Introduction à la cryptographie

Vanessa VITSE

Université Grenoble Alpes

M1 Maths 2021

Plan du cours

7 sessions de 3h avec un mix cours/TP en SageMath

- Premiers concepts de la cryptographie, arithmétique modulaire et complexité
- Génération des nombres premiers et test de primalité
- Logarithme discret comme primitive pour la cryptographie à clef publique
- Factorisation et RSA
- Générateurs de nombres aléatoires
- Codes correcteurs d'erreurs

Plan du cours

7 sessions de 3h avec un mix cours/TP en SageMath
+ 1 TP évalué (3h) + 1 examen terminal (3h)

Il vous faut installer sagemath/Jupyter sur votre ordinateur

- Instructions pour l'installation sur `http://www.sagemath.org/download.html`
- Utilisation de Notebook Jupyter pour l'interface avec sage `https://jupyter.readthedocs.io/`
- Si rien ne marche... `https://sagecell.sagemath.org/`

Plan du cours

7 sessions de 3h avec un mix cours/TP en SageMath
+ 1 TP évalué (3h) + 1 examen terminal (3h)

Il vous faut installer sagemath/Jupyter sur votre ordinateur

- Instructions pour l'installation sur `http://www.sagemath.org/download.html`
- Utilisation de Notebook Jupyter pour l'interface avec sage `https://jupyter.readthedocs.io/`
- Si rien ne marche... `https://sagecell.sagemath.org/`

Polycopié de cours + slides + sujets de TP sur
`https://www-fourier.ujf-grenoble.fr/~viva/teaching.php`

Section 1

Premiers concepts en cryptographie

Missions fondamentales de la crypto

Alice et Bob veulent échanger de l'information confidentielle en présence d'une espionne Ève. Ils veulent

- *la confidentialité* de leur donnée privée qui transite sur des canaux de communication non sécurisés
- *l'authenticité* de ces données, plus précisément *l'intégrité du message*, ce qui signifie qu'Ève ne peut pas modifier le message à l'insu d'Alice et Bob
- *l'authentification* des interlocuteurs, i.e. qu'ils soient sûrs de *l'identité* de la personne à laquelle ils s'adressent

Missions fondamentales de la crypto

Alice et Bob veulent échanger de l'information confidentielle en présence d'une espionne Ève. Ils veulent

- *la confidentialité* de leur donnée privée qui transite sur des canaux de communication non sécurisés
- *l'authenticité* de ces données, plus précisément *l'intégrité du message*, ce qui signifie qu'Ève ne peut pas modifier le message à l'insu d'Alice et Bob
- *l'authentification* des interlocuteurs, i.e. qu'ils soient sûrs de *l'identité* de la personne à laquelle ils s'adressent

Outils

- Des protocoles publiques de chiffrement/déchiffrement
- Des clefs **secrètes** pour retrouver la donnée sensible

Missions fondamentales de la crypto

Alice et Bob veulent échanger de l'information confidentielle en présence d'une espionne Ève. Ils veulent

- *la confidentialité* de leur donnée privée qui transite sur des canaux de communication non sécurisés
- *l'authenticité* de ces données, plus précisément *l'intégrité du message*, ce qui signifie qu'Ève ne peut pas modifier le message à l'insu d'Alice et Bob
- *l'authentification* des interlocuteurs, i.e. qu'ils soient sûrs de *l'identité* de la personne à laquelle ils s'adressent

Outils

- Des protocoles publiques de chiffrement/déchiffrement
- Des clefs **secrètes** pour retrouver la donnée sensible

la sécurité doit reposer sur la clef secrète

Cryptographie symmetrique

Alice et Bob partagent déjà une clef secrète commune

Définition

Chiffrement symétrique = une paire (E, D) d'algorithme publique tels que

- $E : (k, m) \in \mathcal{K} \times \mathcal{M} \rightsquigarrow c \in \mathcal{C}$
- $D : (k, c) \in \mathcal{K} \times \mathcal{C} \mapsto m \in \mathcal{M}$
- $\forall k \in \mathcal{K}, \forall m \in \mathcal{M},\ D(k, E(k, m)) = m$ (correction de l'algorithme)

Cryptographie symmetrique

Alice et Bob partagent déjà une clef secrète commune

Définition

Chiffrement symétrique = une paire (E, D) d'algorithme publique tels que

- $E : (k, m) \in \mathcal{K} \times \mathcal{M} \rightsquigarrow c \in \mathcal{C}$
- $D : (k, c) \in \mathcal{K} \times \mathcal{C} \mapsto m \in \mathcal{M}$
- $\forall k \in \mathcal{K}, \forall m \in \mathcal{M},\ D(k, E(k, m)) = m$ (correction de l'algorithme)

- généralement $\mathcal{M} = \mathcal{K} = \mathcal{C} = \{0, 1\}^n$
- E peut être **non-déterministe**, i.e. il peut produire différents chiffrés pour une même entrée (k, m)
- D est déterministe, donc réalise une fonction mathématique
- hypothèse de sécurité : il doit être **difficile** de retrouver m à partir de c sans connaître k

Un exemple classique : le chiffrement “one-time-pad”

One-time-pad

- $\mathcal{M} = \mathcal{K} = \mathcal{C} = \mathbb{F}_2^n$ (ev sur le corps à 2 éléments)
- $E(k, m) = k \oplus m$ (XOR = addition sans retenue)
- $D(k, c) = k \oplus c$

Un exemple classique : le chiffrement "one-time-pad"

One-time-pad

- $\mathcal{M} = \mathcal{K} = \mathcal{C} = \mathbb{F}_2^n$ (ev sur le corps à 2 éléments)
- $E(k, m) = k \oplus m$ (XOR = addition sans retenue)
- $D(k, c) = k \oplus c$

Vérifier la correction de l'algorithme avec les propriétés arithématiques de $(\mathbb{F}_2^n, \oplus)$:

$$x \oplus y = y \oplus x, \quad x \oplus (y \oplus z) = (x \oplus y) \oplus z, \quad x \oplus 0_{\mathbb{F}_2^n} = x \quad \text{and} \quad x \oplus x = 0_{\mathbb{F}_2^n}.$$

Sécurité parfaite

Le one-time-pad est un cryptosystème optimal :

Sécurité parfaite

Soit (E, D) un cryptosytème sur $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, K, M, C des variables aléatoires telles que $M = D(K, C)$, où K est uniformément distribué sur $\mathcal{K}$ et K, M indépendants
Le cryptosystème (E, D) est **parfaitement sûr** si pour tout $(m, c) \in \mathcal{M} \times \mathcal{C}$,

$$\Pr[M = m \mid C = c] = \Pr[M = m].$$

Sécurité parfaite

Le one-time-pad est un cryptosystème optimal :

Sécurité parfaite

Soit (E, D) un cryptosytème sur $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, K, M, C des variables aléatoires telles que $M = D(K, C)$, où K est uniformément distribué sur $\mathcal{K}$ et K, M indépendants
Le cryptosystème (E, D) est **parfaitement sûr** si pour tout $(m, c) \in \mathcal{M} \times \mathcal{C}$,

$$\Pr[M = m \mid C = c] = \Pr[M = m].$$

Autrement dit M (qui n'est pas uniformément distribué) et C sont **indépendants**

$\rightarrow$ la connaissance de c ne donne aucune information sur m

Sécurité parfaite

Le one-time-pad est un cryptosystème optimal :

Sécurité parfaite

Soit (E, D) un cryptosytème sur $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, K, M, C des variables aléatoires telles que $M = D(K, C)$, où K est uniformément distribué sur $\mathcal{K}$ et K, M indépendants
Le cryptosystème (E, D) est **parfaitement sûr** si pour tout $(m, c) \in \mathcal{M} \times \mathcal{C}$,

$$\Pr[M = m \mid C = c] = \Pr[M = m].$$

Autrement dit M (qui n'est pas uniformément distribué) et C sont **indépendants**

$\rightarrow$ la connaissance de c ne donne aucune information sur m

Le cryptosystème one-time-pad est parfaitement sûr (et c'est le seul cryptosystème à l'être !)

Sécurité parfaite

Le théorème de Shannon

Soit un cryptosystème (E, D) défini sur $\mathcal{K}, \mathcal{M}, \mathcal{C}$ parfaitement sûr, alors $|\mathcal{K}| \geqslant |\mathcal{M}|$.

Sécurité parfaite

Le théorème de Shannon

Soit un cryptosystème (E, D) défini sur $\mathcal{K}, \mathcal{M}, \mathcal{C}$ parfaitement sûr, alors $|\mathcal{K}| \geqslant |\mathcal{M}|$.

Problème de la distribution et du stockage de clefs (qui sont de longueur identique à celle des messages clairs/chiffrés)

- besoin de définir des niveaux de sécurité moins importants : retrouver de l'information sur m à partir de c ne devrait pas être calculatoirement faisable dans un monde réel avec des ressources en calcul limitées
- remplacer le secret du one-time-pad par une *graine* courte permettant de produire une séquence pseudo-aléatoire (PRNG)
  $\rightsquigarrow$ *chiffrement par flux*

Dans la vraie vie c'est plus compliqué !

- Le one-time-pad devient totalement vulnérable s'il n'est pas utilisé correctement

  par exemple si Bob utilise deux fois la même clef secrète pour chiffrer deux messages différents...

- Le one-time-pad est *malléable* : un attaquant peut provoquer un changement prédictible dans le texte clair

  par exemple si Eve change $c = E(k, m)$ en $c' = c \oplus \delta$, alors Alice déchiffre $D(k, c') = m \oplus \delta$...

Chiffrement à clef publique

Limites de la cryptographie symétrique

Alice et Bob doivent en premier lieu s'échanger une clef secrète

Chiffrement à clef publique

Limites de la cryptographie symétrique

Alice et Bob doivent en premier lieu s'échanger une clef secrète

Idée de la crypto à clef publique : reproduire le concept de la boîte aux lettres classique

Schémas de chiffrement à clef publique (1976-1977)

Deux clefs sont nécessaires :

- la clef *publique* d'Alice (connue de tout le monde) qui est utilisée pour chiffrer
- la clef *privée* d'Alice (connue seulement d'elle) qui est utilisée pour déchiffrer

Ces schémas reposent sur des **pbs mathématiques/algorithmiques difficiles**

$\rightsquigarrow$ besoin de définir une notion de complexité

Section 2

Arithmétique et complexité modulaire

Arithmétique sur les grands entiers

Contexte cryptographique

- messages ayant une taille de 256 à 2048 bits >> 64 bits (taille des registres des ordinateurs modernes)
  → **bibliothèques spécifiques** pour les opérations sur de grands entiers

Arithmétique sur les grands entiers

Contexte cryptographique

- messages ayant une taille de 256 à 2048 bits >> 64 bits (taille des registres des ordinateurs modernes)
  → **bibliothèques spécifiques** pour les opérations sur de grands entiers
- addition/multiplication ne peuvent plus être considérées comme des opérations à temps constant quand le nombre de bits n grandit
  → on doit savoir évaluer la **complexité** d'un algorithme

Arithmétique sur les grands entiers

Contexte cryptographique

- messages ayant une taille de 256 à 2048 bits >> 64 bits (taille des registres des ordinateurs modernes)
  → **bibliothèques spécifiques** pour les opérations sur de grands entiers
- addition/multiplication ne peuvent plus être considérées comme des opérations à temps constant quand le nombre de bits n grandit
  → on doit savoir évaluer la **complexité** d'un algorithme

Notation grand O

Soient f, g deux fonctions réelles, avec $g > 0$. On note $f = O(g)$ si

$$\exists C > 0, \forall x \text{ suffisamment grand, } |f(x)| \leqslant Cg(x).$$

Notation grand O

Soient f, g deux fonctions réelles, avec $g > 0$. On note $f = O(g)$ si
$$\exists C > 0, \forall x \text{ suffisamment grand, } |f(x)| \leqslant Cg(x).$$

$f = O(1) \iff f$ est bornée (au voisinage de ∞)

$x^a + \log(x)^b = O(x^a)$ pour tout $a, b, x > 0$

⚠ $f = O(g)$ et $h = O(g) \nRightarrow f = h$ ou $f = O(h)$

Notation grand O

Soient f, g deux fonctions réelles, avec $g > 0$. On note $f = O(g)$ si
$$\exists C > 0, \forall x \text{ suffisamment grand, } |f(x)| \leqslant Cg(x).$$

$f = O(1) \iff f$ est bornée (au voisinage de ∞)

$x^a + \log(x)^b = O(x^a)$ pour tout $a, b, x > 0$

⚠ $f = O(g)$ et $h = O(g) \nRightarrow f = h$ ou $f = O(h)$

Toutes les complexités sont exprimées en fonction de la *taille des entrées*

Exemples à connaître d'opérations sur des entiers de taille n

- Addition/soustraction en $O(n)$
- Multiplication/division (méthode standard) en $O(n^2)$
  (mieux : méthode de Karatsuba en $O(n^{\log_2(3)}) = O(n^{1.584})$)

Notation grand O

Soient f, g deux fonctions réelles, avec $g > 0$. On note $f = O(g)$ si
$$\exists C > 0, \forall x \text{ suffisamment grand, } |f(x)| \leqslant Cg(x).$$

$f = O(1) \iff f$ est bornée (au voisinage de ∞)

$x^a + \log(x)^b = O(x^a)$ pour tout $a, b, x > 0$

⚠ $f = O(g)$ et $h = O(g) \not\Rightarrow f = h$ ou $f = O(h)$

Toutes les complexités sont exprimées en fonction de la *taille des entrées*

Exemples à connaître d'opérations sur des entiers de taille n

- Addition/soustraction en $O(n)$
- Multiplication/division (méthode standard) en $O(n^2)$
  (mieux : méthode de Karatsuba en $O(n^{\log_2(3)}) = O(n^{1.584})$)

Exercice : coût de la multiplication de deux polynômes de degré d avec des coefficients plus petits que B

Division euclidienne

Pour $a, b \in \mathbb{Z},\ b \neq 0$, il existe un unique couple $(q, r) \in \mathbb{Z}^2$ tel que $a = bq + r$ et $0 \leqslant r < |b|$.
L'entier r est appelé *reste* de la division, et q le *quotient*.

Division euclidienne

Pour $a, b \in \mathbb{Z}$, $b \neq 0$, il existe un unique couple $(q, r) \in \mathbb{Z}^2$ tel que $a = bq + r$ et $0 \leqslant r < |b|$.
L'entier r est appelé *reste* de la division, et q le *quotient*.

- b *divise* a si le reste est 0, noté $b|a$
- Soient $x, y, n \in \mathbb{Z}$, $n \neq 0$; x est *congruent* à y *modulo* n si leurs restes dans la division par n sont les mêmes, noté $x = y \bmod n$

Division euclidienne

Pour $a, b \in \mathbb{Z}$, $b \neq 0$, il existe un unique couple $(q, r) \in \mathbb{Z}^2$ tel que $a = bq + r$ et $0 \leqslant r < |b|$.
L'entier r est appelé *reste* de la division, et q le *quotient*.

- b *divise* a si le reste est 0, noté $b|a$
- Soient $x, y, n \in \mathbb{Z}$, $n \neq 0$; x est *congruent* à y *modulo* n si leurs restes dans la division par n sont les mêmes, noté $x = y \bmod n$

Relation de congruence

- relation d'équivalence $\rightarrow \mathbb{Z}/n\mathbb{Z}$ l'ensemble des classes d'équivalence modulo n
- compatibilité avec add/mult $\rightarrow \mathbb{Z}/n\mathbb{Z}$ est un anneau commutatif
- $a = b \bmod n \iff ac = bc \bmod nc$
- $a = b \bmod mn \implies (a = b \bmod m \text{ et } a = b \bmod n)$

Exponentiation modulaire

Question :

étant donnés $g \in \mathbb{Z}/n\mathbb{Z}$ (ou plus généralement dans un groupe G) et $e \in \mathbb{N}^*$, comment calculer $g^e \bmod n$?

Exponentiation modulaire

Question :

étant donnés $g \in \mathbb{Z}/n\mathbb{Z}$ (ou plus généralement dans un groupe G) et $e \in \mathbb{N}^*$, comment calculer $g^e \bmod n$?

Approche naïve :
multiplier par g un nombre total de $e-1$ fois, en réduisant modulo n à chaque étape
$\rightarrow$ complexité en $O(e \log(n)^2)$, donc exponentielle en la taille de e...

Exponentiation modulaire

Question :

étant donnés $g \in \mathbb{Z}/n\mathbb{Z}$ (ou plus généralement dans un groupe G) et $e \in \mathbb{N}^*$, comment calculer $g^e \bmod n$?

Approche naïve :
multiplier par g un nombre total de $e-1$ fois, en réduisant modulo n à chaque étape
$\rightarrow$ complexité en $O(e \log(n)^2)$, donc exponentielle en la taille de e...

Meilleure idée :
Soit $e = \sum_i \epsilon_i 2^i$ l'expression binaire de e. Alors

$$g^e = \prod_{\epsilon_i = 1} g^{2^i} \bmod n.$$

Algorithm 1: Algorithme d'exponentiaition rapide de "droite-à-gauche"

Input : $g \in \mathbb{Z}/n\mathbb{Z},\ e, n \in \mathbb{N}^*$

Output: $g^e \bmod n$

$res \leftarrow 1$

$t \leftarrow g$

while $e \neq 0$ **do**

  if *e is odd* **then**

    $res \leftarrow res \cdot t \bmod n$

  $e \leftarrow \lfloor e/2 \rfloor$

  $t \leftarrow t^2 \bmod n$

return *res*

Algorithm 1: Algorithme d'exponentiaition rapide de "droite-à-gauche"

Input : $g \in \mathbb{Z}/n\mathbb{Z},\ e, n \in \mathbb{N}^*$
Output: $g^e \bmod n$

$res \leftarrow 1$
$t \leftarrow g$
while $e \neq 0$ **do**
  if *e is odd* **then**
    $res \leftarrow res \cdot t \bmod n$
  $e \leftarrow \lfloor e/2 \rfloor$
  $t \leftarrow t^2 \bmod n$
return *res*

Complexité

$O(\log e)$ multiplications dans $\mathbb{Z}/n\mathbb{Z}$ + $\log e$ mises au carré $g^{2^{i+1}} = \left(g^{2^i}\right)^2$

Complexité totale en $O(\log e (\log n)^2)$

En lisant les bits de gauche à droite...

$$g^e = \begin{cases} (g^{e/2})^2 & \text{for even } e, \\ g \cdot (g^{(e-1)/2})^2 & \text{for odd } e. \end{cases}$$

Algorithm 2: Algorithme d'exponentiaition rapide de "gauche-à-droite"

Input : $g \in \mathbb{Z}/n\mathbb{Z}$, $e, n \in \mathbb{N}^*$
Output: $g^e \bmod n$
if $e == 0$ **then return** 1
$res \leftarrow g$
$t \leftarrow \lfloor \log_2(e) \rfloor$ $B \leftarrow$ liste des bits de e $(e = \sum_{i=0}^{t} B[i]2^i,\ B[t] = 1)$
while $t > 0$ **do**
    $t \leftarrow t - 1$
    $res \leftarrow res^2 \bmod n$
    if $B[t] == 1$ **then** $res \leftarrow res \cdot g \bmod n$
return res

Complexités polynomiales vs exponentielles

Quelques chiffres

- un ordinateur portable 4-coeurs à 2.5 GHz peut calculer environ 10 milliards d'opérations en arithmétique flottante par seconde : 2^{36} FLOPS
- un cluster d'ordinateurs modeste : 2^{40} FLOPS
- des supercalculateurs : 2^{50} to 2^{56} FLOPS

Complexités polynomiales vs exponentielles

Quelques chiffres

- un ordinateur portable 4-coeurs à 2.5 GHz peut calculer environ 10 milliards d'opérations en arithmétique flottante par seconde : 2^{36} FLOPS
- un cluster d'ordinateurs modeste : 2^{40} FLOPS
- des supercalculateurs : 2^{50} to 2^{56} FLOPS

- Une simple exponentiation d'entiers de 80 bits integers prendra plusieurs décennies sur un supercalculateur avec une méthode naïve
- Une exponentiation d'entiers de 1000 bits se fera en quelques millisecondes sur un ordinateur portable avec un algorithme d'exponentiation rapide

Complexités polynomiales vs exponentielles

Quelques chiffres

- un ordinateur portable 4-coeurs à 2.5 GHz peut calculer environ 10 milliards d'opérations en arithmétique flottante par seconde : 2^{36} FLOPS
- un cluster d'ordinateurs modeste : 2^{40} FLOPS
- des supercalculateurs : 2^{50} to 2^{56} FLOPS

- Une simple exponentiation d'entiers de 80 bits integers prendra plusieurs décennies sur un supercalculateur avec une méthode naïve
- Une exponentiation d'entiers de 1000 bits se fera en quelques millisecondes sur un ordinateur portable avec un algorithme d'exponentiation rapide

Écart énorme entre des complexités en $O(n)$ et $O(2^n)$

Complexités polynomiales vs exponentielles

En utilisant un ordinateur qui fait 2^{30} opérations élémentaires par sec, combien de temps faut-il pour exécuter $f(n)$ opérations ?

n	$\log_2(n)$	n	$n\log_2(n)$	n^2	n^3	2^n	$n!$
10	3 ns	9 ns	30 ns	90 ns	0.9 μs	0.9 μs	3 ms
20	4 ns	18 ns	80 ns	0.4 μs	7 μs	1 ms	70 years
30	4.5 ns	28 ns	140 ns	0.8 μs	25 μs	1 s	$>$ age of universe
40	5 ns	37 ns	190 ns	1.5 μs	60 μs	1024 s	–
50	5.2 ns	46 ns	260 ns	2.3 μs	0.1 ms	12 days	–
60	5.5 ns	55 ns	330 ns	3.3 μs	0.2 ms	34 years	–
80	5.8 ns	75 ns	470 ns	6 μs	0.4 ms	35 million years	–
100	6.2 ns	93 ns	620 ns	9 μs	0.9 ms	$>$ age of universe	–
200	7.1 ns	186 ns	1.5 μs	37 μs	7 ms	–	–
1000	9.2 ns	0.9 μs	9 μs	1 ms	1 s	–	–
10000	12 ns	9 μs	0.1 ms	100 ms	1000 s	–	–

Complexités polynomiales vs exponentielles

En utilisant un ordinateur qui fait 2^{30} opérations élémentaires par sec, combien de temps faut-il pour exécuter $f(n)$ opérations ?

n	$\log_2(n)$	n	$n\log_2(n)$	n^2	n^3	2^n	$n!$
10	3 ns	9 ns	30 ns	90 ns	0.9 μs	0.9 μs	3 ms
20	4 ns	18 ns	80 ns	0.4 μs	7 μs	1 ms	70 years
30	4.5 ns	28 ns	140 ns	0.8 μs	25 μs	1 s	$>$ age of universe
40	5 ns	37 ns	190 ns	1.5 μs	60 μs	1024 s	–
50	5.2 ns	46 ns	260 ns	2.3 μs	0.1 ms	12 days	–
60	5.5 ns	55 ns	330 ns	3.3 μs	0.2 ms	34 years	–
80	5.8 ns	75 ns	470 ns	6 μs	0.4 ms	35 million years	–
100	6.2 ns	93 ns	620 ns	9 μs	0.9 ms	$>$ age of universe	–
200	7.1 ns	186 ns	1.5 μs	37 μs	7 ms	–	–
1000	9.2 ns	0.9 μs	9 μs	1 ms	1 s	–	–
10000	12 ns	9 μs	0.1 ms	100 ms	1000 s	–	–

Premier exemple de fonction à sens unique (pour la crypto asymétrique)

- les algorithmes d'exponentiation rapide ont une complexité polynomiale
- pas d'algo efficace (en 2021) calculant x à partir de $n, g, g^x \bmod n$ quand n premier grand (*problème du logarithme discret*)

Algorithme d'Euclide étendu

Quid de la division dans un anneau non intègre $\mathbb{Z}/n\mathbb{Z}$?

Algorithme d'Euclide étendu

Quid de la division dans un anneau non intègre $\mathbb{Z}/n\mathbb{Z}$?

pgcd

Soient a et b deux entiers positifs tels que $a > b$, alors

$$\gcd(a, b) = \gcd(b, a \bmod b).$$

Algorithme d'Euclide étendu

Quid de la division dans un anneau non intègre $\mathbb{Z}/n\mathbb{Z}$?

pgcd

Soient a et b deux entiers positifs tels que $a > b$, alors

$$\gcd(a, b) = \gcd(b, a \bmod b).$$

Soient $r_0 := a$ et $r_1 := b$, on calcule itérativement

$r_0 = r_1\, q_1 + r_2$ avec $0 \leqslant r_2 < |r_1| \rightarrow \gcd(a, b) = \gcd(r_1, r_2)$

$r_1 = r_2\, q_2 + r_3$ avec $0 \leqslant r_3 < r_2 \rightarrow \gcd(r_1, r_2) = \gcd(r_2, r_3)$

$\vdots$

$r_{n-1} = r_n\, q_n + r_{n+1}$ avec $r_{n+1} = 0 \rightarrow \gcd(r_{n-1}, r_n) = r_n$

$\gcd(a, b)$ est égal au dernier reste non nul r_n

Algorithme d'Euclide

Complexité

L'algorithme d'Euclide est en $O((\log N)^2)$

Algorithm 3: Algorithme d'Euclide

Input : $a, b \in \mathbb{N}, a > b$
Output: $\gcd(a, b)$
while $b > 0$ **do**
    $r \leftarrow a \bmod b$
    $a \leftarrow b$
    $b \leftarrow r$
return a

Algorithme d'Euclide étendu et inverse modulaire

Lemme de Bézout

Soient $a, b \in \mathbb{Z}$, il existe $u, v \in \mathbb{Z}$ tels que $au + bv = \gcd(a, b)$.

Algorithme d'Euclide étendu et inverse modulaire

Lemme de Bézout

Soient $a, b \in \mathbb{Z}$, il existe $u, v \in \mathbb{Z}$ tels que $au + bv = \gcd(a, b)$.

Application directe de Bézout :

$$a \bmod n \text{ inversible} \iff \gcd(a, n) = 1$$

Algorithm 4: Calcul de l'inverse modulo n

Input : $a \in \mathbb{Z}$, $n \in \mathbb{N}^*$
Output: $a^{-1} \bmod n$

$u_0 \leftarrow 1 \quad u_1 \leftarrow 0$
while $b \neq 0$ **do**
    $tmp \leftarrow a \quad a \leftarrow b$
    $b \leftarrow tmp\%a \quad q \leftarrow tmp/a$
    $tmp \leftarrow u_0 - qu_1 \quad u_0 \leftarrow u_1 \quad u_1 \leftarrow tmp$
return u_0

Éléments inversibles modulo n

Fonction indicatrice d'Euler

$$\forall n \in \mathbb{N}^*,\ \varphi(n) = |(\mathbb{Z}/n\mathbb{Z})^\times|.$$

$\varphi(n)$ est le nombre de générateur de n'importe quel sous-groupe cyclique de cardinalité n

Éléments inversibles modulo n

Fonction indicatrice d'Euler

$$\forall n \in \mathbb{N}^*,\ \varphi(n) = |(\mathbb{Z}/n\mathbb{Z})^\times|.$$

$\varphi(n)$ est le nombre de générateur de n'importe quel sous-groupe cyclique de cardinalité n

Calcul de $\varphi(n)$ facile si la factorisation est connue :

- $\varphi(mn) = \varphi(m)\varphi(n)$ pour tout entier naturel n, m.
- $\varphi(p^e) = p^e - p^{e-1} = p^e(1 - 1/p)$ pour tout premier p et tout entier positif e.
- $\varphi(n) = n\prod_{i=1}^{r}(1 - 1/p_i)$ où $n = p_1^{e_1} \dots p_k^{e_k}$ est le factorisation de n en premiers distincts.
- $n = \sum_{d|n} \varphi(d)$.

Autre application de l'algo d'Euclide étendu

Théorème des restes chinois – CRT

Soient n, m deux entiers premiers entre eux et a, b deux entiers. Alors le system

$$\begin{cases} x = a \bmod n \\ x = b \bmod n \end{cases}$$

admet une unique solution $\bmod mn$.