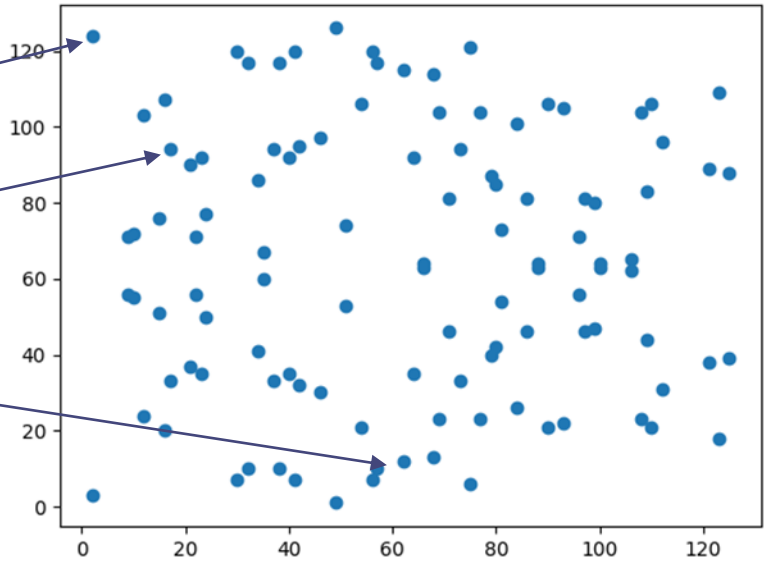
```
>>> p = Point(False, 2, 124)
```

```
>>> my_curve.multi(2, p)
(17, 94)
>>> my_curve.multi(5, p)
(62, 12)
```

```
>>> my_curve.get_order(p)
110
```

```
>>> my_curve.multi_fast(5, p)
(62, 12)
```

$O(\log n)$



$y^2 = x^3 - x + 3$ sur $\mathbb{Z}/127\,\mathbb{Z}$

II/ IMPLÉMENTATION : EL-GAMAL

Clé publique : Courbe elliptique $\mathcal{E}$ sur $\mathbb{Z}/p\mathbb{Z}$, un point G d'ordre n un point $P = qG$

Clé privée : un entier $q, 1 \leq q < n$

Message : un point $M \in < G >$

Fonctions :

- Génération des clés
- Chiffrement
- Déchiffrement

Choix des paramètres

- Courbe $\varepsilon: y^2 = x^3 - x + 3 \, mod \, 7919$

```
>>> my_curve = Curve(-1, 3, 7919)
```

- Un point de la courbe : $G: (1, 3023)$

```
>>> g = Point(False, 1, 3023)
```

- G est d'ordre $n = 1299$

```
>>> my_curve.get_order(g)
1299
```

- On choisit $q = 500$

```
>>> q = 500
```

II/ IMPLÉMENTATION : EL-GAMAL

Clé publique : Courbe elliptique $\mathcal{E}$ sur $\mathbb{Z}/p\mathbb{Z}$, un point G d'ordre n
un point $P = qG$
Clé privée : un entier $q, 1 \leq q < n$
Message : un point $M \in < G >$

Création des clés

```
def create_keys(a: int, b: int, p: int, g: Point, order: int, q: int):
```

```
>>> private_key, public_key = create_keys(-1, 3, 7919, Point(False, 1, 3023), 1299, 500)
```

```
>>> public_key
((Curve : y^2 = x^3 + -1 * x + 3 mod 7919), (1, 3023), 1299, (5279, 7180))
```

Courbe ε G n $P = qG$

```
>>> private_key
500
```

q

II/ IMPLÉMENTATION : EL-GAMAL

Clé publique : Courbe elliptique $\mathcal{E}$ sur $\mathbb{Z}/p\mathbb{Z}$, un point G d'ordre n
un point $P = qG$
Clé privée : un entier $q, 1 \leq q < n$
Message : un point $M \in < G >$

Chiffrement d'un point

```
>>> public_key
((Curve : y^2 = x^3 + -1 * x + 3 mod 7919), (1, 3023), 1299, (5279, 7180))
```

Courbe ε G n $P = qG$

```
>>> msg = my_curve.multi_fast(52, g)
>>> msg
(5800, 460)
```

```
>>> coded_msg = encryption(public_key, msg)
>>> coded_msg
((4117, 5975), (305, 6787))
```

C_1 C_2

Chiffrement

- Choix d'un entier $0 \leq k < n$
- Calcul de
  - $C_1 = kG$
  - $C_2 = M + kP$
- Message : (C_1, C_2)

II/ IMPLÉMENTATION : EL-GAMAL

Clé publique : Courbe elliptique $\mathcal{E}$ sur $\mathbb{Z}/p\mathbb{Z}$, un point G d'ordre n
un point $P = qG$
Clé privée : un entier $q, 1 \leq q < n$
Message : un point $M \in < G >$

Déchiffrement d'un point

```
>>> public_key
((Curve : y^2 = x^3 + -1 * x + 3 mod 7919), (1, 3023), 1299, (5279, 7180))
```

Courbe ε G n $P = qG$

```
>>> msg = my_curve.multi_fast(52, g)
>>> msg
(5800, 460)
```

```
>>> decoded_msg = decryption(private_key, public_key, coded_msg)
>>> decoded_msg
(5800, 460)
```

Déchiffrement

- Calcul de $qC_1 = qkG$
- Alors $C_2 - kP = M$

II/ IMPLÉMENTATION : EL-GAMAL

Clé publique : Courbe elliptique $\mathcal{E}$ sur $\mathbb{Z}/p\mathbb{Z}$, un point G d'ordre n
un point $P = qG$
Clé privée : un entier $q, 1 \leq q < n$
Message : un point $M \in < G >$

Echange d'un message

```
>>> msg = "Le theme de l'année est la ville !"
```

```
>>> coded_msg = send_msg(public_key, msg)
>>> coded_msg
[((5450, 3717), (3740, 5255)), ((6495, 2637), (3118, 3063)), ((2722, 2548), (185, 6938)), ((6936, 2962), (75
 5995)), ((1309, 4788), (5745, 3341)), ((5745, 3341), (3118, 3063)), ((3630, 2374), (4901, 4873)), ((1135, 6
 (7818, 2860)), ((4850, 3068), (6931, 6782)), ((1800, 164), (1498, 1754)), ((6370, 4303), (1260, 2566)), ((5
 7815), (3329, 7650)), ((364, 2859), (7287, 7197)), ((1152, 467), (5450, 3717)), ((825, 5361), (7818, 5059))
 6430), (5485, 3265)), ((3740, 5255), (1952, 6284)), ((6984, 1454), (5769, 2287)), ((5392, 1441), (7456, 686
 ((4476, 6478), (2094, 5123)), ((6909, 4267), (3312, 5941)), ((6371, 6637), (6936, 4957)), ((709, 7273), (56
```

```
>>> decoded_msg = receive_msg(public_key, private_key, coded_msg)
>>> decoded_msg
"Le theme de l'année est la ville !"
```

III/ ANALYSE SÉCURITÉ : PROBLÈME DU LOGARITHME DISCRET

Baby step giant step : une attaque possible

Entrée : $< g > d'ordre\ n, x \in < g >$
Sortie : $k\ tel\ que\ x = kg$

Division euclidienne : $k = im + j, avec\ m = \lceil \sqrt{n} \rceil$

$$jg = x + i(-gm)$$

Algorithme:

- Calcule liste L : $(j, jg),\ 0 \leq j < m$
- Calcule des $(i,\ x + i(-gm))$
  - Cherche si $x + i(-gm))$ est dans L
- Retourner $im + j$

$$0 \leq i, j < m$$

Complexité : $O(\sqrt{n})$

III/ BABY STEP GIANT STEP

Entrée : $< g > d'ordre\ n, x \in < g >$
Sortie : $k\ tel\ que\ x = g^k$

```
>>> my_curve = Curve(-1, 3, 7919)
>>> g = Point(False, 1, 3023)
>>> P = my_curve.multi_fast(500, g)
```

$P = 500G$

```
>>> baby_step_giant_step(my_curve, g, 1299, P)
500
```

```
>>> decoded_msg_attack = attack(public_key, coded_msg)
>>> decoded_msg_attack
"Le theme de l'année est la ville !"
```

III/ DANS LA PRATIQUE

Génération de courbes :

- Choix d'un point
- Construction de la courbe

Conversion en point de la courbe

Maël Feurgard

- ENS Lyon
- Stage Kalay:
  Développement de librairies cryptographiques optimisées pour processeur VLIW Coolidge

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

III/ DANS LA PRATIQUE

Taille des clés

Table 1. Comparable Key Size (in bits) [3]

Symmetric Algorithms	ECC	RSA
80	163	1024
112	233	2240
128	283	3072
192	409	7680
256	571	15360

ECC : Elliptic curve cryptography

Génération des clés

Table 2. Key Generation Performance [3]

Key Length (bits)		Time (s)	
RSA	ECC	RSA	ECC
1024	163	0.16	0.08
2240	233	7.47	0.18
3072	283	9.80	0.27
7680	409	133.90	0.64
15360	571	679.06	1.44

Source : N. Jansma and B. Arrendondo,"Performance Comparison of Elliptic Curve and RSA Digital Signatures", 2004

III/ DANS LA PRATIQUE

Chiffrement

Longueur de clé		Temps de calcul (s)	
ECC	RSA	ECC	RSA
163	1024	0.15	0.01
233	2240	0.34	0.15
283	3072	0.59	0.21
409	7680	1.18	1.53
571	15360	3.07	9.20

Déchiffrement

Longueur de clé		Temps de calcul (s)	
ECC	RSA	ECC	RSA
163	1024	0.23	0.01
233	2240	0.51	0.01
283	3072	0.86	0.01
409	7680	1.80	0.01
571	15360	4.53	0.03

Chiffres issus du NIST (National Institute of Standards and Technology)

III/ CONCLUSION

- Crypto système robuste
- Tailles des clés réduites
- Temps génération des clés réduit
- Augmentation du temps de déchiffrement
- Utile pour les communications sensibles au sein d'une ville

BIBLIOGRAPHIE

[1] HÄGLER MICHAEL : *Courbes elliptiques et cryptographie :*
https://math.univ-bpclermont.fr/~rebolledo/page-fichiers/projetMichael.pdf

[2] UNIVERSITÉ DE BORDEAUX : *Racine de Z/nZ :*
https://www.math.u-bordeaux.fr/~jcouveig/cours/CSI4.pdf

[3] CÉCILE GONÇALVES : *Cryptographie Avancée, Courbes elliptique :*
https://www.lix.polytechnique.fr/Labo/Cecile.Goncalves/Downloads/Cours2_Courbes_elliptiques.pdf

[4] Wikipedia: *Baby-step giant-step :*
https://fr.wikipedia.org/wiki/Baby-step_giant-step

ANNEXE

COURBES ELLIPTIQUES

Equation de Weierstrass

Soit K un corps, $E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \ , a_i \epsilon K$

Equation lisse : $\begin{cases} a_1y = 3x^2 + 2a_2x + a_4 \\ 2y + a_1x + a_3 = 0 \end{cases}$ pas de solution

Si K caractéristique différente 2 et 3

$$E : y^2 = x^3 + Ax + B$$

Lisse si $\Delta = -16\,(\,4A^3 + 27B^2\,) \neq 0$

COURBES SUR $\mathbb{Z}/p\,\mathbb{Z}$

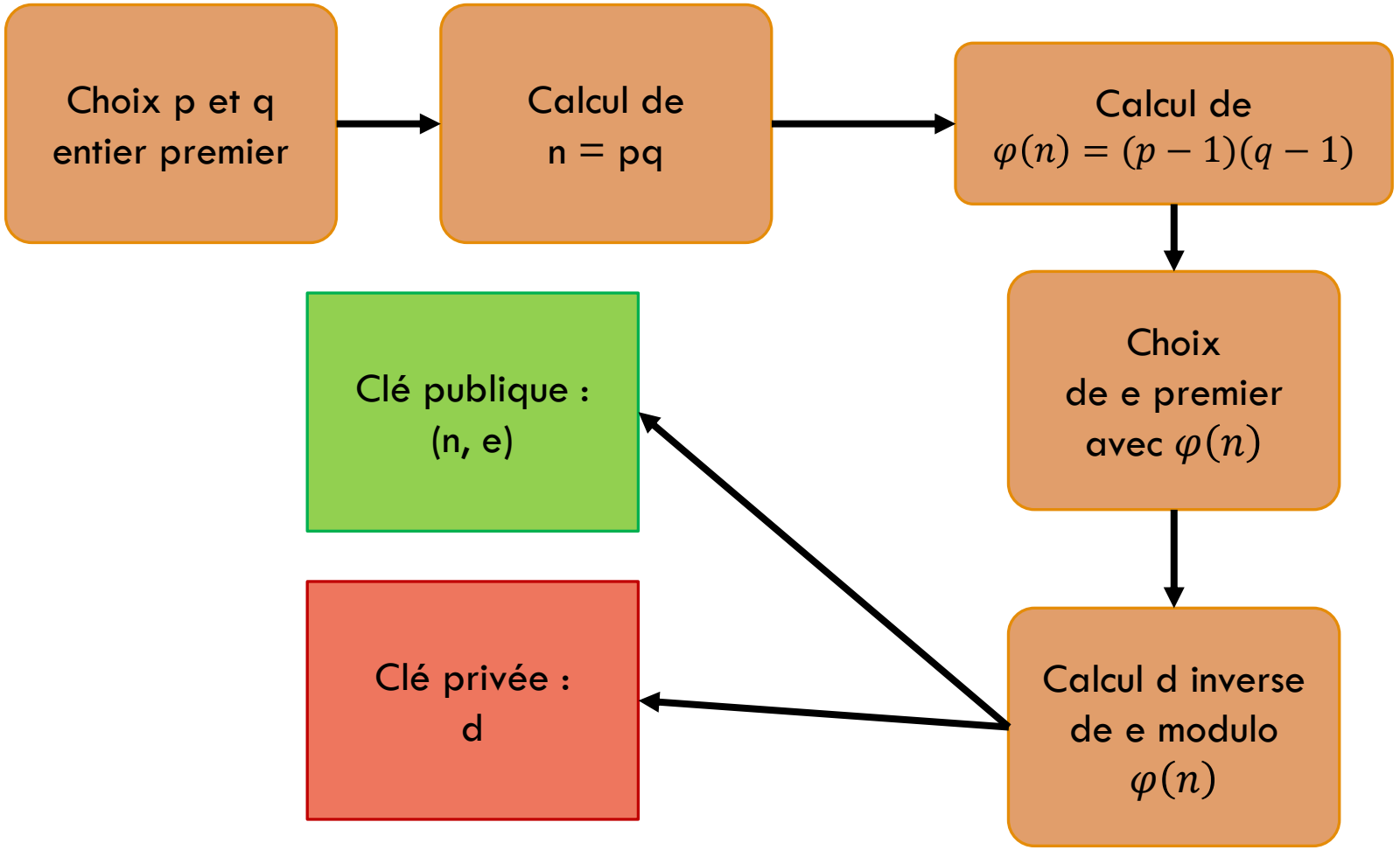
Génération des points : trouver les racines carrées dans $\mathbb{Z}/p\mathbb{Z}$

Exemple :

Posons $x = 15, p = 127$ $(127 \equiv 3[4])$

- $\frac{127-1}{2} = 63$
- $-31 \times 2 + 1 \times 63 = 1\ donc\ u = -31, v = 1$
- $15^{63} \equiv 1[127]\ donc\ \left(\frac{15}{127}\right) = 1$
- $y = 15^{-31}$
- $y^2 = 15^{2\times-31} = 15^{-62} = 15^{1-63\times1} = 15 \times (15^{63})^{-1} \equiv 15[127]$

RSA : CRÉATION DES CLÉS



RSA : CHIFFREMENT / DÉCHIFFREMENT

Message : entier m < n

Clé publique :
(n, e)

Clé privée :
d

Chiffrement :
$C \equiv m^e \ [n]$

Déchiffrement :
$m \equiv C^d \ [n]$

CODE

PUISSANCE MODULAIRE

```python
def modular_power (x, n, q):
    """
    :param x: nombre à mettre à la puissance n modulo q
    :param n: puissance
    :param q: modulo
    :return: x^n mod q
    """
    e = 0
    res = 1
    while e != n:
        res = (res * x)% q
        e += 1
    return res
```

DIVISION MODULAIRE

```python
def modular_division(a: int, b: int, q: int):
    """
    :param a:
    :param b:
    :param q:
    :return: a / b mod p
    """
    a = a % q
    b = b % q
    acc = b
    res = 1
    while a != acc:
        acc = (acc + b) % q
        res += 1
    return res
```

Soient a, b, p trois entiers
a/b [p]
y tel que

$$y * b \equiv a \ [p]$$

TROUVER LES ENTIER QUI SONT DES CARRÉS

```python
def find_square(p: int):
    """
    :param p: a primary integer, with q = 3 mod 4
    :return: tab array of integers that are square
    """
    power = int((p - 1) / 2)
    tab = []
    for i in range(p):
        if modular_power(i, power, p) == 1:
            tab.append(i)
    return tab
```

Symbole de Legendre

$$\begin{cases} \left(\frac{x}{p}\right) = 0 \; si \; x|p \\ \left(\frac{x}{p}\right) = 1 \; si \; \exists k \in \mathbb{N}^*, k^2 \equiv x \; [p] \\ \left(\frac{x}{p}\right) = -1 \; sinon \end{cases}$$

$$\left(\frac{x}{p}\right) \equiv x^{\frac{p-1}{2}} \; [p]$$

ALGORITHME D'EUCLIDE ÉTENDUE

```python
def extended_euclid_rec(a: int, b: int):
    """
    this function don't return pgcd(a,b) because we will us it only for a, b such as pgcd(a, b) = 1
    :param a: an integer
    :param b: an integer
    :return: (u, v) such as u*a + v*b = pgcd(a, b)
    """
    if b == 0:
        return (1, 0)
    else:
        u, v = extended_euclid_rec(b, a % b)
        return (v, u - (a // b) * v)
```

Exemple: a = 37, b = 15

$37 * 1 + 15 * 0 = 37$ (1)
$37 * 0 + 15 * 1 = 15$ (2)
$37 - 2 * 15 = 4$ (3) = (1)-2(2)
$-2 * 32 + 5 * 15 = 1$ (4) = (2) –2(3)

TROUVER UNE RACINE

```python
def find_root(x: int, p: int):
    """
    :param x: an integer : we suppose that x is a square
    :return: y such as y^2 = x
    """
    u, v = extended_euclid_rec(2, int((p - 1) / 2))
    if u < 0:
        u = p - 1 + u
    y = modular_power(x, u, p)
    return y
```

CLASS POINT

```python
class Point:
    def __init__(self, is_inf, x=0, y=0):
        self.is_inf = is_inf
        self.x = x
        self.y = y

    def __hash__(self):
        return hash((self.x, self.y, self.is_inf))

    def __eq__(self, other):
        if (self.is_inf or other.is_inf):
            return self.is_inf and other.is_inf
        else:
            return self.x == other.x and self.y == self.y

    def __repr__(self):
        if self.is_inf:
            return "inf"
        else:
            return "(" + str(self.x) + ", " + str(self.y) + ")"

    def __str__(self):
        if self.is_inf:
            return "inf"
        else:
            return "(" + str(self.x) + ", " + str(self.y) + ")"


    9 usages (9 dynamic)
    def print_point(self):
        print("(", self.x, ",", self.y, ")")

    2 usages (2 dynamic)
    def inv(self):
        return Point(self.is_inf, self.x, - self.y)
```

CLASS CURVE 1/6

```python
class Curve:
    def __init__(self, a: int, b: int, q: int):
        """
        The equation of the curve is y^2 = x^3 + a*x + b mod q
        :param a:
        :param b:
        """
        self.a = a
        self.b = b
        self.q = q

    def __repr__(self):
        return "(Curve : y^2 = x^3 + " + str(self.a) + " * x + " + str(self.b) + " mod " + str(self.q) + ")"

    def __str__(self):
        return "(Curve : y^2 = x^3 + " + str(self.a) + " * x + " + str(self.b) + " mod " + str(self.q) + ")"

    1 usage
    def belongs_to_curve(self, p: Point):
        """

        :param p: a point
        :return: if p is a point of the curve
        """
        if p.is_inf:
            return True
        return (p.y ** 2) % self.q == (p.x ** 3 + self.a * p.x + self.b) % self.q

```

CLASS CURVE 2/6

```
    2 usages
34  def generate_points(self):
35      """
36      This function generates all the point of the elliptic curve y^2 = x^3 + a*x + b mod p
37      """
38      points = []
39      power = int((self.q - 1) / 2)
40      for i in range(self.q):
41          z = ((i ** 3) + self.a * i + self.b) % self.q
42          if modular_power(z, power, self.q) == 1:
43              root = find_root(z, self.q)
44              points.append((i, root))
45              points.append((i, self.q - root))
46      return points
47
    5 usages
48  def generate_points_object(self):
49      """
50      This function generates all the point of the elliptic curve y^2 = x^3 + a*x + b mod p
51      """
52      points = []
53      power = int((self.q - 1) / 2)
54      for i in range(self.q):
55          z = ((i ** 3) + self.a * i + self.b) % self.q
56          if modular_power(z, power, self.q) == 1:
57              root = find_root(z, self.q)
58              points.append(Point(False, i, root))
59              points.append(Point(False, i, self.q - root))
60      return points
61
```

CLASS CURVE 3/6

```
1 usage
def get_a_point(self, n: int):
    """
    :return
    :param n:
    :return:
    """
    power = int((self.q - 1) / 2)
    for i in range(n,self.q):
        z = ((i ** 3) + self.a * i + self.b) % self.q
        if modular_power(z, power, self.q) == 1:
            root = find_root(z, self.q)
            return Point(False, i, root)
    return Point(True)

```

CLASS CURVE 4/6

Courbe : $y^2 = x^3 + ax + b\ [p]$

```
16 usages (2 dynamic)
def add(self, p1: Point, p2: Point):
    """

    :param p1: A point on the curve
    :param p2: An other point on the curve
    :return: p = p1 + p2
    """
    if p1.is_inf:
        return p2
    elif p2.is_inf:
        return p1
    elif p1.x != p2.x:
        m = modular_dividion((p2.y - p1.y), (p2.x - p1.x), self.q)
        x3 = (m ** 2 - p1.x - p2.x) % self.q
        y3 = (m * (p1.x - x3) - p1.y) % self.q
        return Point(False, x3, y3)
    elif p1.x == p2.x and p1.y != p2.y:
        return Point(True)
    elif (p1.x, p1.y) == (p2.x, p2.y) and p1.y != 0:
        m = modular_dividion((3 * (p1.x ** 2) + self.a), (2 * p1.y), self.q)
        x3 = (m ** 2 - 2 * p1.x) % self.q
        y3 = (m * (p1.x - x3) - p1.y) % self.q
        return Point(False, x3, y3)
    else:
        return Point(True)
```

$P_1 = (x_1, y_1), P_2 = (x_2, y_2),$
$P_1 + P_2 = P_3 = (x_3, y_3)$

- Si $P_1 = O$
  $P_3 = P_2$
- Si $x_1 \neq x_2$

$x_3 = m^2 - x_1 - x_2$
$y_3 = m(x_1 - x_3)$
$m = \frac{y_2 - y_1}{x_2 - x_1}$

- Si $x_1 = x_2\ et\ y_1 \neq y_2$
  $P_3 = O$
- Si $P_1 = P_2\ et\ y_1 \neq 0$

$x_3 = m^2 - 2x_1$
$y_3 = m(x_1 - x_3) - y_1$
$m = \frac{3x_1^2 + a}{2y_1}$

- Si $P_1 = P_2$
  $P_3 = O$

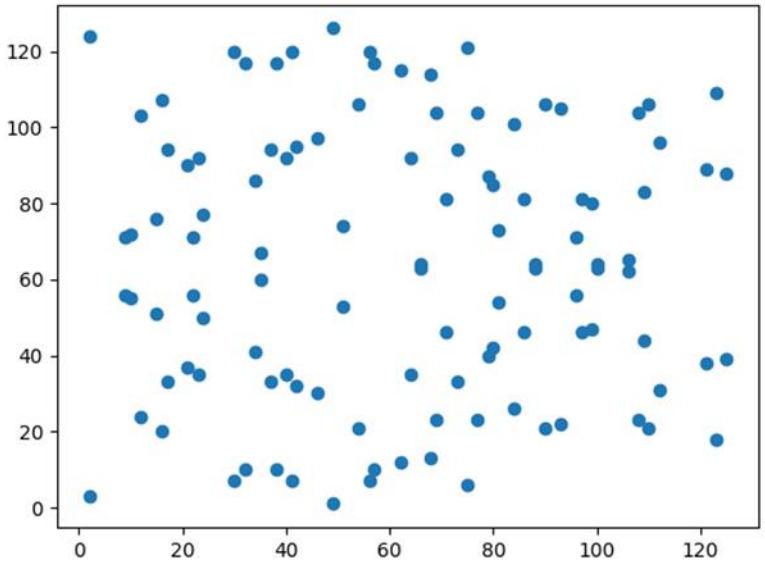
CLASS CURVE 5/6

```
4 usages
def multi(self, n: int, p: Point):
    """
    :param n: integer assumed to be positive
    :param p: a point
    :return: n * p
    """
    res = Point(True)
    for i in range(n):
        res = self.add(res, p)
    return res

6 usages (3 dynamic)
def multi_fast(self, n: int, p: Point):
    """

    :param n: integer assumed to be positive
    :param p: a point
    :return: n * p
    """
    if n == 0:
        return Point(True)
    res = p
    acc = Point(True)
    while n > 1:
        if n % 2 == 1:
            acc = self.add(acc, res)
            n = n - 1
        else:
            res = self.add(res, res)
            n = int(n / 2)
    return self.add(res, acc)
```

$$Puissance(x,n) = \begin{cases} x, & si\ n{=}1 \\ puissance(x^2, \frac{n}{2}), & si\ n \text{ Pair} \\ x * puissance(x^2, \frac{n-1}{2}), & si\ n \text{ Pair} \end{cases}$$

CLASS CURVE 6/6

```
        2 usages
133     def get_order(self, p: Point):
134         n = 0
135         save = p
136         while not p.is_inf:
137             p = self.add(p, save)
138             n += 1
139         return n
140
        2 usages
141     def show(self):
142         tab = self.generate_points()
143         tabx = [i[0] for i in tab]
144         taby = [i[1] for i in tab]
145         plt.scatter(tabx, taby)
146         plt.show()
147
```



$y^2 = x^3 - x + 3$ sur $\mathbb{Z}/127\,\mathbb{Z}$

CRÉATION DES CLÉS

```python
def create_keys(a: int, b: int, p: int, g: Point, order: int, q: int):
    """

    :param a:
    :param b:
    :param p:
    :param g: generating point
    :param order: the order of the point g
    :param q: an integer such as 0 < n < order
    :return:
    """
    curve = Curve(a, b, p)
    h = curve.multi_fast(q, g)
    public_key = (curve, g, order, h)
    private_key = q
    return private_key, public_key
```

Clé publique : Courbe elliptique $\mathcal{E}$ sur $\mathbb{Z}/p\mathbb{Z}$, un point G d'ordre n un point $P = qG$

Clé privée : un entier $q, 1 \leq q < n$

Message : un point $M \in < G >$

CHIFFREMENT ET DÉCHIFFREMENT D'UN POINT

```python
def encryption(public_key, msg: Point):
    """
    This function encrypt a msg with the public key of the person we want to send the message
    :param public_key: the public key contains: the curve on which we are working, a point p, the order of p,
    h another point
    :param msg: the message we want to send that has been converted in a point of the elliptic curve
    :return:
    """
    curve, p, order, h = public_key
    k = randint(0, order)
    c1 = curve.multi_fast(k, p)
    c2 = curve.add(msg, curve.multi_fast(k, h))
    return c1, c2


2 usages
def decryption(private_key, public_key, msg_encrypted):
    curve, p, order, h = public_key
    c1, c2 = msg_encrypted
    return curve.add(c2, curve.multi_fast(private_key, c1).inv())
```

Chiffrement

- Choix d'un entier
  $0 \leq k < n$
- Calcul de
  - $C_1 = kG$
  - $C_2 = M + kP$
- Message : (C_1, C_2)

Déchiffrement

- Calcul de
  $qC_1 = qkG = kP$
- Alors
  $C_2 - kP = M$

CONSTRUCTION DES DICTIONNAIRES

```
1 usage
def construct_dic_p_to_c(p: Point, c: Curve):
    """
    :param p: p is assumed to have an order greater than 256
    :param c: p is a point of c
    :return: dictionary point to character
    """
    dic = {}
    it = p
    for i in range(256):
        dic[it] = chr(i)
        it = c.add(it, p)
    return dic


1 usage
def construct_dic_c_to_p(p: Point, c: Curve):
    """
    :param p: p is assumed to have an order greater than 256
    :param c: p is a point of c
    :return: dictionary character to point
    """
    dic = {}
    it = p
    for i in range(256):
        dic[chr(i)] = it
        it = c.add(it, p)
    return dic
```

TRANSFERT DE MESSAGES

```python
def send_msg(public_key, msg):
    l = []
    length = len(msg)
    point_dic = construct_dic_c_to_p(public_key[1], public_key[0])
    for i in range(length):
        # chaque charatere est representé par le ieme point de la courbe
        l.append(encryption(public_key, point_dic[msg[i]]))
    return l


def receive_msg(public_key, private_key, encrypted_msg):
    res = ""
    length = len(encrypted_msg)
    dic = construct_dic_p_to_c(public_key[1], public_key[0])
    for i in range(length):
        p = decryption(private_key, public_key, encrypted_msg[i])
        res = res + dic[p]
    return res
```

BABY STEP GIANT STEP : UNE ATTAQUE POSSIBLE

```python
def baby_step_giant_step(p: Point, c: Curve, g: Point, n: int):
    """
    :param p:
    :param c:
    :param g:
    :param n: g is of order n
    :return: k such as kg = p
    """
    m = round(sqrt(n)) + 1
    d = {}
    new_p = Point(True)
    for j in range(m):
        d[new_p] = j
        new_p = c.add(new_p, g)
    inv = (c.multi(m, g)).inv()
    y = p

    for i in range(m+1):
        j = search(y, d)
        if j == -1:
            y = c.add(y, inv)
        else:
            return i*m + j
```

Complexité en temps : $O(\sqrt{n})$
Complexité en espace : $O(\sqrt{n})$

Algorithm 1 Baby step, giant step

Require: un groupe cyclique G d'ordre n, un générateur g et un element b
Ensure: une valeur x vérifiant $g^x = b$

1: $m \leftarrow \sqrt{n} + 1$
2: **for** j=1, m **do** ▷ baby steps
3: sauvgarder dans un tableau (j, g^j)
4: Calculer g^{-m}
5: $y \leftarrow 1$
6: **for** i=0, m-1 **do** ▷ Giant steps
7: **if** y est le second composant d une paire dans le tableau **then**
8: **return** $im + j$
9: **else** $y \leftarrow yg^{-m}$

```python
1 usage
def search(p: Point, d):
    try:
        return d[p]
    except:
        return -1
```

UTILISATION DE BABY STEP GIANT STEP

```python
def attack(public_key, coded_msg):
    c, g, n, p = public_key
    q = baby_step_giant_step(c, g, n, p)
    return receive_msg(public_key, q, coded_msg)
```

BABY STEP GIANT STEP DANS $\mathbb{Z}/p\mathbb{Z}$

```python
def baby_step_giant_step(q, g, b):
    """
    :param q: on travaille dans Z/qZ
    :param g: g est un générateur de Z/qZ
    :param b: g^x : inconnu
    :return: x tel que g^x = b mod q
    """
    m = int(ceil(sqrt(q)))
    tab = []
    """Baby step"""
    for j in range(1, m+1):
        tab.append((j, modular_power(g, j, q)))

    y = b
    inv = modular_power(g, (q - m - 1), q)
    """Giant step"""
    for i in range(m+1):
        j = recherche(tab, y)
        if j == -1:
            y = y * inv % q
        else:
            return i*m + j
```

BABY STEP GIANT STEP: GROUPE MULTIPLICATIFE

Entrée : $< g > d'ordre\ n, x \in < g >$
Sortie : $k\ tel\ que\ x = g^k$

Division euclidienne : $k = im + j, avec\ m = \lceil \sqrt{n} \rceil$

$$g^j = x(g^{-m})^i$$

Algorithme:

- Calcule liste L: $(j,\ g^j),\ 0 \leq j < m$
- Calcule des $(i,\ x(g^{-m})^i)$
  - Cherche si $x(g^{-m})^i$ est dans L
- Retourner $im + j$

$$0 \leq i, j < m$$

Complexité : $O(\sqrt{n})$

EL GAMAL DANS $\mathbb{Z}/p\mathbb{Z}$: CRÉATION DES CLÉS

```python
def creation_cles(q,g,a):
    """
    :param q: un entier premier (dans la pratique tres grand)
    :param g: un entier générateur de Z/qZ donc tel que p^q = 1
    :param a: un entier compris entre 0 et ordre(g) = q
    :return: couple de clé publique et clé privée pour le crypto systeme El Gamal
    """
    h = g ** a
    public_key = (q, h, g)
    private_key = a
    return private_key, public_key
```

EL GAMAL DANS $\mathbb{Z}/p\mathbb{Z}$: CHIFFREMENT / DÉCHIFFREMENT

```python
def chiffrement(public_key, msg):
    """
    :param public_key: clé publique de la personne à qui on veut envoyer un msg
    :param msg: message à chiffrer convertie en entier
    :return: le message crypté
    """
    q, h, g = public_key
    k = randint(0, q)
    p = (g ** k) % q
    s = (h ** k) % q
    return p, msg * s
```

```python
def dechiffrement(private_key, public_key, msg_encrypte):
    """
    :param private_key: clé privé de la personne qui recois le message
    :param public_key: clé publique de la personne qui recois le message
    :param msg_encrypte: message recu crypté
    :return: retourne le message decrypté
    """
    q, h, g = public_key
    p, msg = msg_encrypte
    s = p ** private_key % q
    return int(msg/s)
```

EL GAMAL DANS $\mathbb{Z}/p\mathbb{Z}$: CHIFFREMENT / DÉCHIFFREMENT MESSAGE

```python
def chiffrement_message(public_key, msg):
    res = []
    for lettre in msg :
        res.append(chiffrement(public_key, ord(lettre)))
    return res
```

```python
def dechiffrement_message(private_key, public_key, msg_encrypte):
    res = ""
    for i in msg_encrypte :
        res = res + chr(dechiffrement(private_key, public_key, i))
    return res
```