

Dimension algorithmique et chiffrement post-quantique

Gabriel Chênevert



12 octobre 2018

Dimension de Hausdorff



Notion formulée par Felix Hausdorff en 1918

Dimension de Hausdorff

Intuition

Mesure d'une boule de rayon r , dimension d :

| segment | disque | boule | hyperboule |
|---|---|---|-----------------------|
|  |  |  | ? |
| $2r$ | πr^2 | $\frac{4\pi r^3}{3}$ | $\frac{\pi^2 r^4}{2}$ |

Comportement sous dilatation :

$$m(\alpha A) = \alpha^d m(A)$$

Dimension de Hausdorff

Notion de contenu

$$m_d(A) := \inf \left\{ \sum_i r_i^d \mid A \text{ couvert par des boules de rayon } r_i \right\}$$

Définition

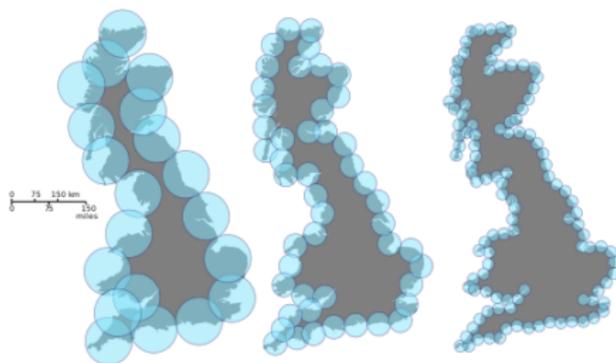
$$\dim A := \sup\{d \mid m_d(A) = +\infty\} = \inf\{d \mid m_d(A) = 0\}$$

Dimension de Hausdorff

Dans les cas simples

Si $N(r)$ désigne le nombre de boules de rayon r nécessaires pour couvrir A ,

$$\dim A = \lim_{r \rightarrow 0} \frac{\log N(r)}{\log 1/r}.$$



Dimension algorithmique

Étant donné un algorithme A , on considère pour $n \in \mathbf{N}$ le nombre maximal A_n d'étapes de calcul effectuées par A sur une entrée de taille n .

Définition

$$\dim A := \lim_{n \rightarrow \infty} \frac{\log A_n}{\log n}$$

En particulier : si $A_n \sim C n^d$, alors $\dim A = d$.

Mais aussi par exemple $A_n = C n^d \log n (\log \log n)^4$

Dimension algorithmique

Exemples

- Énumérer les points à coordonnées entières dans une d -boule : dimension d
- Test de primalité naïf : dimension $\frac{1}{2}$
- Multiplication matricielle : dimension comprise entre 2 et 3
 - ▶ algorithme naïf : 3
 - ▶ Strassen : 2,807
 - ▶ Coppersmith-Winograd : 2,376

Dimension algorithmique

Algorithme d'Euclide

Le cas le plus défavorable est le calcul du PGCD de deux termes successifs dans la suite de Fibonacci : on obtient

$$A_n = C \log_{\phi}(n) \quad \text{où} \quad \phi = \frac{1 + \sqrt{5}}{2}$$

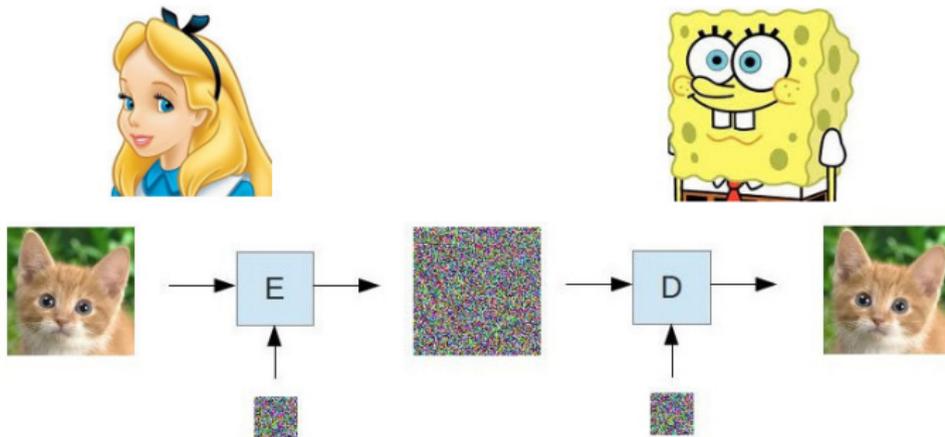
$$\implies \dim A = 0$$

Disons plutôt

$$\log \dim A = 1.$$

Cryptographie

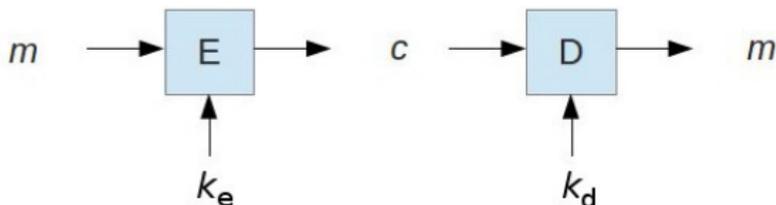
Chiffrement symétrique (AES)



- E et D : dimension finie par rapport à la *taille* de la clé
- attaque : dimension finie par rapport au *nombre* de clés

Cryptographie

Chiffrement asymétrique



- k_e publique, k_d privée : chiffrement à clé publique
- k_e privée, k_e publique : signature numérique

Cryptographie

RSA (1977)

Notons $a \equiv_n b$ pour dire que $n \mid (b - a)$.

Étant donné un (grand) entier n :

$$\begin{cases} E(e, m) \equiv_n m^e \\ D(d, c) \equiv_n c^d \end{cases}$$

Pour que ça fonctionne : on demande que

$$de \equiv_{\phi(n)} 1$$

où $\phi(n)$ est le nombre d'entiers entre 1 et n sans facteur commun avec n (*indicatrice d'Euler*)

Cryptographie

Exponentiation modulaire

Algorithme naïf d'exponentiation modulaire ($m^e \% n$) : $\text{dim} = \infty !$

```
IPython: home/chenevert
Fichier Édition Affichage Rechercher Terminal Aide
sage: n = ZZ.random_element(2^2048); n
106411881768371001152464110814034202672845951716242514973869111854747080690334964509462443088860986150621
880117176268384319411338691887108883834946659818941062363830404996839580180816121729691078990971605487750
229478200302571996547414709618034523465329344903953871028030382796650431723375760274488637640094734482118
0486576428833229882539168558079690135158779146653122483620801867828458829508746867319540295463380012105
873793602909573094802168795769992603348646014754318746686358148751665025259706780880149277685828464759646
8338831047596389073952443382847601615159352798624655951734993683329016052896465388539210967
sage: m = ZZ.random_element(2^2048); m
7280808935117355896139522449370554546386191173971911480888999523980113553662878285990799021448463021978
272304062675316180678340066665247485600611327845654529213958302878476959538452605496623392461878736269
76665419715834475947381594352063394900688838291782120320060740831258864002239993243448084193856912687457
869902420441211747971268204263944991074069025585874355193379857249012185087762561691060726083194210030469
57094012038079718302238246874560274558304981398642229686480205514347079381750728477381587592830378462436
241674539961067124973985815693265445104939290487172563512566117486145442316887196697806888
sage: e = ZZ.random_element(2^2048); e
1654771018611985504947830179883588107566591474251287915264809691402329172244182071681762585659929108530
966472107619461741735021562420434164020120095645778979094523712826854404525405198458160748411058097117034
949389171497295440969533188206950597881515228688450365129800522946263496318342548682410284567618933346903
24175621031080257878916805288343084551488865054783877647327789371253651345416303984975236056619243489406
63167912735945088406196266942706133478810471607765637488023503556854308101762574234173604413124655362677
0048545420581608438763347855899615762832468411789245921635119923716353430185381395984872994
sage: m^e % n
-----
RuntimeError                                Traceback (most recent call last)
<ipython-input-10-416a16ef3763> in <module>()
----> 1 m**e % n

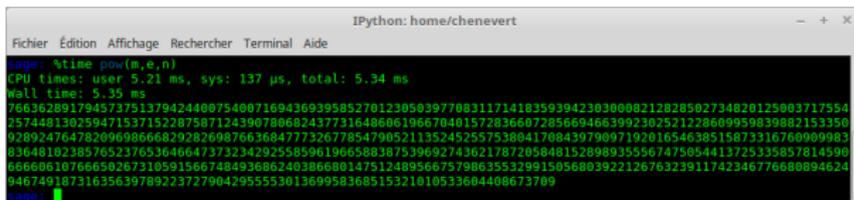
/opt/sage/sage-8.1/src/sage/rings/integer.pyx in sage.rings.integer.Integer.__pow__ (build/cythonized/sag
e/rings/integer.c:14183)()
   2058         elif mpz_cmp_si(self.value, -1) == 0:
   2059             return self if n % 2 else -self
-> 2060         raise RuntimeError("exponent must be at most %s" % sys.maxsize)
   2061
   2062         if nn == 0:
RuntimeError: exponent must be at most 9223372036854775807
sage: █
```

Cryptographie

Exponentiation rapide

Mais :

- pour la multiplication d'entiers de ℓ chiffres,
 - ▶ algorithme naïf : dimension 2
 - ▶ algorithme de Karatsuba : dimension 1,585
 - ▶ algorithme de Schönage-Strassen / Fürer : dimension 1
- l'exponentiation modulaire peut être réalisée à l'aide de $\log_2(e)$ mises au carré et multiplications modulaires.



```
IPython: home/chenevert
Fichier  Édition  Affichage  Recherche  Terminal  Aide
age: %time pow(0,e,n)
CPU times: user 5.21 ms, sys: 137 µs, total: 5.34 ms
Wall time: 5.35 ms
7663628917945737513794244007540071694369395852701230503977083117141835939423030008212828502734626125003717554
2574481302594715371522875871243907806824377316406061966704015728366072856694663992302521228609959839882153350
92892476478209698660292826987663684773267785479052113524525575380417084397909719201654638515873316760909983
836481023857652376536466473732342925585961966588387539692743621787205848152898935567475054413725335857814590
6666061076665026731059156674849368624038668014751248956675798635532991505680392212676323911742346776680894624
04674918731635639789223727904295555301369958368515321010533604408673709
age: █
```

Cryptographie

Attaquer RSA

Connaissant d ou e , on obtient l'autre comme coefficient de Bézout à l'aide de l'algorithme d'Euclide :

$$de + k\phi(n) = 1.$$

On a donc intérêt à ce que le calcul de $\phi(n)$ soit **long**, pour cela on prend $n = pq$ avec p et q deux grands nombres premiers.

Meilleure attaque à ce jour : *factoriser* n puis en déduire

$$\phi(n) = (p - 1)(q - 1).$$

Cryptographie

Notation L

Pour $d \in \mathbf{R}$ et $\alpha \in [0, 1]$,

$$L_\alpha(n) := \exp(d (\log n)^\alpha (\log \log n)^{1-\alpha}).$$

- $L_0(n) = (\log n)^d$
- $L_1(n) = n^d$
- en général $L_\alpha(n)$ est en quelque part entre les deux.

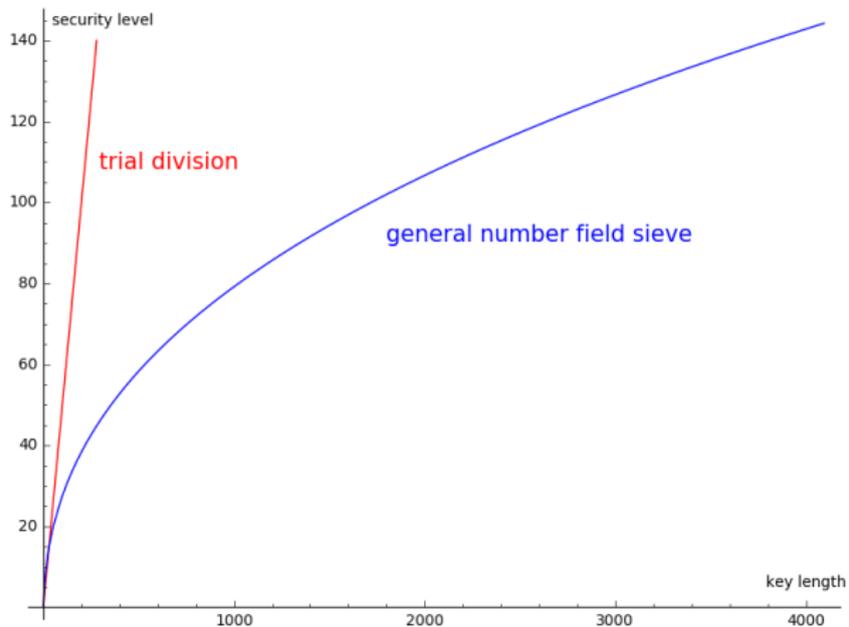
Cryptographie

Factorisation

- Recherche exhaustive de facteur : L_1 avec $d = \frac{1}{2}$
- Crible quadratique : $L_{\frac{1}{2}}$ avec $d = 1$
- Crible du corps de nombres généralisé : $L_{\frac{1}{3}}$ avec $d \approx 1,923$.

Cryptographie

Crible du corps de classe généralisé



Cryptographie

Logarithme discret

Autre façon d'utiliser l'exponentiation modulaire : $x \equiv_n g^y$

Problème du logarithme discret : retrouver $y = \ll \log_g(x) \gg$

Même complexité que la factorisation dans le cas arithmétique (DSA, Diffie-Hellman)

Mais : \sqrt{n} dans un groupe quelconque

\implies utilisation de courbes elliptiques (ECDSA, ECDH)

Menace quantique

État de l'art



Recuit simulé



IBM : 50 qubits



Google : 70 qubits

Menace quantique

Algorithme de Grover



Menace quantique

Algorithme de Grover

Étant donné une fonction « quelconque » $f : A \rightarrow B$ avec $|A| = n$ et $b \in f(A)$, la recherche d'une préimage $a \in A$ telle que $f(a) = b$ prend classiquement n évaluations.

Un ordinateur quantique pourrait en trouver une (*probablement*) avec seulement \sqrt{n} évaluations de f .

\implies doubler la taille des clés pour le chiffrement symétrique

Menace quantique

Algorithme de Shor

Factorise un entier n en nombre d'opérations

$$(\log n)^2(\log \log n)(\log \log \log n)$$

$$\implies \log \dim = 2!$$

Basé sur la capacité d'un ordinateur quantique à découvrir facilement la période d'une fonction périodique.

Résoud aussi le problème du logarithme discret.

Menace quantique

Algorithme de Shor



RSA (EC)DSA (EC)DH

Chiffrement post-quantique

Il est nécessaire de réfléchir dès maintenant à des alternatives asymétriques efficaces et sécurisées, même face à un éventuel attaquant quantique.

4 pistes principales à ce jour :

- à base de fonctions de hachage
- à base de codes correcteurs
- à base de réseaux
- équations quadratiques à plusieurs variables.

Processus de standardisation en cours par le NIST.

Chiffrement post-quantique

Chiffrement à base de réseaux

On se donne un réseau $\Lambda = \{ \sum_i a_i \mathbf{v}_i \mid a_i \in \mathbf{Z} \}$ où $\mathbf{v}_1, \dots, \mathbf{v}_n$ sont des vecteurs linéairement indépendants dans \mathbf{R}^n .

Chiffrement : on ajoute à un message $\mathbf{v} \in \Lambda$ un petit bruit $\mathbf{e} \in \mathbf{R}^n$:

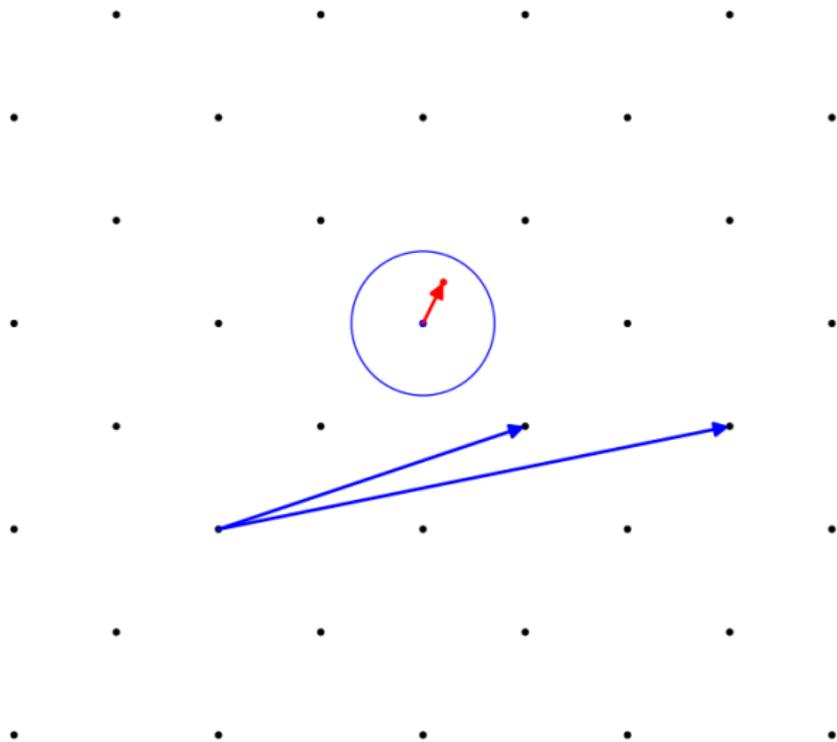
$$\mathbf{c} = \mathbf{v} + \mathbf{e}.$$

Déchiffrement : étant donné \mathbf{c} , on déchiffre au vecteur \mathbf{v} du réseau le plus proche.

Problème algorithmiquement difficile sans connaître une base adaptée.

Chiffrement post-quantique

Chiffrement à base de réseaux



Merci

et bon congrès !

