Courbes elliptiques et Cryptographie

TIPE

Antoine Goudey - n°32037

2026

Introduction

Problématique

Dans quelle mesure la structure de groupe des courbes elliptiques permet-elle d'optimiser le ratio sécurité/performance par rapport à d'autres cryptosystèmes, et quelles sont les limites concrètes de cette robustesse face à différentes attaques ?

Plan

1. Définitions : des courbes elliptiques à la loi de groupe

2. Quelques techniques de chiffrement : El Gamal et Diffie Hellman

3. Simulations d'attaques et résultats expérimentaux

Une courbe elliptique

- Équation :

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

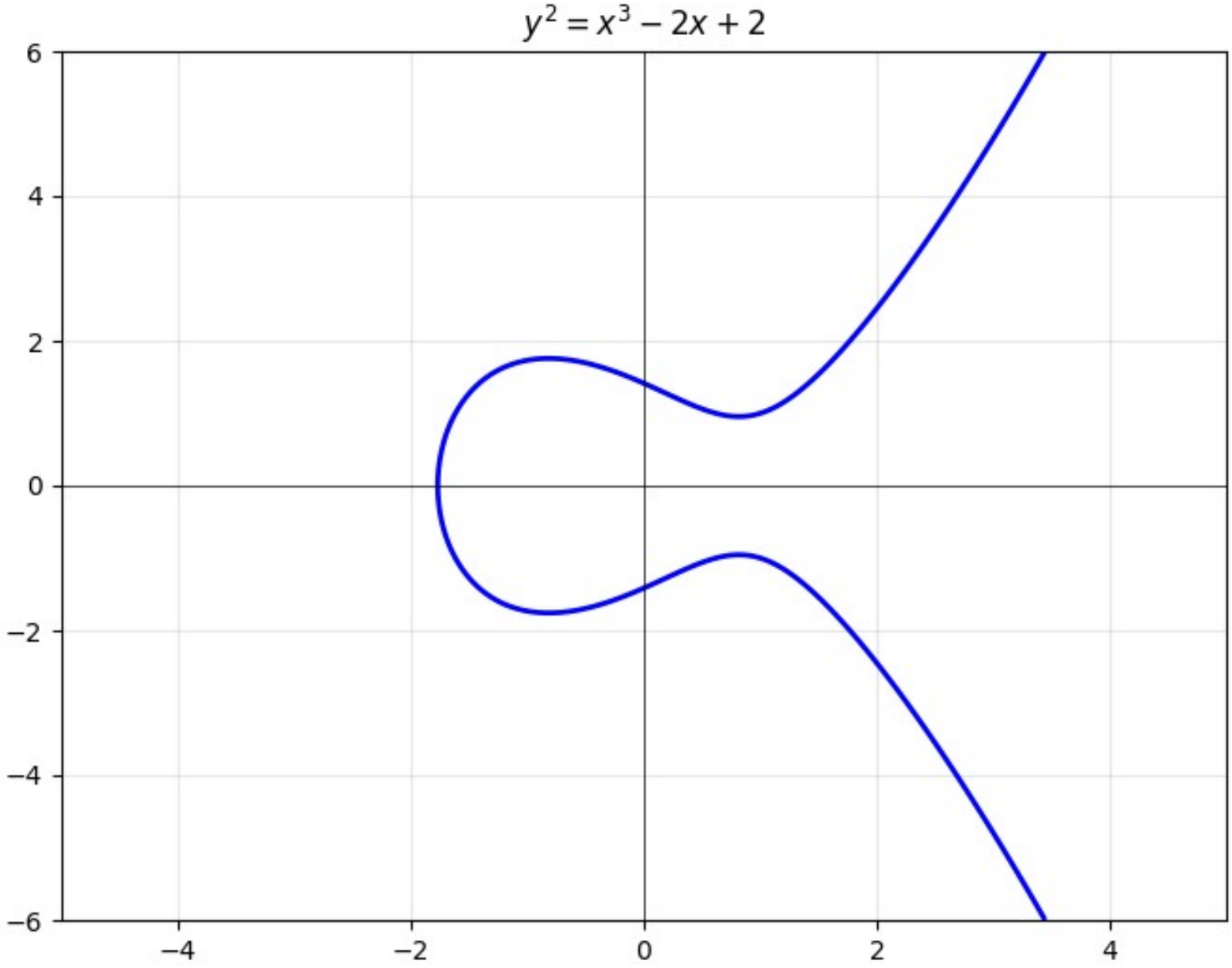
- Condition : $a, b \in K$ tels que

$$4a^3 + 27b^2 \neq 0$$

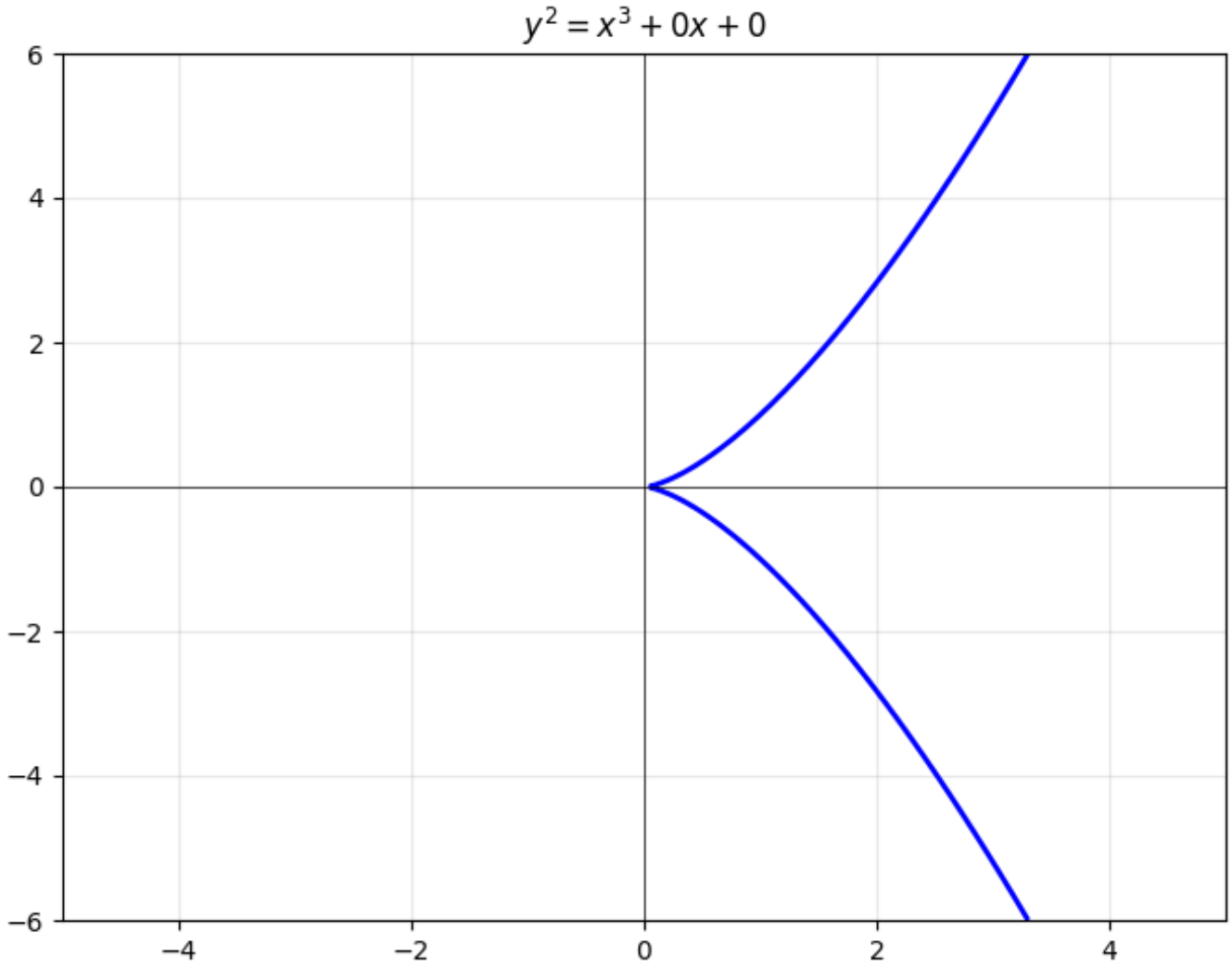
- Ensemble des points de cette courbe :

$$E(K) = \{(x, y) \in K^2 \mid y^2 = x^3 + ax + b\} \cup \{\infty\}$$

Une courbe elliptique - Exemple



Et si $4a^3 + 27b^2 = 0$?



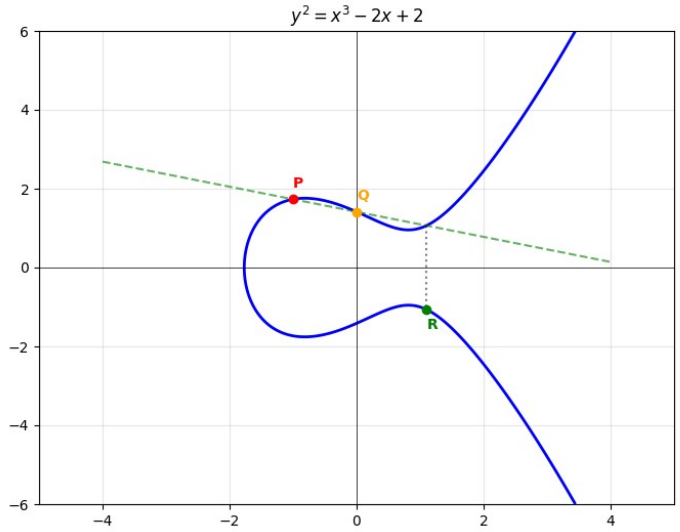
Addition de points

Définition

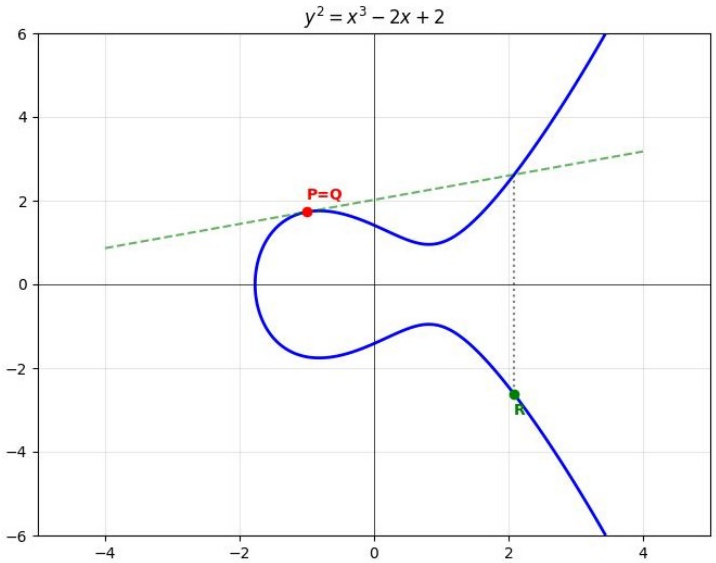
Si $P \in E \setminus \{\infty\}$, on pose $-P = (x, -y)$.

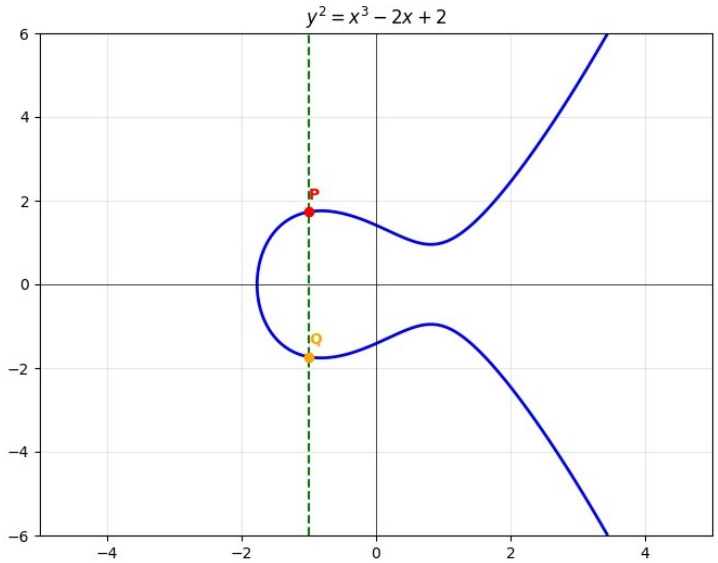
Définition

Si P, Q sont dans E, on définit $P + Q$ géométriquement de la manière suivante.



Addition de points - Cas particuliers : $Q = P$, $Q = -P$





Addition de points - Formules algébriques

- $P + \infty = \infty + P = P$
- $P + (-P) = \infty$
- Si $P = (x_p, y_p)$, $Q = (x_q, y_q)$ et $Q \neq \pm P$, on pose $P + Q = S(x_s, y_s)$, avec :

$$\begin{cases} x_s = \left(\frac{y_q - y_p}{x_q - x_p}\right)^2 - x_p - x_q \\ y_s = \left(\frac{y_q - y_p}{x_q - x_p}\right)(x_p - x_s) - y_p \end{cases}$$

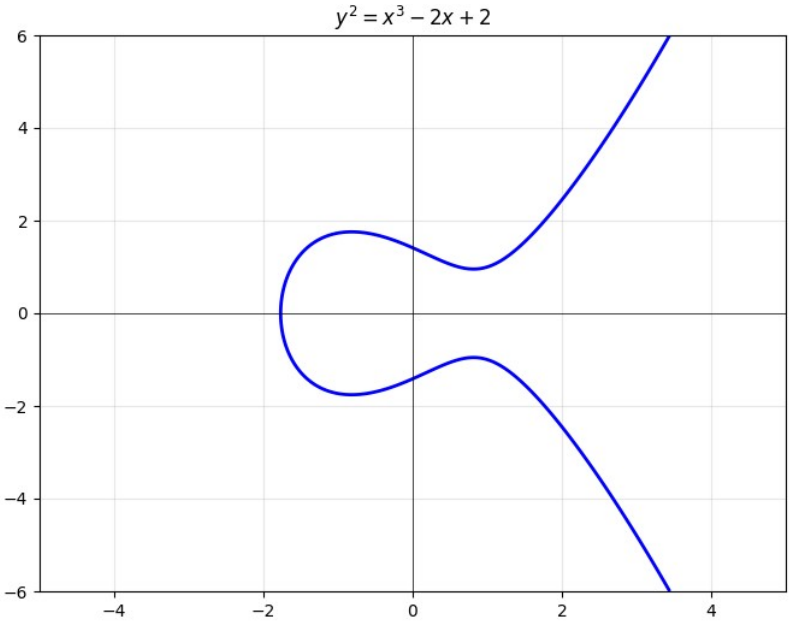
- Si $P(x_p, y_p)$, et $P \neq -P$, alors $P + P = S(x_s, y_s)$, avec :

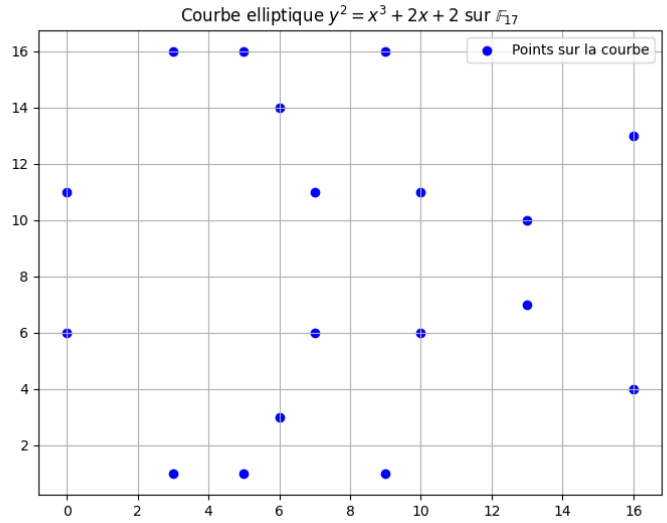
$$\begin{cases} x_s = \left(\frac{3x_p^2 + a}{2y_p}\right)^2 - 2x_p \\ y_s = \left(\frac{3x_p^2 + a}{2y_p}\right)(x_p - x_s) - y_p \end{cases}$$

Structure de groupe - Passage à $\mathbb{F}_p$

Théorème

L'ensemble des points $E(K)$ d'une courbe elliptique sur un corps K, muni de l'addition de points, forme un groupe abélien, d'élément neutre $+\infty$.





Addition de points - Exponentiation rapide, Ordre d'un point

Pour calculer nP, on peut utiliser l'algorithme d'exponentiation rapide, qui est logarithmique en n.

- Si n est pair, $nP = 2\left(\frac{n}{2}P\right)$
- Si n est impair, $nP = P + (n-1)P$

Complexité

$O(\log n)$ opérations pour calculer nP.

Définition

L'ordre d'un point P est le plus petit entier ω (ou $\omega(P)$) tel que $\omega P = \infty$.

▸ Code en Annexe

Cryptographie asymétrique et problème du logarithme discret elliptique

- Basée sur des fonctions à sens unique : faciles à calculer, difficiles à inverser.
- Étant donné $G \in E$, et un entier $n \in \mathbb{N}^*$, il est simple de calculer $nG = \underbrace{G + G + \cdots + G}_{n \text{ fois}}$.
- En revanche, étant donné G et $P = nG$, il est difficile de retrouver n.

Encodage d'un message - Méthode de Koblitz [1]

- $K \in \mathbb{N}^*$, par exemple $K = 100$.
- Message : $m \in \mathbb{N}$, tel que $0 \leq K(m+1) < p$.
- On cherche $j \in \{0, 1, \ldots, K-1\}$ et $y \in \mathbb{F}_p$ tel que $(x_j, y) \in E$, $x_j = \overline{Km + j}$.
- On teste pour $j = 0, 1, \ldots, K-1$ si $y^2 = x_j^3 + ax_j + b$ admet une solution dans $\mathbb{F}_p$.
- $\frac{p-1}{2}$ carrés dans $\mathbb{F}_p^*$, la probabilité d'échec est d'environ $\frac{1}{2^K}$.
- Décodage : $j \equiv x \pmod{K}$ si $P = (x, y)$, et $m = \frac{x-j}{K}$

Exemple

Avec $K = 100$, le temps d'attente moyen pour voir un échec avec un milliard d'encodages par seconde est 2000 fois l'âge de l'univers.

▸ Code en Annexe

1. Koblitz, 1987.

Chiffrement d'El Gamal [2] [3]

G est un point de E d'ordre ω grand.

Alice

1. Entier secret : $0 < n_A < \omega$
2. Clé publique : $P_A = n_A G$
3. Donne (P_A, G, ω) à Bob
8. Déchiffrement par Alice : $S_2 - n_A S_1 = M$

Bob

4. Message : $M \in E$
5. Entier aléatoire : $0 \leq k < \omega$
6. Calcule $S_1 = kG$, $S_2 = M + kP_A$
7. Envoie (S_1, S_2) à Alice

Conclusion

Peu utilisé en pratique, à cause de Koblitz ($K(m+1) < p$).

▸ Code en Annexe

2. ElGamal, 1985.
3. Hoffstein et al., 2010.

Échange de clés Diffie-Hellman [4]

Alice		Bob
1. Entier secret : n_A		1. Entier secret : n_B
2. Calcule $P_A = n_A G$		2. Calcule $P_B = n_B G$
3. Donne P_A à Bob	$\longleftrightarrow$	3. Donne P_B à Alice
4. Calcule le secret $S = n_A P_B$		4. Calcule le secret $S = n_B P_A$

Conclusion

Très utilisé en pratique, une fois le secret partagé la communication est sécurisée par un chiffrement symétrique (exemple : AES).

▸ Code en Annexe ▸ AES

4. Diffie et Hellman, 1976.

Attaques et expériences

Questions

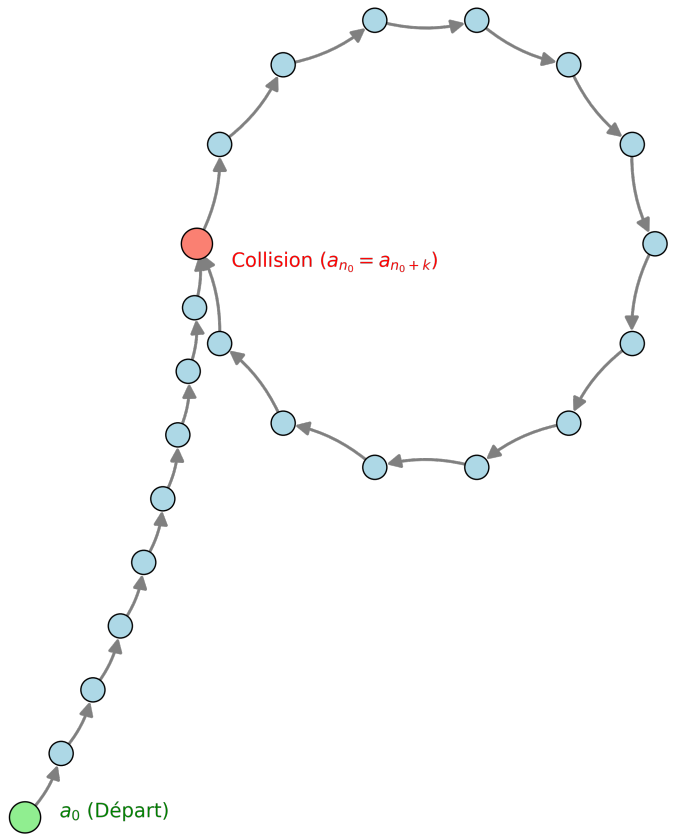
1. Quelles attaques peut-on utiliser pour résoudre le problème du logarithme discret ?
2. Quelle est la complexité de ces attaques, est-ce qu'il y a une plus grande sécurité par rapport à d'autres cryptosystèmes ?
3. En simulant ces attaques moi-même, puis-je obtenir les résultats attendus théoriquement ?

Attaque de Pollard's Rho fondée sur l'algorithme de Floyd

- Méthode naïve : on stocke $a_0, a_1, a_2, \ldots, a_{m-1}$, on calcule a_m puis on compare.
- Algorithme de Floyd : Même temps, mais mémoire constante.

Théorème

Soit $f : I \to I$ une fonction, avec I fini. Soit $a_0 \in I$, $a_{n+1} = f(a_n)$. Alors la suite (a_n) est périodique à partir d'un certain rang n_0, de période k. De plus, il existe m tel que $a_m = a_{2m}$, et $n_0 \leq m < n_0 + k$



▸ Code en Annexe

Attaque de Pollard's Rho - Attaque [5]

- On part d'un point $a_0 = c_0 P + d_0 Q$
- On dispose d'une fonction f qui, à partir de a_n, calcule a_{n+1}, et les coefficients c_{n+1} et d_{n+1} tels que $a_{n+1} = c_{n+1} P + d_{n+1} Q$.
- On calcule a_m et a_{2m} jusqu'à trouver $a_m = a_{2m}$. Cela donne $c_m P + d_m Q = c_{2m} P + d_{2m} Q$, donc $(c_m - c_{2m}) P = (d_{2m} - d_m) Q$
- Enfin : $n \equiv (c_m - c_{2m}) \times (d_{2m} - d_m)^{-1} \pmod{\omega(P)}$

Question

Quel est le temps d'attente moyen avant de trouver une collision ?

5. Hankerson et al., 2004.

Attaque de Pollard's Rho - Paradoxe des anniversaires I

Définition

Soit ω boules numérotées dans une urne. On tire les boules indépendamment et uniformément, avec remise. Soit T le nombre de tirages nécessaires pour obtenir une boule déjà tirée. Quelle est la valeur de $E(T)$?

- Probabilité de faire au moins i tirages :

$$\mathbb{P}(T > i) = \frac{\omega!}{(\omega - i)!\omega^i} = \prod_{j=0}^{i-1} \left(1 - \frac{j}{\omega}\right)$$

Attaque de Pollard's Rho - Paradoxe des anniversaires II

- T est à valeurs dans $\mathbb{N}$:

$$E(T) = \sum_{i=0}^{+\infty} \mathbb{P}(T > i) = \sum_{i=0}^{\omega} \frac{\omega!}{(\omega - i)!\omega^i}$$

- Approximation de la probabilité :

$$\mathbb{P}(T > i) = \exp\left(\sum_{j=0}^{i-1} \ln\left(1 - \frac{j}{\omega}\right)\right) \approx \exp\left(-\frac{i(i-1)}{2\omega}\right)$$

- Approximation de l'espérance :

$$\mathbb{E}(T) \underset{\omega \to +\infty}{\sim} \sqrt{\frac{\pi\omega}{2}}$$

▸ Démonstration

Attaque de Pollard's Rho - Complexité

Complexité

L'attaque de Pollard's Rho trouve n en $O(\sqrt{\omega(P)})$ opérations, avec une mémoire constante. C'est le meilleur algorithme connu dans le cas général.

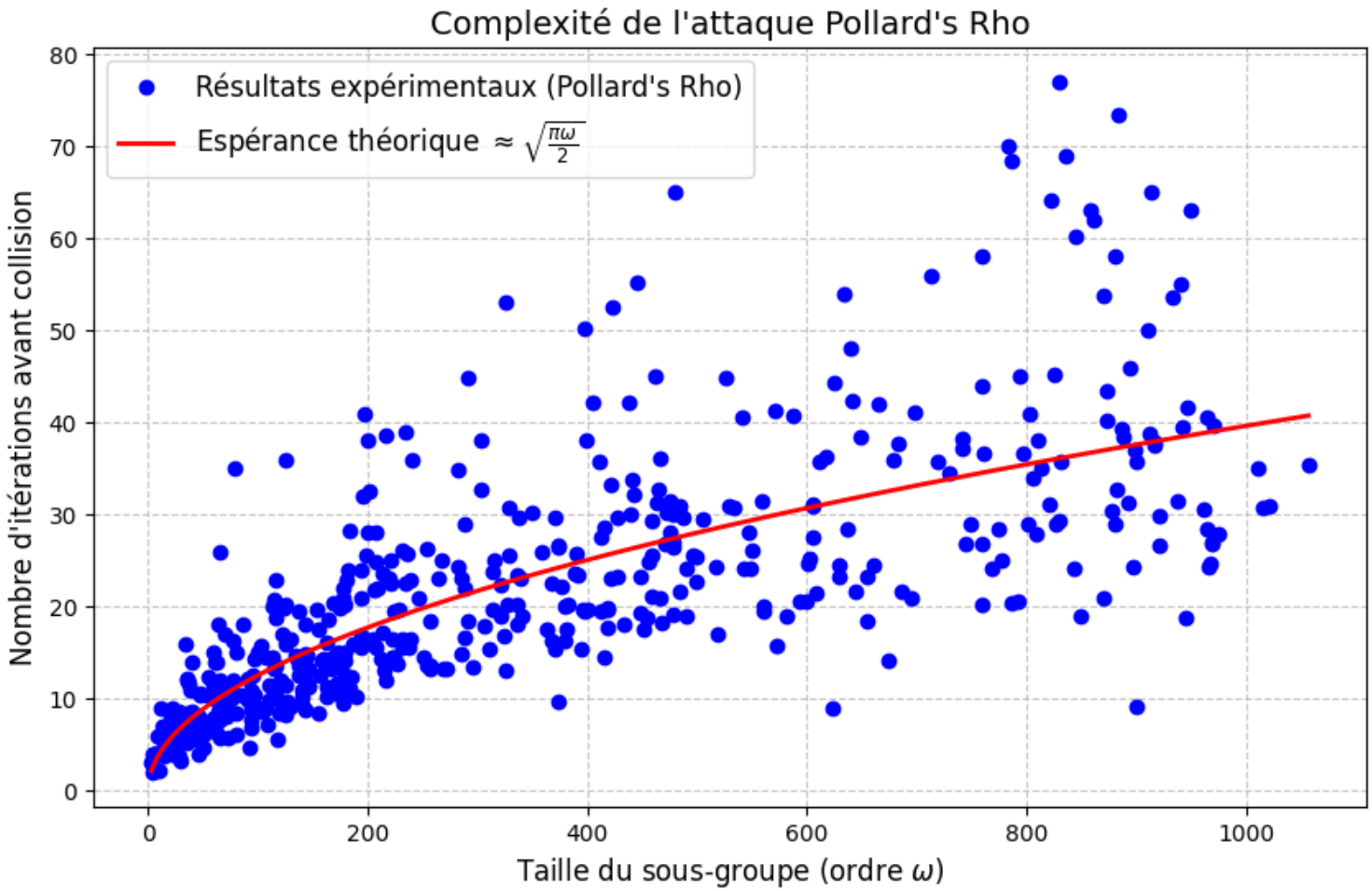
Remarque

Pour une courbe sécurisée sur $\mathbb{F}_p$, $\omega(P) \approx p$

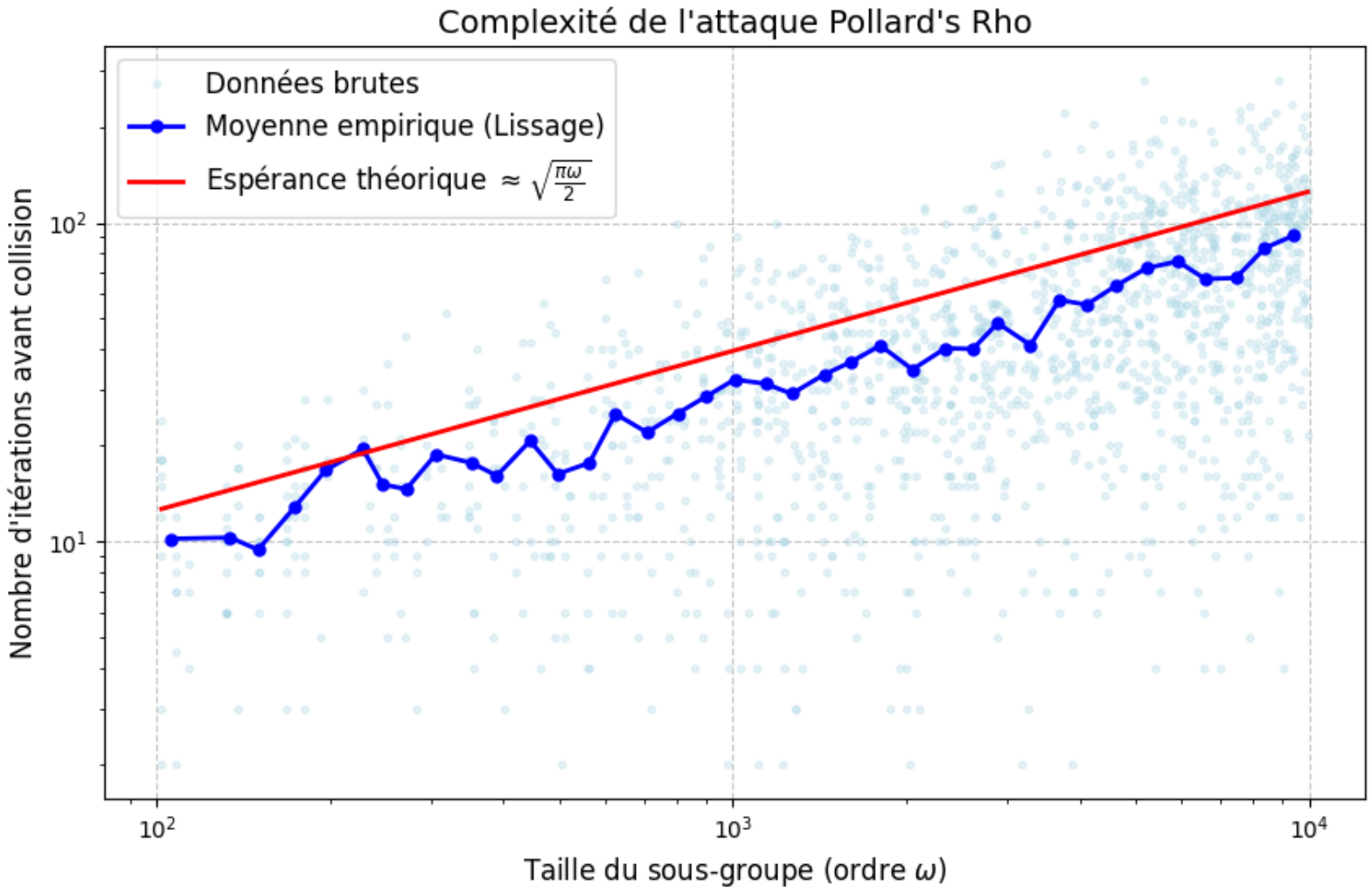
Remarque

Complexité **exponentielle** en le nombre de bits de p.

Attaque de Pollard's Rho - Résultats expérimentaux



Attaque de Pollard's Rho - Résultats expérimentaux lissés



▸ Précision

Meilleur algorithme classique dans le cas général

p est le nombre premier en paramètre, on suppose $p \approx 2^k$

Algorithme Pollard's rho

- Pour RSA [a] et ECC [b]
- Complexité temporelle : $O(\sqrt{p}) = O(2^{k/2})$
- Complexité spatiale : $O(1)$
- Complexité exponentielle

a. Rivest–Shamir–Adleman
b. Elliptic-Curve Cryptography

Algorithme GNFS [a]

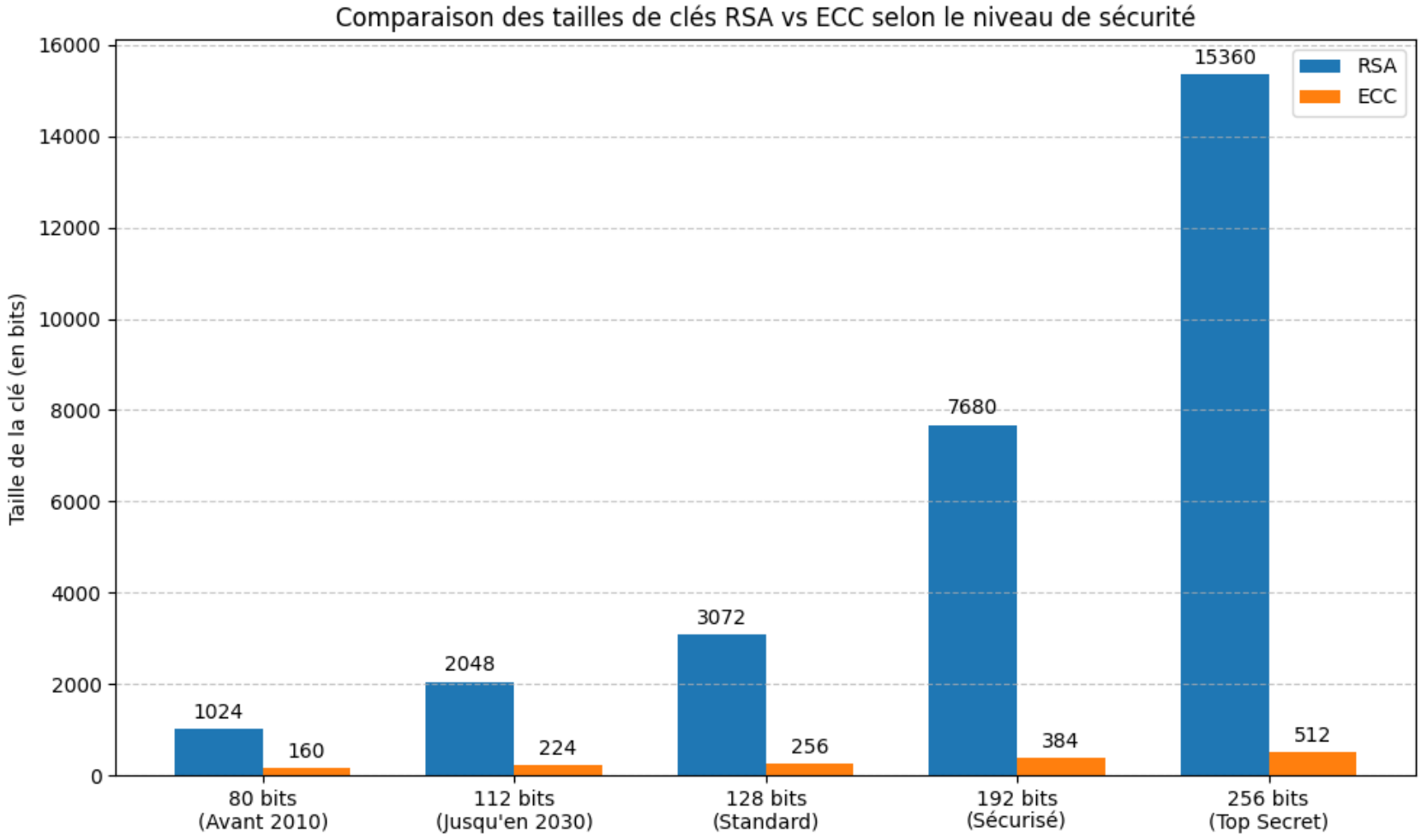
- Pour RSA **uniquement**
- Complexité spatiale et temporelle : $O\left(\exp\left(ck^{1/3}(\log k)^{2/3}\right)\right)$
- Complexité sous-exponentielle

a. General Number Field Sieve

Conclusion

GNFS est bien meilleur que Pollard's rho, mais il ne s'applique pas pour ECC

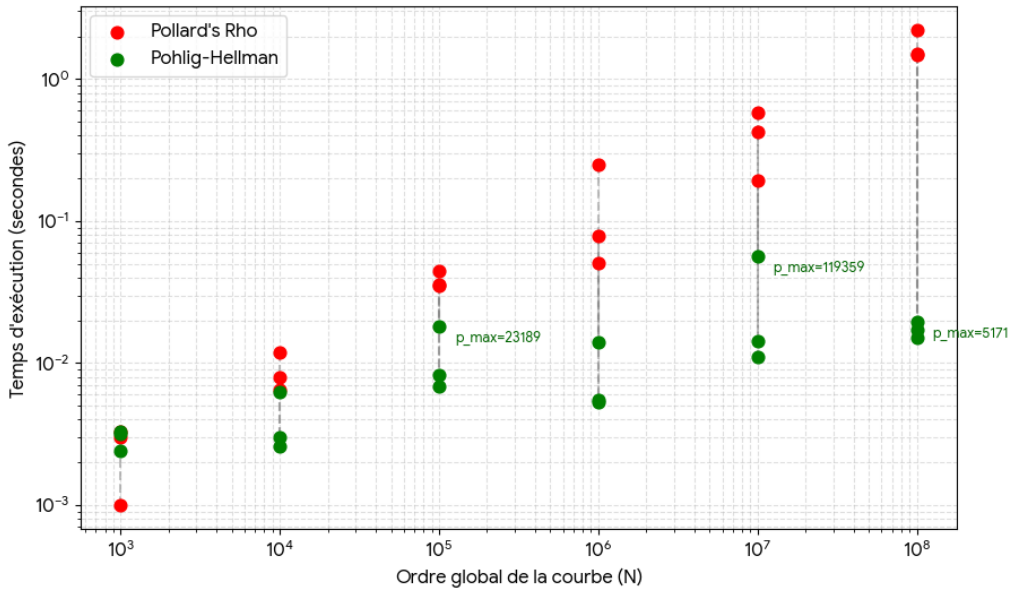
Tailles des clés pour niveau de sécurité équivalent [6] [7]



6. Barker, 2020.
7. ANSSI, 2026.

Attaques optimisées [8] [9]

- Ordre de la courbe friable : Attaque de Pohlig-Hellman.
  ▸ Explication ▸ Variante
- Protocoles mal implémentés : Invalid Curve Attack
  ▸ Explication
- Dans le futur : Attaques quantiques



8. Aumasson, 2017.
9. Bernstein et Lange, 2017.

Bibliographie I

ANSSI. (2026, mars). *Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques* (rapp. tech.). Agence nationale de la sécurité des systèmes d'information.

Aumasson, J.-P. (2017). *Serious Cryptography*. No Starch Press.

Barker, E. (2020, avril). *Recommendation for Key Management : Part 1 – General* (rapp. tech. N° NIST Special Publication (SP) 800-57 Part 1 Revision 5). National Institute of Standards et Technology.

Bernstein, D. J., & Lange, T. (2017). *SafeCurves : choosing safe curves for elliptic-curve cryptography* [Consulté le : 21/10/2025]. URL.

Bibliographie II

Diffie, W., & Hellman, M. (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory, 22*(6).

ElGamal, T. (1985). A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory, 31*(4).

Hankerson, D., Menezes, A., & Vanstone, S. (2004). *Guide to Elliptic Curve Cryptography*. Springer.

Hoffstein, J., Pipher, J., & Silverman, J. (2010). *An Introduction to Mathematical Cryptography*. Springer.

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of computation, 48*(177), 203-209.

Annexe

Informations complémentaires et codes Python

Python - Objets I

```
import matplotlib.pyplot as plt

class Entierp:
    def __init__(self, n, p):
        self.n = n % p
        self.p = p

    def __add__(self, autre):
        if isinstance(autre, int):
            return Entierp(self.n + autre, self.p)
        assert self.p == autre.p
        return Entierp(self.n + autre.n, self.p)

    def __radd__(self, autre):
        return self.__add__(autre)

    def __sub__(self, autre):
        if isinstance(autre, int):
            return Entierp(self.n - autre, self.p)
        assert self.p == autre.p
        return Entierp(self.n - autre.n, self.p)
```

Python - Objets II

```
    def __rsub__(self, autre):
        return Entierp(autre - self.n, self.p)

    def __neg__(self):
        return Entierp(-self.n, self.p)

    def __mul__(self, autre):
        if isinstance(autre, int):
            return Entierp(self.n * autre, self.p)
        assert self.p == autre.p
        return Entierp(self.n * autre.n, self.p)

    def __rmul__(self, autre):
        if isinstance(autre, int):
            return Entierp(self.n * autre, self.p)
        assert self.p == autre.p
        return Entierp(self.n * autre.n, self.p)

    def inv(self):
        return Entierp(pow(self.n, -1, self.p), self.p)

    def __truediv__(self, autre):
```

Python - Objets III

```
        if isinstance(autre, int):
            autre = Entierp(autre, self.p)
        return self.__mul__(autre.inv())

    def __pow__(self, k):
        return Entierp(pow(self.n, k, self.p), self.p)

    def __eq__(self, autre):
        if isinstance(autre, Entierp):
            return (self.n - autre.n) % self.p == 0
        elif isinstance(autre, int):
            return (self.n - autre) % self.p == 0
        return False

    def __ne__(self, autre):
        return not self.__eq__(autre)

    def __repr__(self):
        return f"{self.n}"

class CourbeElliptiqueFp:
    def __init__(self, a, b):
```

Python - Objets IV

```
        self.a = a
        self.b = b

    def appartient(self, point):
        # En coordonnées projectives : Y^2 * Z = X^3 + aXZ^2 + bZ^3
        gauche = (point.y ** 2) * point.z
        droite = (point.x ** 3) + self.a * point.x * (point.z ** 2) +
        ↪  self.b * (point.z ** 3)
        return gauche == droite

    def evaluer(self, x):
        return x ** 3 + self.a * x + self.b

    def afficher(self, p):
        """
        Affiche les points de la courbe sur le corps fini F_p, avec un
        ↪  algorithme naïf.
        """
        points_x = []
        points_y = []

        for x_val in range(p):
```

Python - Objets V

```
            x_entier = Entierp(x_val, p)
            droite = self.evaluer(x_entier)

            for y_val in range(p):
                y_entier = Entierp(y_val, p)
                if y_entier ** 2 == droite:
                    points_x.append(x_val)
                    points_y.append(y_val)

        plt.figure(figsize=(6, 6))
        plt.scatter(points_x, points_y, c='blue', marker='o')
        plt.title(f"Points de la courbe sur $F_{{{p}}}$")
        plt.grid(True)
        plt.xlim(-1, p)
        plt.ylim(-1, p)
        plt.show()

    def __repr__(self):
        return f"y^2=x^3+{self.a}x+{self.b}"

class Point:
    def __init__(self, x, y, z, courbe):
```

Python - Objets VI

```
        self.x = x
        self.y = y
        self.z = z
        self.c = courbe

    def sur_la_courbe(self):
        return self.c.appartient(self)

    def est_infini(self):
        return self.z == 0

    def reduire(self):
        if self.z == 0:
            # Point à l'infini : (0:1:0)
            return Point(Entierp(0, self.x.p), Entierp(1, self.x.p),
            ↪  Entierp(0, self.x.p), self.c)
        else:
            return Point(self.x / self.z, self.y / self.z, Entierp(1,
            ↪  self.x.p), self.c)

    def __add__(self, autre):
        P = self
```

Python - Objets VII

```
Q = autre

if P.est_infini():
    return Q
elif Q.est_infini():
    return P

P = P.reduire()
Q = Q.reduire()

if P.x == Q.x and P.y == -Q.y:
    return Point(Entierp(0, P.x.p), Entierp(1, P.x.p), Entierp(0,
    ↪  P.x.p), self.c)

if P.x == Q.x and P.y == Q.y: # Cas P == Q
    if P.y == 0: # Tangente verticale
        return Point(Entierp(0, P.x.p), Entierp(1, P.x.p),
        ↪  Entierp(0, P.x.p), self.c)
    l = (3 * (P.x ** 2) + self.c.a) / (2 * P.y)

else: # Cas P != Q (Addition classique)
    l = (Q.y - P.y) / (Q.x - P.x)
```

Python - Objets VIII

```
        xs = (l ** 2) - P.x - Q.x
        ys = l * (P.x - xs) - P.y

        return Point(xs, ys, Entierp(1, P.x.p), self.c)

    def __rmul__(self, k):
        R = Point(Entierp(0, self.x.p), Entierp(1, self.x.p), Entierp(0,
        ↪  self.x.p), self.c) # Point à l'infini
        P = self
        while k > 0:
            if k % 2 == 1:
                R = R + P
            P = P + P
            k //= 2
        return R

    def __sub__(self, autre):
        return self + Point(autre.x, -autre.y, autre.z, autre.c)

    def __eq__(self, autre):
        # En projectif
```

Python - Objets IX

```
        return (self.x * autre.z == autre.x * self.z) and \
               (self.y * autre.z == autre.y * self.z)

    def __repr__(self):
        if self.z == 0:
            return "(0)"
        P_red = self.reduire()
        return f"({P_red.x} : {P_red.y})"
```

◂ Retour à la présentation

Python - Koblitz I

```
def koblitz_encoder(message: int, courbe: CourbeElliptiqueFp, p: int, K:
↪  int = 20):
    """Encode un entier 'message' en un Point sur la courbe."""
    for j in range(K):
        x_val = message * K + j

        if x_val >= p:
            raise ValueError("Message trop grand pour les paramètres de la
            ↪  courbe")

        x_entier = Entierp(x_val, p)

        y_carre = courbe.evaluer(x_entier)

        if est_residu_quadratique(y_carre.n, p):
            y_val = racine_carree_mod(y_carre.n, p)
            return Point(x_entier, Entierp(y_val, p), Entierp(1, p),
            ↪  courbe)

    raise Exception("Impossible d'encoder ce message (pas de point
    ↪  trouvé)")
```

Python - Koblitz II

```python
def koblitz_decoder(point: Point, K: int = 20):
    """Récupère l'entier m à partir du point P(x, y)"""
    return point.x.n // K
```

◂ Retour à la présentation

Python - ElGamal

```python
def elgamal_chiffrer(message_point: Point, cle_publique_dest: Point,
↪  generateur: Point, ordre: int):
    """Chiffre un Point message M. Retourne (C1, C2)"""
    k = random.randint(0, ordre - 1)
    C1 = k * generateur
    C2 = message_point + (k * cle_publique_dest)

    return C1, C2

def elgamal_dechiffrer(C1: Point, C2: Point, cle_privee: int):
    """Déchiffre la paire (C1, C2). M = C2 - d * C1"""
    S = cle_privee * C1
    M = C2 - S
```

◂ Retour à la présentation

Python - Diffie-Hellman

```python
def ecdh_generer_cles(generateur: Point, ordre_courbe: int):
    """Génère (clé_privée, clé_publique)"""
    privee = random.randint(1, ordre_courbe - 1)
    publique = privee * generateur
    return privee, publique

def ecdh_calculer_secret(cle_privee: int, cle_publique_autre: Point):
    """Calcule le secret partagé S = d_moi * Q_autre"""
    return cle_privee * cle_publique_autre
```

◂ Retour à la présentation

Python - Pollard's Rho I

```python
# ordre est l'ordre de P
def pollard_rho(P: Point, Q: Point, ordre: int, c:int=1, d:int=0) ->
↪  tuple[int, int]:
    """
    Renvoie n tel que Q=nP en utilisant l'algorithme Pollard's Rho.
    """
    compteur = 1
    c_lent = c
    d_lent = d
    c_rapide = c
    d_rapide = d

    def f(R: Point, c_val: int, d_val: int):
        if R.x.n % 3 == 0:
            return R + P, (c_val + 1) % ordre, d_val
        elif R.x.n % 3 == 1:
            return 2 * R, (2 * c_val) % ordre, (2 * d_val) % ordre
        else:
            return R + Q, c_val, (d_val + 1) % ordre

    depart = c * P + d * Q
    lent, c_lent, d_lent = f(depart, c_lent, d_lent)
```

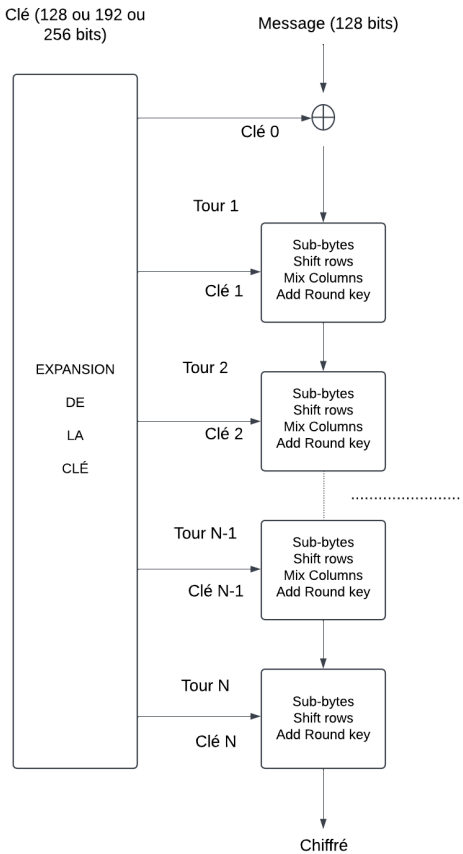
Python - Pollard's Rho II

```
rapide, c_rapide, d_rapide = f(depart, c_rapide, d_rapide)
rapide, c_rapide, d_rapide = f(rapide, c_rapide, d_rapide)

while rapide != lent:
    lent, c_lent, d_lent = f(lent, c_lent, d_lent)
    rapide, c_rapide, d_rapide = f(rapide, c_rapide, d_rapide)
    rapide, c_rapide, d_rapide = f(rapide, c_rapide, d_rapide)
    compteur += 1

delta_d = (d_rapide - d_lent) % ordre
delta_c = (c_lent - c_rapide) % ordre
pgcd = gcd(delta_d, ordre)
if pgcd == 1:
    inv_delta_d = pow(delta_d, -1, ordre)
    n_calc = (delta_c * inv_delta_d) % ordre
    return n_calc, compteur
else:
```

◂ Retour à la présentation

Advanced Encryption Standard (AES)

- Chiffrement symétrique (même clé pour chiffrer et déchiffrer) par blocs (de 128 bits, représentés par une matrice 4×4)
- Clés de 128, 192 ou 256 bits, avec plusieurs tours de chiffrement (resp. 10, 12 et 14), avec 4 opérations à chaque tour



Définition

- Sub-Bytes : Substitution de chaque octet
- Shift rows : Décalage de la i-ème ligne de $i + 1$
- Mix Columns : Multiplication matricielle
- Add Round Key : XOR avec la clé étendue

◂ Retour à la présentation

Démonstration du paradoxe des anniversaires

- T à valeurs dans $\mathbb{N}$: $\mathbb{E}[T] = \int_0^{+\infty} \mathbb{P}(T > t)dt$
- $\mathbb{E}(T) = \sqrt{\omega} \int_0^{+\infty} g_\omega(u)du$ avec $g_\omega(u) = \mathbb{P}(T > \lfloor u\sqrt{\omega} \rfloor)$
- Avec $n_\omega = \lfloor u\sqrt{\omega} \rfloor$, $\ln(g_\omega(u)) = \sum_{i=1}^{n_\omega - 1} \ln\left(1 - \frac{i}{\omega}\right)$ si $n_\omega < \omega$
- Par Taylor-Lagrange : $\forall u > 0, g_\omega(u) \xrightarrow{\omega \to +\infty} e^{-u^2/2}$
- Par concavité de $\ln$, $g_\omega(u) \leq \exp\left(-\frac{n_\omega(n_\omega - 1)}{2\omega}\right)$ si $n_\omega < \omega$ et on domine par :

$$\varphi(u) = \begin{cases} 1 & \text{si } 0 < u \leq 2 \\ e^{-u^2/16} & \text{si } u > 2 \end{cases}$$

Conclusion

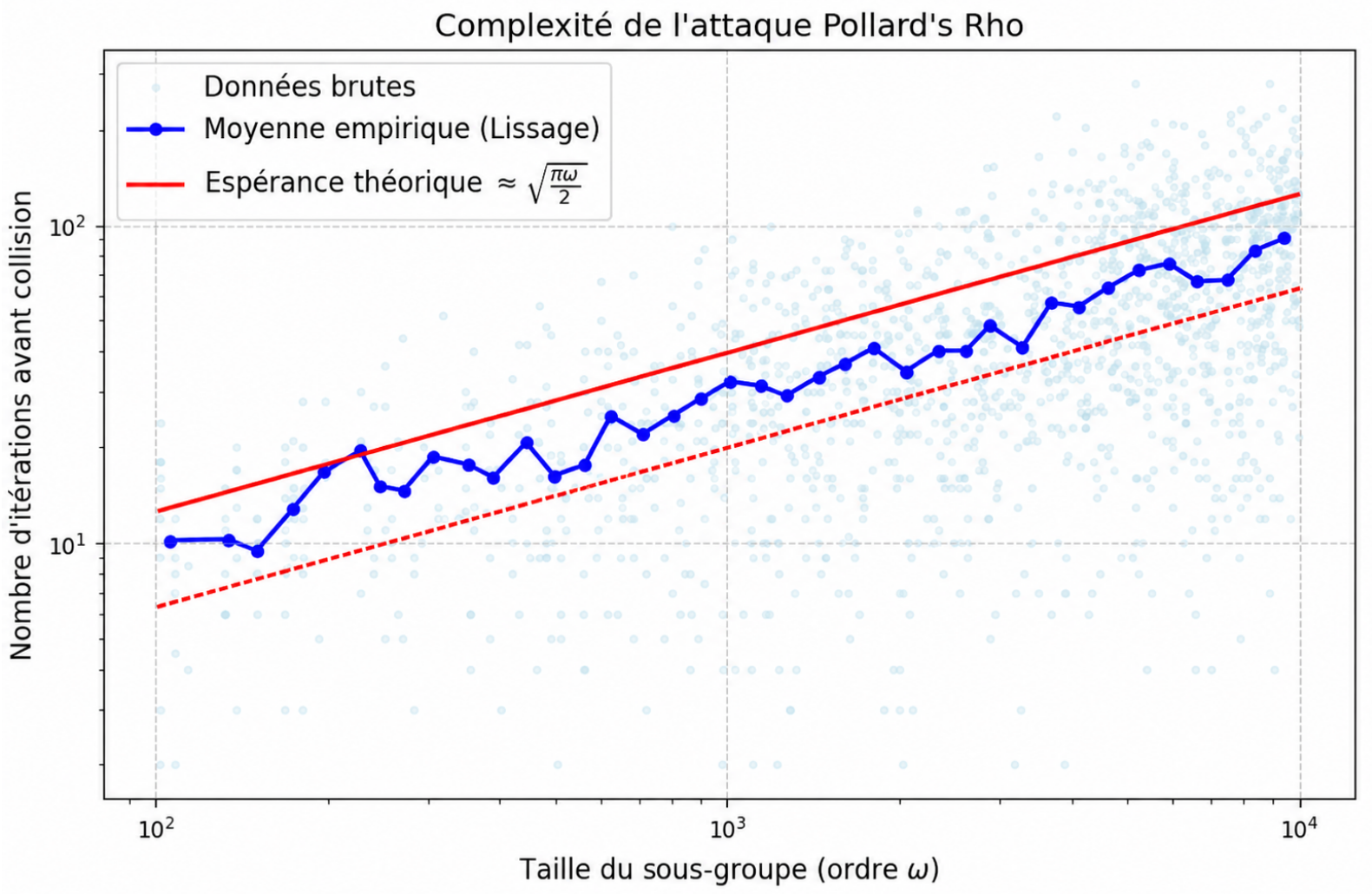
Par théorème de convergence dominée, avec l'intégrale de Gauss :
$\mathbb{E}[T] \underset{\omega \to +\infty}{\sim} \sqrt{\frac{\pi\omega}{2}}$

◂ Retour à la présentation

Python - Théorème des restes chinois

```python
def crt(congruences: list[int], modulo: list[int]) -> int:
    """
    Entrée : Liste de congruences, de modulo
    Sortie : x entre 0 et le produit des modulo tel que x \equiv
    ↪  congruences[i] mod modulo[i]
    """

    N = reduce(lambda x, y: x*y, modulo, 1)
    liste_x = [0 for _ in range(len(modulo))]
    for i,m in enumerate(modulo):
        prod_partiel = N//m
        if gcd(prod_partiel, m) == 1:
            inv_prod_partiel = pow(prod_partiel, -1, m)
        else:
            raise ValueError("Les modulos ne sont pas premiers entre eux 2
            ↪  à 2.")
        liste_x[i] = inv_prod_partiel * prod_partiel % N
    x = 0
    for i, xi in enumerate(liste_x):
        x += xi * congruences[i]
        x %= N
    return x
```

Attaque de Pollard's Rho - Précision sur les résultats



◂ Retour à la présentation

Attaque de Pohlig-Hellman [10]

- Soit $P \in E$ d'ordre ω, $Q = nP$. Supposons que $\omega = \prod_{i=1}^{k} p_i^{\alpha_i}$, avec p_i premiers.
- Soit $\omega_i = p_i^{\alpha_i}$, $P_i = \frac{\omega}{\omega_i} P$, $Q_i = \frac{\omega}{\omega_i} Q$ (P_i d'ordre ω_i).
- On résout le logarithme discret sur Q_i et P_i (avec Pollard's Rho), on trouve n_i, tel que $Q_i = n_i P_i$.
- Si $n = k\omega_i + r$, $nP_i = (k\omega_i + r)P_i = rP_i$ et $nP_i = n\frac{\omega}{\omega_i} P = \frac{\omega}{\omega_i} Q = Q_i = n_i P_i$.
- Donc : $r_i \equiv n_i \pmod{\omega_i}$
- On trouve n avec le théorème des restes chinois.

Complexité

$O\left(\sum_{i=1}^{k} \sqrt{\omega_i}\right)$, soit $O\left(\sqrt{\omega_{i,\max}}\right)$

◂ Retour à la présentation

10. Hoffstein et al., 2010.

Python - Pohlig-Hellman

```python
def pohlig_hellman(P: Point, Q: Point, ordre: int) -> int:
    """
    Renvoie n tel que Q=nP en utilisant l'algorithme de Pohlig-Hellman.
    """
    congruences = []
    modulos = []
    factorisation = factorint(ordre) # Utilisation d'une bibliothèque
    ↪  externe
    for premier, exposant in factorisation.items():
        oi = pow(premier, exposant)
        Pi = (ordre//oi) * P
        Qi = (ordre//oi) * Q
        xi, _ = pollard_rho_safe(Pi, Qi, oi)
        congruences.append(xi)
        modulos.append(oi)
    return crt(congruences, modulos)
```

Small Subgroup Attack [11]

- Si N est divisible par un petit nombre premier l
- Un attaquant peut envoyer un point P d'ordre l à la victime, qui va calculer nP.
- Par un algorithme de force brute, l'attaquant trouve n modulo l.

Conclusion

Au lieu de calculer nP, on calcule nhP

◂ Retour à la présentation

11. Bernstein et Lange, 2017.

Attaque sur les protocoles - Invalid Curve Attack [12]

- On cherche la clé privée n d'une victime.
- Les formules d'addition ne dépendent pas de b. On choisit une courbe $E' : y^2 = x^3 + ax + b'$, avec un ordre divisible par un petit nombre premier l.
- On choisit un point P d'ordre l.
- Si la victime ne vérifie pas que P est sur la bonne courbe, elle va calculer nP, qui est le même sur E et E'.
- L'attaquant récupère alors la valeur de n modulo l. En répétant l'attaque pour plusieurs l, on obtient n par TRC.

Conclusion

Il faut vérifier que les points envoyés sont sur la courbe.

12. Bernstein et Lange, 2017.

◂ Retour à la présentation

Génération d'exemples - Courbes supersingulières I

- Définition : $N - (p+1) \equiv 0 \pmod{p}$
- Propriété arithmétique liée aux points de p-torsion
- Attaque MOV (Menezes-Okamoto-Vanstone) : réduction à $\mathbb{F}_{p^k}^*$, k petit.
- Exemple : $y^2 = x^3 + 1$ sur $\mathbb{F}_p$, $p \equiv 2 \pmod{3}$.

Théorème

La courbe $y^2 = x^3 + 1$ sur $\mathbb{F}_p$, avec $p \equiv 2 \pmod{3}$ est d'ordre $p+1$.

Génération d'exemples - Courbes supersingulières II

Démonstration

- $p - 1 \equiv 1 \pmod 3$, donc $\mathrm{pgcd}(p-1, 3) = 1$. Soit $e = 3^{-1} \pmod{p-1}$
- $f : x \in \mathbb{F}_p \mapsto x^3 \in \mathbb{F}_p$ est une bijection.
- Injectivité : Si $x^3 \equiv y^3 \pmod p$, alors $x^{3e} \equiv y^{3e} \pmod p$.
- Surjectivité : Pour $y \in \mathbb{F}_p$, $f(y^e) = y^{3e} \equiv y \pmod p$.
- En particulier, pour tout $y \in \mathbb{F}_p$, il existe un unique $x \in \mathbb{F}_p$ tel que $y^2 = x^3 + 1$.
- Avec le point à l'infini : $N = p + 1$

Python - Génération d'exemples I

```
def generation_courbe_supersinguliere(pmin: int, pmax: int):
    p = randint(pmin, pmax)
    # On force p = 2 mod 3 et premier
    while p % 3 != 2 or not est_premier(p):
        p = randint(pmin, pmax)

    courbe = CourbeElliptiqueFp(0, 1) # y^2 = x^3 + 1
    ordre_courbe = p + 1
    return courbe, p, ordre_courbe

def trouver_generateur(courbe: CourbeElliptiqueFp, p: int, ordre_courbe:
↪   int) -> Point:
    # On factorise l'ordre de la courbe avec sympy
    facteurs_premiers = list(factorint(ordre_courbe).keys())
    infini = Point(Entierp(0, p), Entierp(1, p), Entierp(0, p), courbe)

    while True:
        P = generation_point(courbe, p)
        est_generateur = True
        for q in facteurs_premiers:
            # Si (N/q) * P == infini, alors l'ordre de P divise N/q, il
            ↪  n'est pas N.
```

Python - Génération d'exemples II

```
        if (ordre_courbe // q) * P == infini:
            est_generateur = False
            break

    if est_generateur:
        return P
```