Sécurité du logiciel, septième cours

Calculer avec des données chiffrées ou privées

Xavier Leroy

2022-04-21

Collège de France, chaire de sciences du logiciel
xavier.leroy@college-de-france.fr

L'approche cryptographique

Un certain nombre de primitives cryptographiques

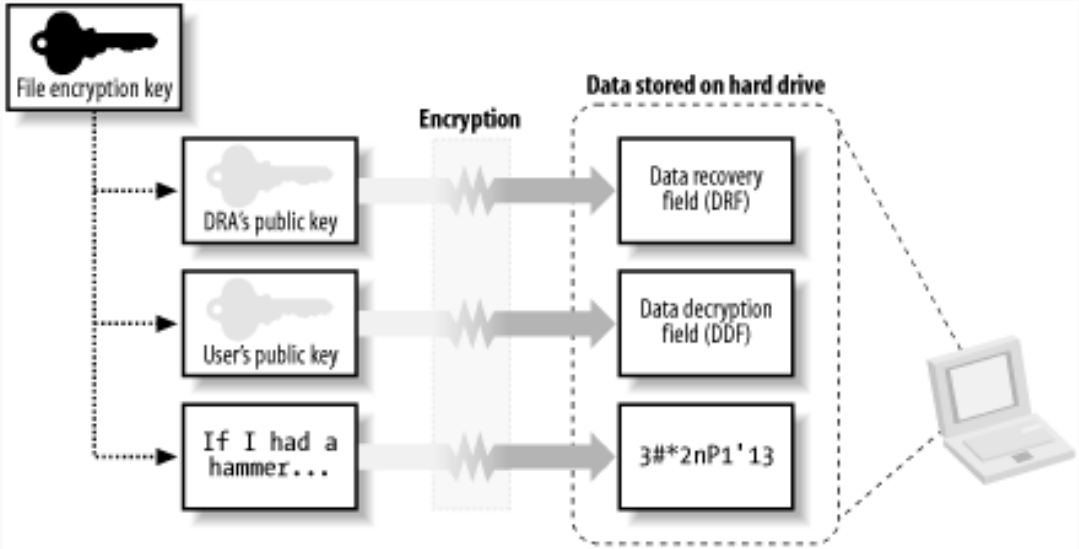
- chiffrement (symétrique ou à clé publique)
- signature (clé publique)
- empreintes (*hashes*)
- etc.

que l'on combine et applique

pour garantir la confidentialité et l'intégrité des informations

au repos (stockage) et en transit (réseaux).

Exemple : chiffrement de fichiers



Chiffrement avec une clé secrète, tirée au hasard, sur-chiffrée avec les mots de passe des utilisateurs autorisés.

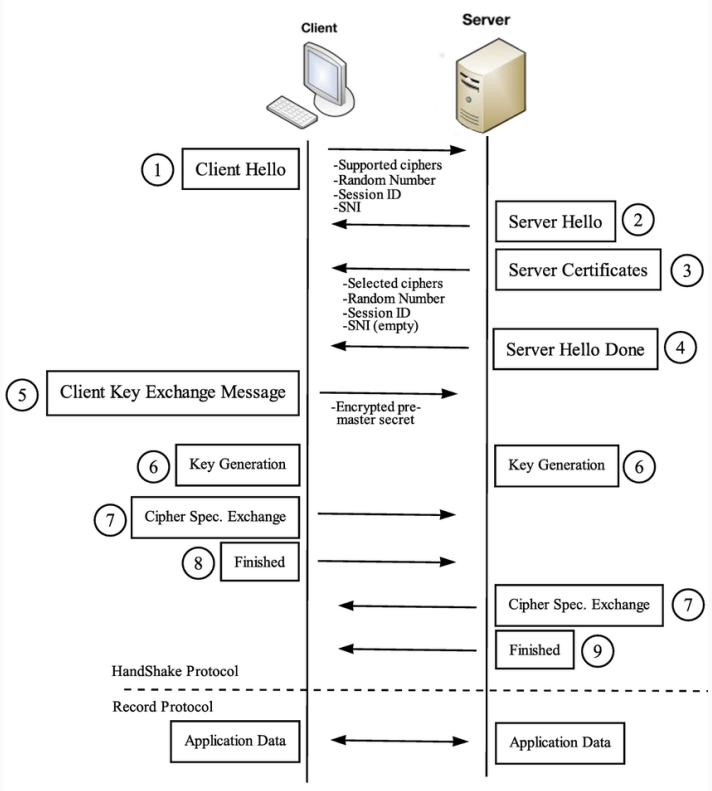
Meilleure protection connue contre les attaques bas-niveau (on vole la machine, on vole le disque, on démarre un autre système).

Seul moyen connu pour effacer instantanément une grande quantité de données.

Exemple : les protocoles réseaux TLS, SSH, Signal, …

Communication point-à-point sur Internet avec

- chiffrement et authentification des données échangées (pas d'écoute, d'injection, de rejeu des paquets réseau);
- authentification (par certificats) de l'interlocuteur (pas d'impersonation ni de *man in the middle*).



Seule protection connue contre un attaquant qui contrôlerait une partie du réseau.

Quid des données pendant le calcul ?

Les logiciels classiques opèrent sur les données en clair.

P.ex. un moteur de base de données : pour exécuter les requêtes il ≪faut bien≫ avoir accès aux données en clair.

Mais c'est difficile de garantir la confidentialité des données pendant le calcul :

- flux d'information mal maîtrisés ;
- canaux indirects : temps, cache, spéculations, …

Ce cours

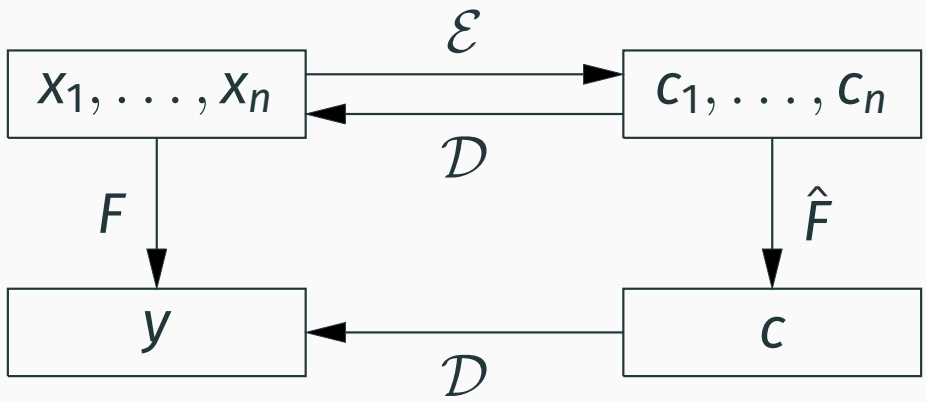
Des solutions cryptographiques au problème de calculer en préservant la confidentialité des données.

Deux exemples détaillés :

- Le chiffrement homomorphe : pour calculer directement sur les données chiffrées, sans les déchiffrer.
- Le calcul multipartite sécurisé : plusieurs participants calculent ensemble une fonction de leurs données privées, sans révéler ces données aux autres.

Chiffrement homomorphe

Le chiffrement homomorphe



Soit F une fonction à n arguments : $y = F(x_1, \ldots, x_n)$.

Un chiffrement $\mathcal{E}, \mathcal{D}$ est homomorphe pour la fonction F s'il existe une fonction $\hat{F}$ telle que

$$\mathcal{D}(\hat{F}(c_1, \ldots, c_n)) = F(\mathcal{D}(c_1), \ldots, \mathcal{D}(c_n))$$

pour tous les arguments chiffrés $c_1, \ldots, c_n$.

En corollaire, on a, pour tous les arguments en clair $x_1, \ldots, x_n$,

$$\mathcal{D}(\hat{F}(\mathcal{E}(x_1), \ldots, \mathcal{E}(x_n)) = F(x_1, \ldots, x_n)$$

Exemple : RSA est homomorphe pour la multiplication

Chiffrement RSA : (e, N clé publique ; d clé secrète)

$$\mathcal{E}(m) = m^e \bmod N \qquad \mathcal{D}(c) = c^d \bmod N$$

Si c_1, c_2 sont deux messages chiffrés,

$$\mathcal{D}(c_1) \cdot \mathcal{D}(c_2) = c_1^d \cdot c_2^d = (c_1 \cdot c_2)^d = \mathcal{D}(c_1 \cdot c_2) \pmod N$$

L'opérateur F de multiplication modulo N a pour opérateur homomorphe $\hat{F}$ la multiplication modulo N.

Application : dépouillement d'une élection

Votes v_i : 1 pour blanc, 2 pour Alice, 3 pour Bob.

Chaque électeur i chiffre son vote v_i avec la clé publique de l'autorité du vote.

L'opérateur du vote collecte les votes $\mathcal{E}(v_i)$ et en fait le produit

$$P \stackrel{def}{=} \mathcal{E}(v_1) \cdots \mathcal{E}(v_n) \pmod{N}$$

Le produit chiffré P est transmis à l'autorité du vote, qui déchiffre :

$$\mathcal{D}(P) = \mathcal{D}(\mathcal{E}(v_1)) \cdots \mathcal{D}(\mathcal{E}(v_n)) = v_1 \cdots v_n \pmod{N}$$

Si $v_1 \cdots v_n < N$, ce résultat $\mathcal{D}(P)$ est de la forme $2^a \cdot 3^b$
où a est le nombre de voix pour Alice et b celui pour Bob.

(Attention : très mauvais protocole, ne pas utiliser !)

Le chiffrement d'El Gamal homomorphe pour l'addition

Un groupe fini G d'ordre q.

Clé secrète : $x \in \{1, \ldots, q-1\}$.

Clé publique : un générateur g de G et $h \stackrel{def}{=} g^x$.

Chiffrement : $\mathcal{E}(m) = (c_1, c_2)$ avec

- $y \in \{1, \ldots, q-1\}$ choisi aléatoirement
- $s = h^y$ le secret partagé
- $c_1 = g^y$ et $c_2 = g^m \cdot s$.

Déchiffrement : $\mathcal{D}(c_1, c_2) = m$ où

- on retrouve le secret partagé s en calculant c_1^x
- on retrouve g^m en calculant $c_2 \cdot s^{-1}$
- on retrouve m par logarithme discret en temps $O(\sqrt{m})$.

Le chiffrement d'El Gamal homomorphe pour l'addition

Homomorphisme :

$$\begin{aligned}
\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) &= (g^{y_1} \cdot g^{y_2}, (g^{m_1} \cdot h^{y_1}) \cdot (g^{m_2} \cdot h^{y_2})) \\
&= (g^{y}, g^{m_1+m_2} \cdot h^{y}) \qquad \text{(avec } y = y_1 + y_2 \bmod q\text{)} \\
&= \text{un chiffré de } m_1 + m_2
\end{aligned}$$

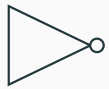
Donc, l'opération homomorphe de l'addition des clairs est la multiplication des chiffrés.

Propriété utilisée dans le système de vote électronique Belenios !

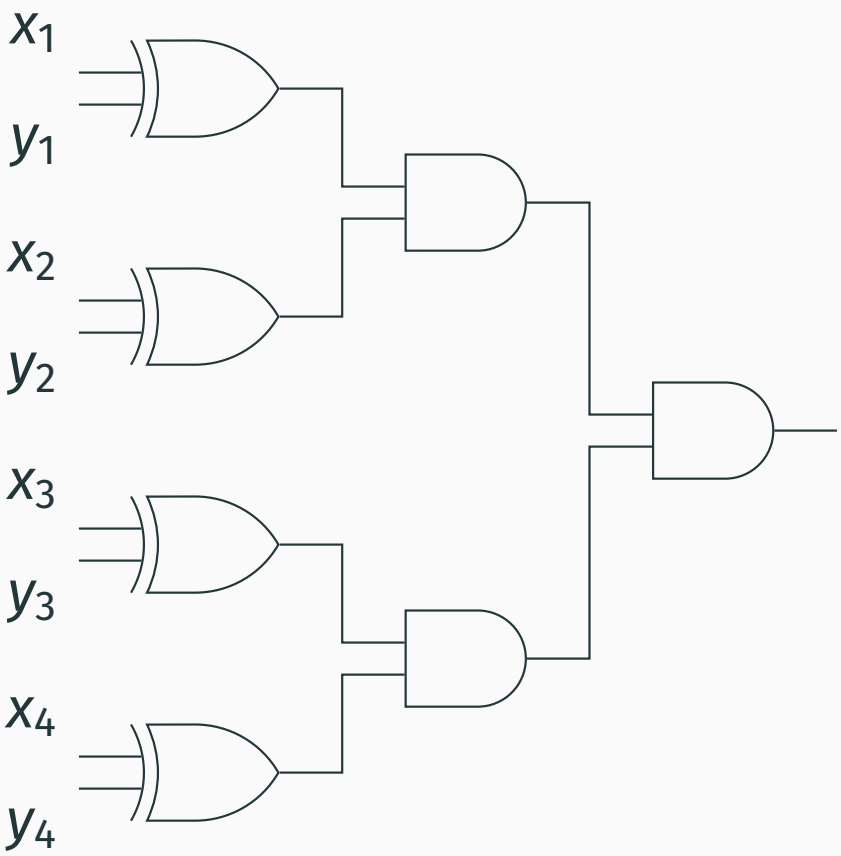
Chiffrement complètement homomorphe et circuits booléens

Étape cruciale : trouver un chiffrement homomorphe à la fois pour l'addition (modulo 2) et pour la multiplication (modulo 2).

Un tel chiffrement est dit complètement homomorphe
(FHE, *Fully Homomorphic Encryption*)
car il permet d'évaluer n'importe quel circuit booléen.

Porte logique		Calcul arithmétique
ou exclusif		$(a+b) \bmod 2$
et		$a \cdot b$
non		$1-a = (1+a) \bmod 2$
ou		$1-(1-a)(1-b)$

Exemple : un comparateur



Évalué de manière homomorphe, ce circuit peut servir à faire une recherche dans une base de données chiffrées.

Chiffrer = ajouter du bruit

On va utiliser des algorithmes de chiffrement qui s'appuient sur l'idée suivante :

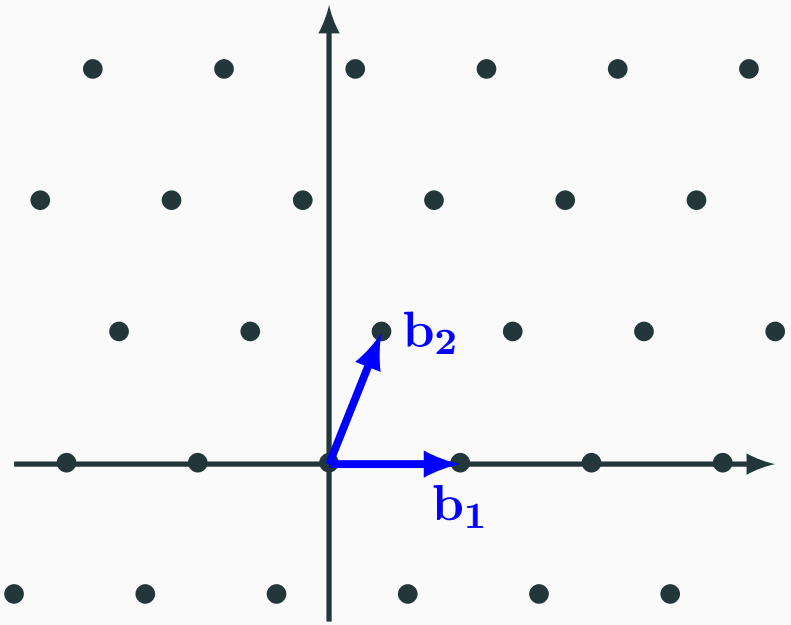
chiffrer un message m = noyer m dans le bruit (aléatoire)

de telle sorte que

déchiffrer = enlever le bruit pour retrouver m

soit facile si on a la clé secrète, et infaisable sinon.

Un exemple à base de réseaux euclidiens

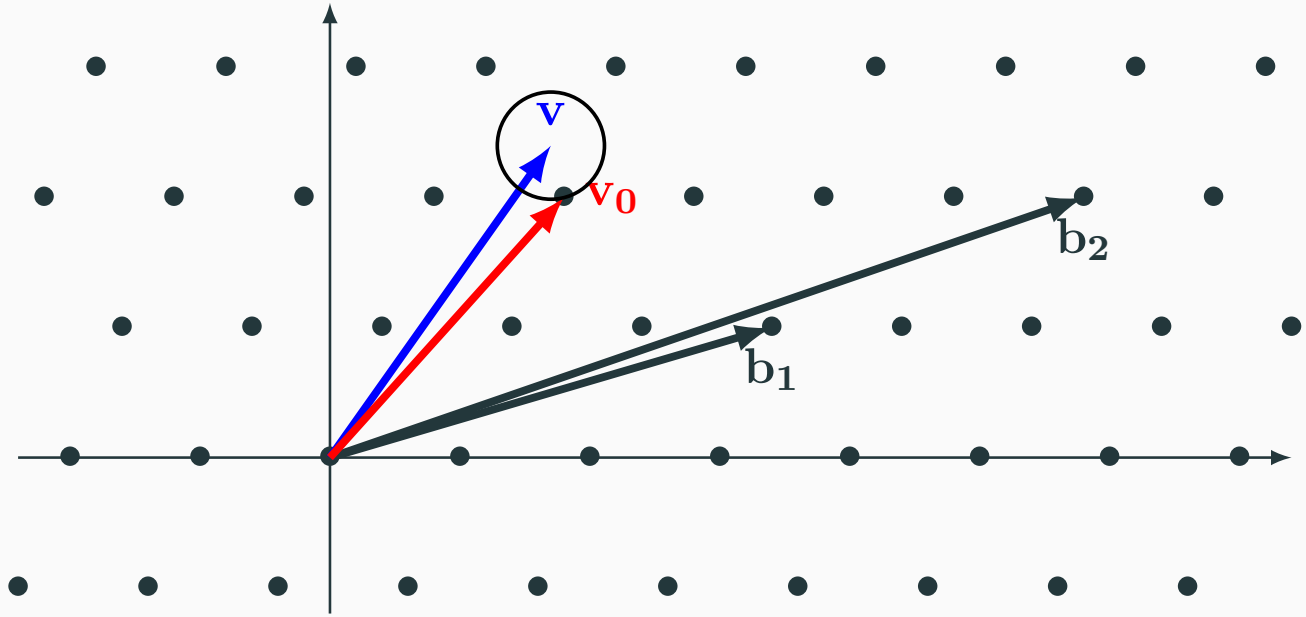


Un réseau euclidien (*lattice*) = l'ensemble des vecteurs à coordonnées entières dans une base $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ donnée.

$$\left\{ \sum_{i=1}^{n} p_i \, \mathbf{b}_i \;\middle|\; p_i \in \mathbb{Z} \right\}$$

Le problème du vecteur le plus proche

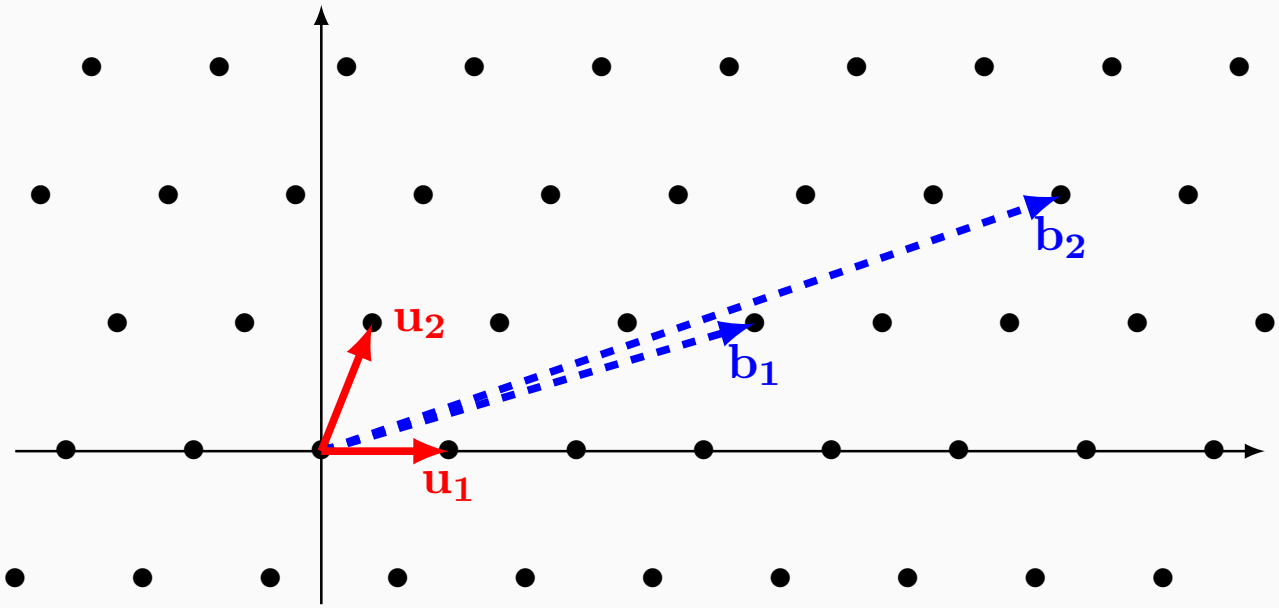
(CVP, *Closest Vector Problem*.)



Étant donné un vecteur **v**, trouver les coordonnées d'un vecteur $\mathbf{v}_0$ appartenant au réseau et le plus proche de **v**.

Problème difficile, même dans le cas moyen, même avec des solutions approchées, même avec un ordinateur quantique.

Bonnes bases



On peut changer de base sans changer l'ensemble des vecteurs du réseau euclidien : $B \mapsto U.B$ où la matrice U est unimodulaire.

Le problème CVP est facile si on a une bonne base, dont les vecteurs sont courts et presque orthogonaux.

Chiffrer avec du bruit (Goldreich–Goldwasser–Halevi)

Clé secrète : une bonne base B_0, une matrice unimodulaire U.

Clé publique : la mauvaise base $B = U.B_0$.

Chiffrer un message $(m_1, \ldots, m_n)$ (un n-uplet d'entiers) :

$$\mathcal{E}(m_1, \ldots, m_n) = \sum_{i=1}^{n} m_i \, \mathbf{b}_i + \mathbf{e}$$

où **e** est une petite erreur tirée au hasard.

Déchiffrer **c** :

Trouver le vecteur du réseau le plus proche de c

(avec U, passer en base B_0, résoudre CVP, repasser en base B)

Il est de la forme $\sum_{i=1}^{n} m_i \, \mathbf{b}_i$ et $(m_1, \ldots, m_n)$ est le texte clair.

Un chiffrement simple à base d'entiers

(Craig Gentry, *Computing arbitrary functions of encrypted data*, 2010.)

Clé : un entier p impair de P bits.

Chiffrement d'un bit $m \in \{0, 1\}$:

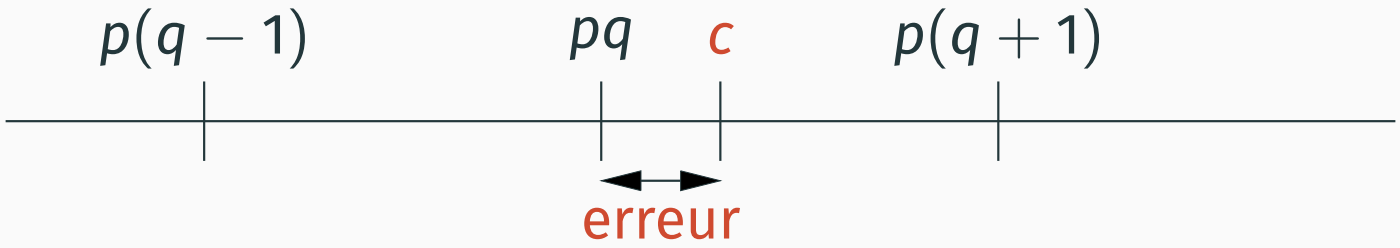
$$\mathcal{E}_p(m) = pq + 2r + m$$

q entier aléatoire de Q bits
$r \ll p$ entier aléatoire de N bits

C.à.d. un multiple de p plus une erreur $2r + m$, le message étant le bit de poids faible de l'erreur.

(Paramètres typiques : $N = \lambda$, $P = \lambda^2$, $Q = \lambda^5$.)

Un chiffrement simple à base d'entiers



Déchiffrement :

$$\mathcal{D}_p(c) = (c \bmod p) \bmod 2$$

Intuition : le multiple de p immédiatement inférieur est $p\lfloor c/p \rfloor$.
L'erreur $2r + m$ est $c - p\lfloor c/p \rfloor = c \bmod p$.
Le message m est $(2r + m) \bmod 2$.

L'attaquant ne peut pas retrouver p à partir de chiffrés $c_1, \ldots, c_n$
(le problème du PGCD approché est difficile).

Un chiffrement simple à base d'entiers

Pour en faire un chiffrement à clé publique, on peut garder p comme clé secrète et diffuser comme clé publique un ensemble Z de chiffrements aléatoires de zéro :

$$pk = \{2r_i + pq_i \mid r_i \text{ aléatoire } N \text{ bits}, q_i \text{ aléatoire } Q \text{ bits}\}$$

Chiffrement d'un bit $m \in \{0, 1\}$:

$$\mathcal{E}_{pk}(m) = m + \sum_{z \in Z} z$$

où Z est un sous-ensemble aléatoire de pk de taille raisonnable.

Un chiffrement presque homomorphe

$$\begin{aligned}\mathcal{E}_p(m_1) + \mathcal{E}_p(m_2) &= (m_1 + 2r_1 + pq_1) + (m_2 + 2r_2 + pq_2) \\ &= m_1 \oplus m_2 + 2(m_1 m_2 + r_1 + r_2) + p(q_1 + q_2)\end{aligned}$$

Se déchiffre en $m_1 \oplus m_2$ tant que le bruit $2(m_1 m_2 + r_1 + r_2)$ reste inférieur à p.

$$\begin{aligned}\mathcal{E}_p(m_1) \cdot \mathcal{E}_p(m_2) &= (m_1 + 2r_1 + pq_1)\,(m_2 + 2r_2 + pq_2) \\ &= m_1 m_2 + 2(r_1 m_2 + r_2 m_1 + 2r_1 r_2) + p(\ldots)\end{aligned}$$

Se déchiffre en $m_1 m_2$ tant que le bruit $2(r_1 m_2 + r_2 m_1 + 2r_1 r_2)$ reste inférieur à p.

Un chiffrement presque homomorphe

(SHE, *Somewhat Homomorphic Encryption*.)

Le bruit augmente

- lentement (+ 1 bit) à chaque addition ;
- rapidement ($\times$ 2 bits) à chaque multiplication.

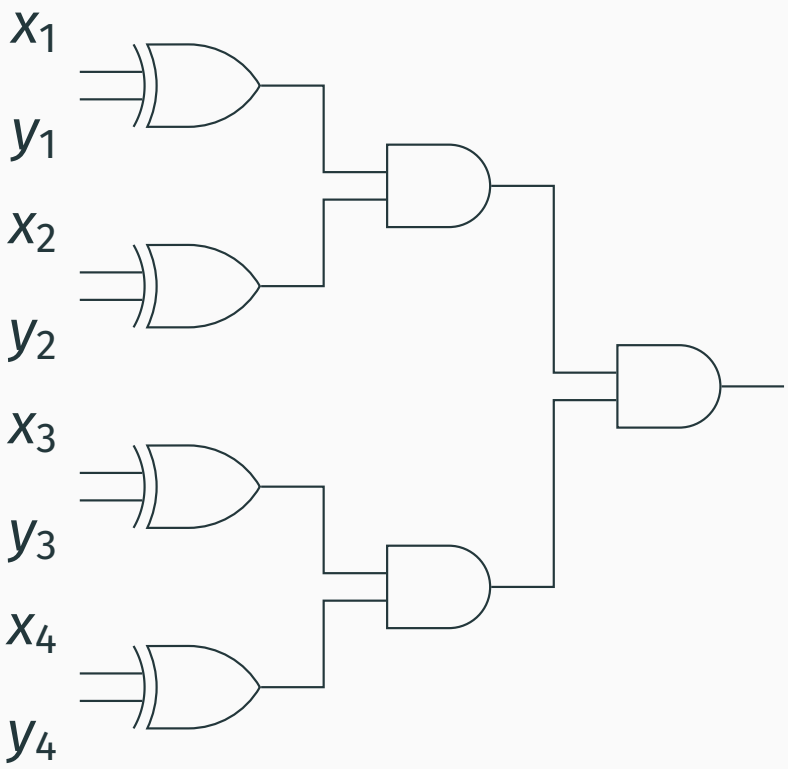
Lorsque le bruit dépasse p, les résultats chiffrés deviennent faux.

Cela limite grandement la profondeur multiplicative (et faiblement la profondeur additive) des calculs que l'on peut faire de manière homomorphe.

Des circuits limités

La profondeur multiplicative d'un circuit est le nombre maximal de portes «et» / «ou» entre une entrée et une sortie.

Exemple : un comparateur n bits est de profondeur multiplicative $\lceil \log_2 n \rceil$.



De SHE à FHE : comment réduire le bruit ?

Pour évaluer des circuits arbitrairement profonds de manière homomorphe, il faut réduire le bruit de certains résultats intermédiaires chiffrés afin que le bruit ne dépasse jamais P bits.

Idée naïve : on déchiffre puis on re-chiffre !

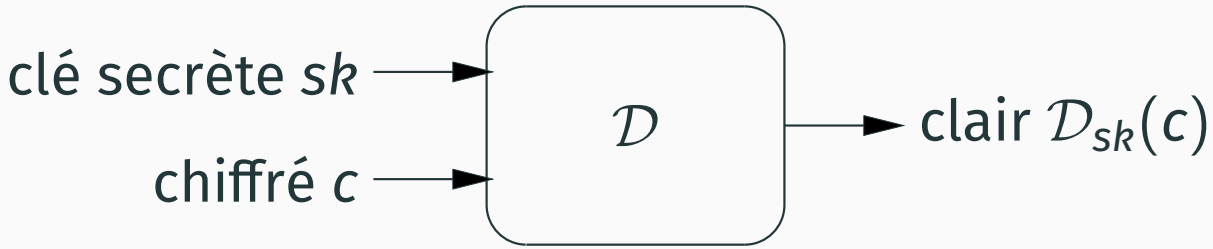
$$c \mapsto \mathcal{E}_{pk}(\mathcal{D}_{sk}(c))$$

Le bruit, qui approchait P bits, retombe à N bits.

Problèmes : le résultat intermédiaire $\mathcal{D}_{sk}(c)$ serait en clair ; on n'a pas accès à la clé secrète sk (l'entier p).

De SHE à FHE : le *bootstrap* de Gentry

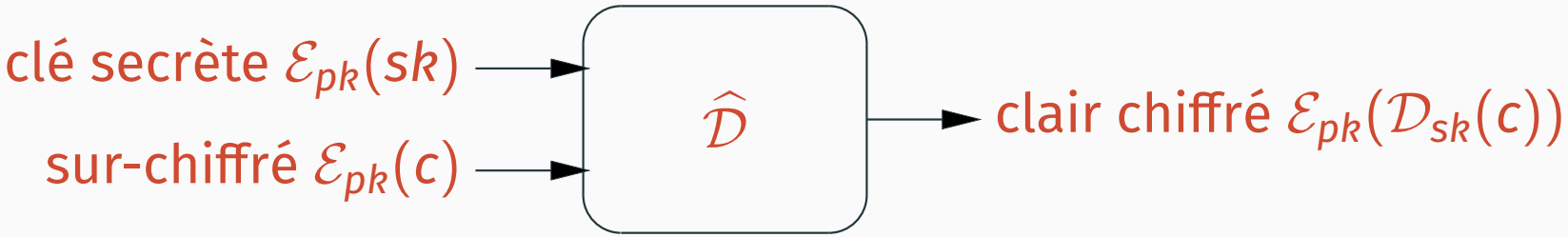
(Craig Gentry, *A fully homomorphic encryption scheme*, PhD, Stanford, 2009.)



L'algorithme de déchiffrement $\mathcal{D}$ peut être réalisé par un circuit.

De SHE à FHE : le *bootstrap* de Gentry

(Craig Gentry, *A fully homomorphic encryption scheme*, PhD, Stanford, 2009.)

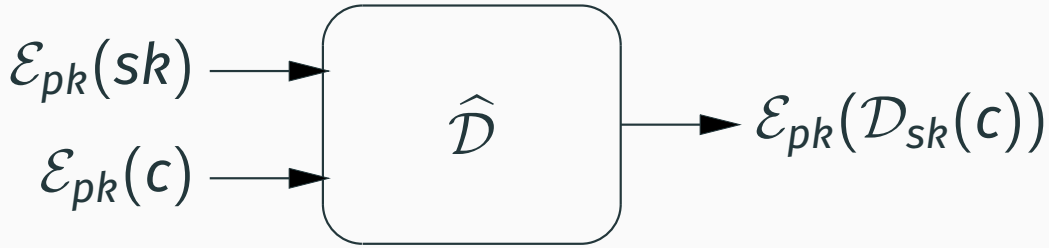


L'algorithme de déchiffrement $\mathcal{D}$ peut être réalisé par un circuit.

Si sa profondeur multiplicative est suffisamment faible, ce circuit peut être évalué de manière homomorphe en utilisant notre chiffrement presque homomorphe (SHE).

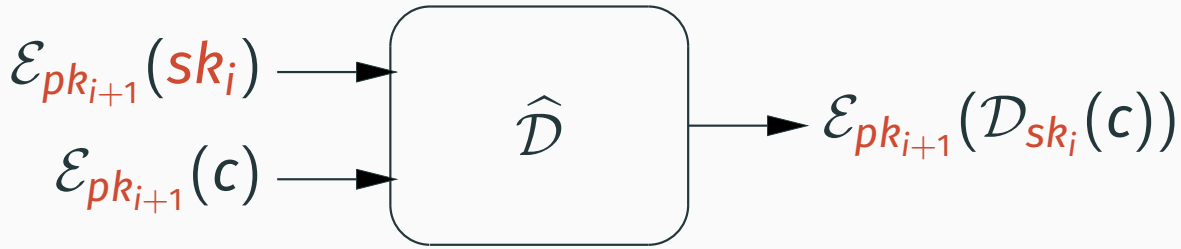
Le résultat est un chiffré équivalent à c, mais dont le bruit dépend uniquement de la profondeur multiplicative du circuit.

Les difficultés du *bootstrap*



Les cryptographes n'aiment pas l'idée de chiffrer une clé secrète sk avec sa clé publique pk.

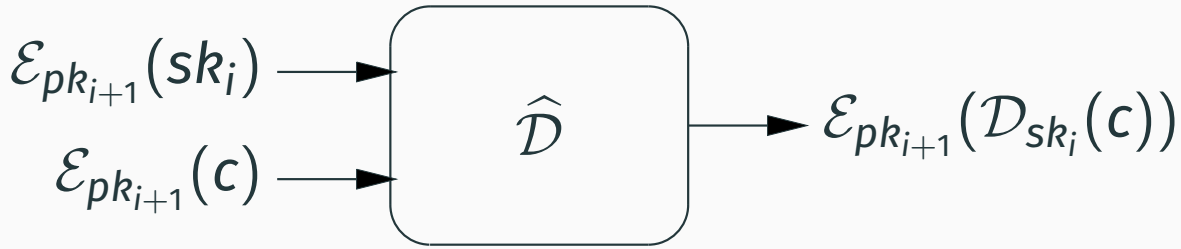
Les difficultés du *bootstrap*



Les cryptographes n'aiment pas l'idée de chiffrer
une clé secrète sk avec sa clé publique pk.
$\rightarrow$ On change de clés au moment du bootstrap.

Pour pouvoir itérer le bootstrap aussi souvent que nécessaire, le calcul homomorphe reçoit une suite de clés publiques pk_i
et les clés secrètes correspondantes (chiffrées) $\mathcal{E}_{pk_{i+1}}(sk_i)$.

Les difficultés du *bootstrap*



Le sur-chiffrement $\mathcal{E}_{pk_{i+1}}(c)$ augmente fortement la taille du résultat intermédiaire chiffré c.

Le circuit de déchiffrement est souvent trop ≪profond≫

→ Préférer les SHE où le déchiffrement est simple
(p.ex. à base de réseaux euclidiens).

→ Modifier le chiffrement pour qu'il laisse des indications (*hints*) qui simplifient le déchiffrement.

Bilan sur le chiffrement homomorphe

Beaucoup de travaux depuis le résultat fracassant de Gentry :

- autres chiffrements presque homomorphes ;
- multiplications qui augmentent moins le bruit ;
- bootstrap plus efficace.

Des implémentations avec des performances presque raisonnables, p.ex. TFHE (`https://github.com/tfhe/tfhe`) :

- évaluation d'une porte logique $\approx$ 20 ms
- bootstrap $\approx$ 100 ms.

Une nouvelle direction : le chiffrement homomorphe approché, pour l'apprentissage statistique sur des données confidentielles.

Calcul multipartite sécurisé

Le problème des millionaires (A. Yao, 1982)

Alice et Bob veulent savoir qui est le plus riche, sans révéler leur patrimoine à l'autre.

Avec un tiers de confiance (Charlie) :
Alice communique le montant de son patrimoine à Charlie.
Bob fait de même.
Charlie annonce qui est le plus riche, et ne révèle rien d'autre.

Sans tiers de confiance : quel algorithme distribué, exécuté par Alice et par Bob, donnerait le même résultat et les mêmes garanties ?

Calcul multipartite sécurisé

n participants, ayant chacun une donnée secrète x_i,
coopèrent pour calculer une fonction $y = F(x_1, \ldots, x_n)$
sans rien révéler sur les x_i qui ne soit impliqué par le résultat y.

Exemple : un appel d'offres

$$F(x_1, \ldots, x_n) = (i, x_i) \quad \text{avec } x_i = \min(x_1, \ldots, x_n)$$

Révèle l'identité du mieux disant et le montant de son offre, mais pas les montants des autres offres.

Autres exemples : des statistiques sur les x_i
(la moyenne, la médiane, un histogramme par déciles, etc.)

Utiliser un chiffrement homomorphe

Une clé secrète sk découpée en n parts $(sk_1, \ldots, sk_n)$
+ la clé publique pk correspondante.

1. Chaque participant chiffre sa donnée et la publie : $\mathcal{E}(x_i)$.
2. Quelqu'un calcule F homomorphiquement à partir des $\mathcal{E}(x_i)$.
3. Les participants collaborent pour déchiffrer le résultat.

C'est une solution correcte, mais il est beaucoup plus efficace de distribuer le calcul entre les participants.

Partager un bit

Comment partager un bit b secret entre deux participants ?

- Tirer un bit r au hasard.
- Envoyer $b_1 = r$ à un participant et $b_2 = b \oplus r$ à l'autre.

Aucun participant ne peut retrouver b à lui tout seul.

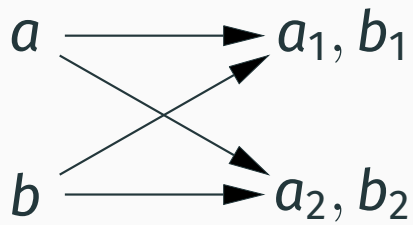
Si les deux participants publient leurs bits b_1 et b_2, ils retrouvent b en calculant $b_1 \oplus b_2 = r \oplus b \oplus r = b$.

On note $[b]$ un partage d'un bit b : une paire de bits (b_1, b_2) tels que $b = b_1 \oplus b_2$.

Mettre en commun des bits privés

Le partage de bit permet également à deux participants A, B de mettre en commun deux bits privés, a fourni par A et b par B :

- A tire un partage $[a] = (a_1, a_2)$ et envoie a_2 à B.
- B tire un partage $[b] = (b_1, b_2)$ et envoie b_1 à A.



Additionner deux bits partagés

On a deux bits partagés, $[x] = (x_1, x_2)$ et $[y] = (y_1, y_2)$.

Le participant 1 connait x_1 et y_1, et calcule $z_1 \stackrel{def}{=} x_1 \oplus y_1$.

Le participant 2 connait x_2 et y_2, et calcule $z_2 \stackrel{def}{=} x_2 \oplus y_2$.

La paire (z_1, z_2) est bien un partage de $x \oplus y$:

$$z_1 \oplus z_2 = (x_1 \oplus y_1) \oplus (x_2 \oplus y_2) = (x_1 \oplus x_2) \oplus (y_1 \oplus y_2) = x \oplus y$$

Le calcul est local (pas de communication entre participants).

Multiplier deux bits partagés

On a deux bits partagés, $[x] = (x_1, x_2)$ et $[y] = (y_1, y_2)$ et on souhaite calculer un partage (z_1, z_2) de $x \wedge y$.

Aucun calcul purement local ne convient. Notamment,

$$(x_1 \wedge y_1) \oplus (x_2 \wedge y_2) \neq (x_1 \oplus x_2) \wedge (y_1 \oplus y_2)$$

Une solution coûteuse à base de transfert inconscient 1 sur 4 (*1-in-4 oblivious transfer*).

Multiplication par transfert inconscient

P1 choisit z_1 au hasard et tabule la bonne valeur de z_2 ($z_2 = z_1 \oplus ((x_1 \oplus x_2) \wedge (y_1 \oplus y_2))$) suivant les inconnues x_2 et y_2 :

ligne	x_2	y_2	z_2
0	0	0	$z_1 \oplus (x_1 \wedge y_1)$
1	0	1	$z_1 \oplus (x_1 \wedge \neg y_1)$
2	1	0	$z_1 \oplus (\neg x_1 \wedge y_1)$
3	1	1	$z_1 \oplus (\neg x_1 \wedge \neg y_1)$

P2 choisit la ligne (0 à 3) correspondant à ses valeurs de x_2 et y_2 et reçoit le z_2 correspondant.

P1 ne sait pas quelle ligne P2 a choisie.

P2 n'apprend rien sur les autres lignes.

Multiplication avec triplets de Beaver

On peut préparer à l'avance des triplets multiplicatifs aussi appelés triplets de Beaver :
des bits partagés $[a], [b], [c]$ tels que $c = a \wedge b$.

(Par transfert inconscient, ou autre protocole *zero knowledge*, ou via un tiers de confiance.)

P1 a les parties (a_1, b_1, c_1) des triplets et P2 les parties (a_2, b_2, c_2).

Multiplication avec triplets de Beaver

Pour calculer un partage (z_1, z_2) de $x \wedge y$:

P1 et P2 prennent le prochain triplet de Beaver (a, b, c) sur la liste.

P1 publie $a_1 \oplus x_1$ et $b_1 \oplus y_1$. (i.e. ses x, y masqués par a, b)

P2 publie de même $a_2 \oplus x_2$ et $b_2 \oplus y_2$.

P1 et P2 connaissent maintenant $d = a \oplus x$ et $e = b \oplus y$.

P1 calcule z_1 et P2 calcule z_2 comme suit :

$$z_i = d \wedge e \oplus d \wedge b_i \oplus a_i \wedge e \oplus c_i$$

C'est un partage de $x \wedge y$, car

$$x \wedge y = (d \oplus a) \wedge (e \oplus b) = d \wedge e \oplus d \wedge b \oplus a \wedge e \oplus \underbrace{a \wedge b}_{=c}$$

Extension à $n > 2$ participants

On peut partager un bit b entre $n > 2$ participants :

$$[b] = (b_1, \ldots, b_n) \quad \text{avec} \quad b = b_1 \oplus \cdots b_n$$

Avec $b_1, \ldots, b_{n-1}$ choisis au hasard, aucun participant n'a d'information sur b.

Il faut que les n participants partagent leurs information pour retrouver b.

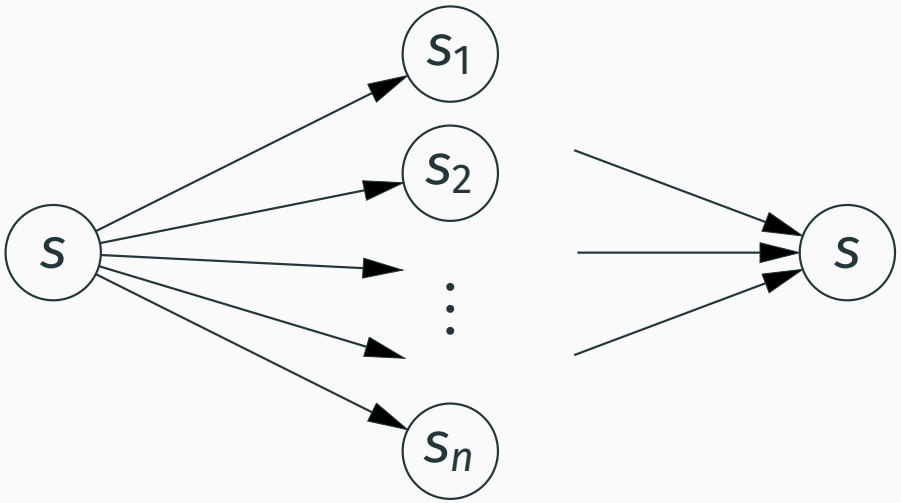
Une collusion de $t < n$ participants ne peut pas retrouver b.

Problème : il suffit d'un participant qui tombe en panne ou produit un résultat faux pour que le calcul multipartite échoue ou produise un résultat faux.

Partage de secret : le cas général

Partager un secret s en n parts $s_1, \ldots, s_n$ de sorte que

- t parts suffisent pour retrouver s ;
- moins de t parts ne révèlent rien sur s.



Distribution de n parts

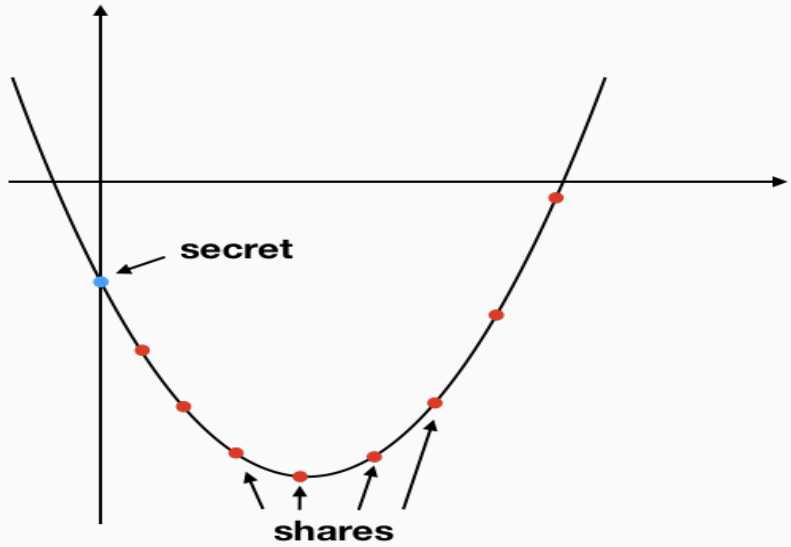
Reconstruction depuis t parts

Le partage de secret de Shamir

Le secret s est un élément d'un corps fini, p.ex. $\mathbb{Z}/p\mathbb{Z}$.

Partager le secret :

- Choisir un polynôme P de degré $t-1$ avec le coefficient constant égal à s et les autres coefficients aléatoires.
- Les parts sont $s_i = P(i)$ pour $i = 1, \ldots, n$.



Le partage de secret de Shamir

Le secret s est un élément d'un corps fini, p.ex. $\mathbb{Z}/p\mathbb{Z}$.

Retrouver le secret :

Connaître t parts, c'est connaître t points $(x_1, y_1), \ldots, (x_t, y_t)$ sur la courbe de P.

P étant de degré $t-1$, ces t points déterminent P entièrement.

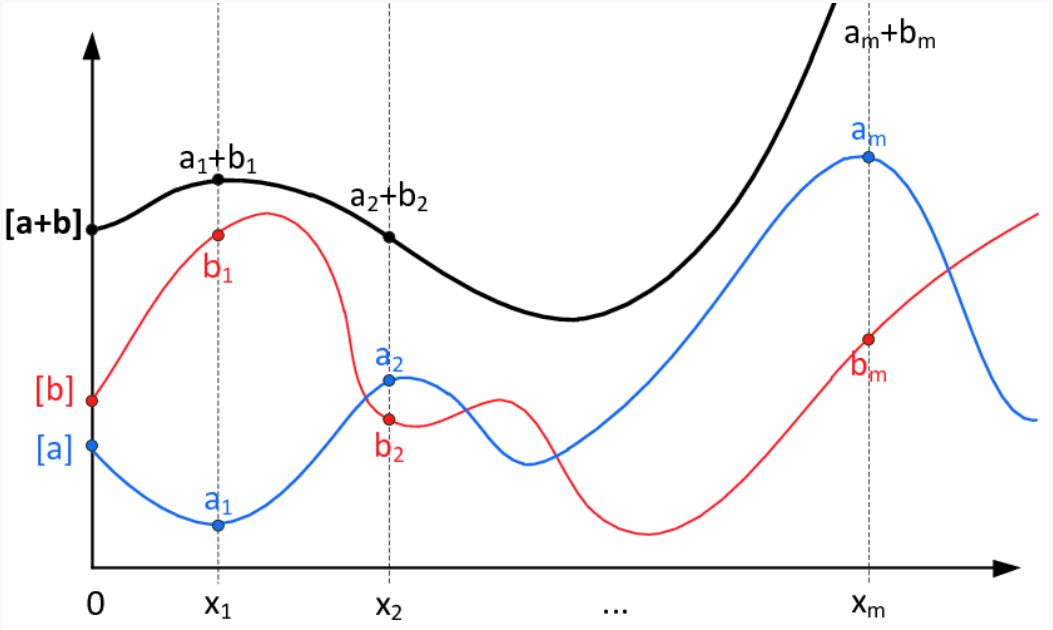
Le secret s est $P(0)$.

Plus directement, par la formule d'interpolation de Lagrange :

$$s = P(0) = \sum_{j=1}^{t} y_j \prod_{k=1, k\neq j}^{t} \frac{x_k}{x_k - x_j}$$

(Note : si plus de t parts sont données, on peut non seulement retrouver s mais aussi vérifier la cohérence des parts entre elles.)

Calculer avec des partages de Shamir : l'addition



Soient $[a] = (a_1, \ldots, a_n)$ et $[b] = (b_1, \ldots, b_n)$ des partages de Shamir pour les secrets a et b.

Alors $(a_1 + b_1, \ldots, a_n + b_n)$ est un partage de Shamir pour $a + b$.

Il peut être calculé localement par chacun des n participants.

Calculer avec des partages de Shamir : la multiplication

Soient $[a] = (a_1, \ldots, a_n)$ et $[b] = (b_1, \ldots, b_n)$ des partages de Shamir pour les secrets a et b :

$$a = A(0) \quad a_i = A(i) \quad b = B(0) \quad b_i = B(i)$$

où A et B sont des polynômes de degré $t - 1$.

Les points $(i, a_i b_i)$ sont sur la courbe du polynôme AB.

Mais AB est de degré $2t - 2$, donc $t - 1$ points ne suffisent pas à déterminer $AB(0) = ab$.

Et donc $(a_1 b_1, \ldots, a_n b_n)$ n'est pas un partage de ab.

Calculer avec des partages de Shamir : la multiplication

Chacun des $2t-1$ premiers participants prépare un partage de son coefficient $a_i b_i$, c.à.d. un polynôme aléatoire P_i tel que $P_i(0) = a_i b_i$.

Ils publient ces partages :
le participant i envoie $P_i(j)$ au participant j.

Les n participants reconstruisent alors un partage $(c_1, \ldots, c_n)$ par la formule d'interpolation de Lagrange :

$$c_j = \sum_{i=1}^{2t-1} P_i(j)\lambda_i \quad \text{avec} \quad \lambda_i = \prod_{k=1, k\neq i}^{2t-1} \frac{k}{k-i}$$

C'est un partage de ab, car $P = \sum_{i=1}^{2t-1} P_i\lambda_i$ est un polynôme de degré $t-1$ qui vaut ab en 0 : $P(0) = \sum_{i=1}^{2t-1} a_i b_i \lambda_i = AB(0) = ab$.

Point d’étape

Calculer sur des données chiffrées ou masquées

Trois approches qui commencent à être bien comprises :

	Calcul mutipartite	Chiffrement homomorphe	Chiffrement fonctionnel
Entrées	masquées	chiffrées	chiffrées
Sorties	en clair	chiffrées	en clair
Communications	oui	non	non
Efficacité	bonne	faible	bonne dans cas particuliers

D'autres approches restent encore théoriques, comme l'obfuscation indistinguable.

Protéger les données pendant le calcul

L'approche cryptographique :

- sécurité élevée et que l'on peut étudier mathématiquement ;
- exécution coûteuse ;
- expressivité restreinte des calculs
  (circuits uniquement, ni conditionnelles ni boucles).

Déjà utilisable pour des calculs simples mais hautement confidentiels : dépouillement d'élections, d'enchères secrètes, …

Sécurité du logiciel

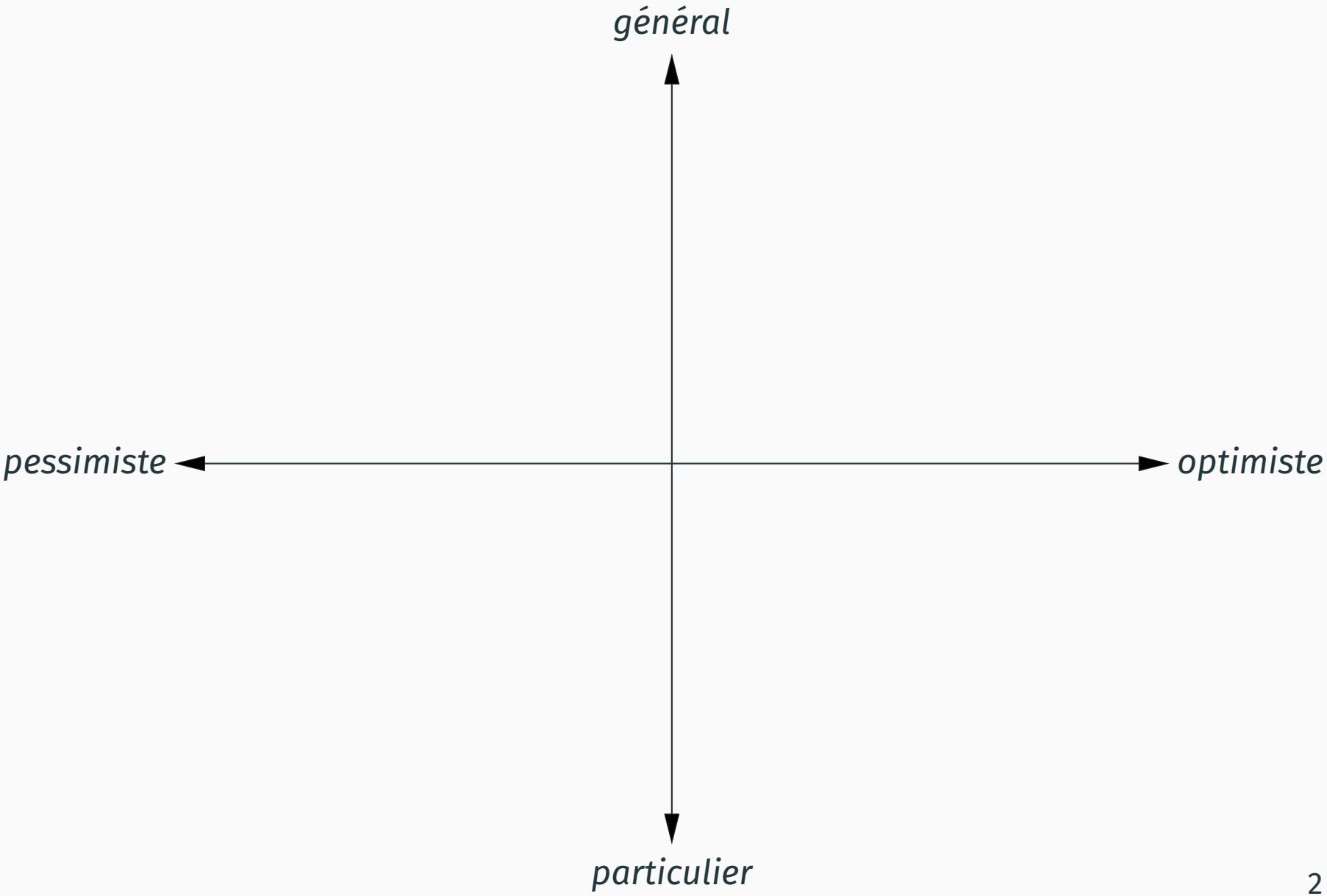
Bilan du cours

Xavier Leroy

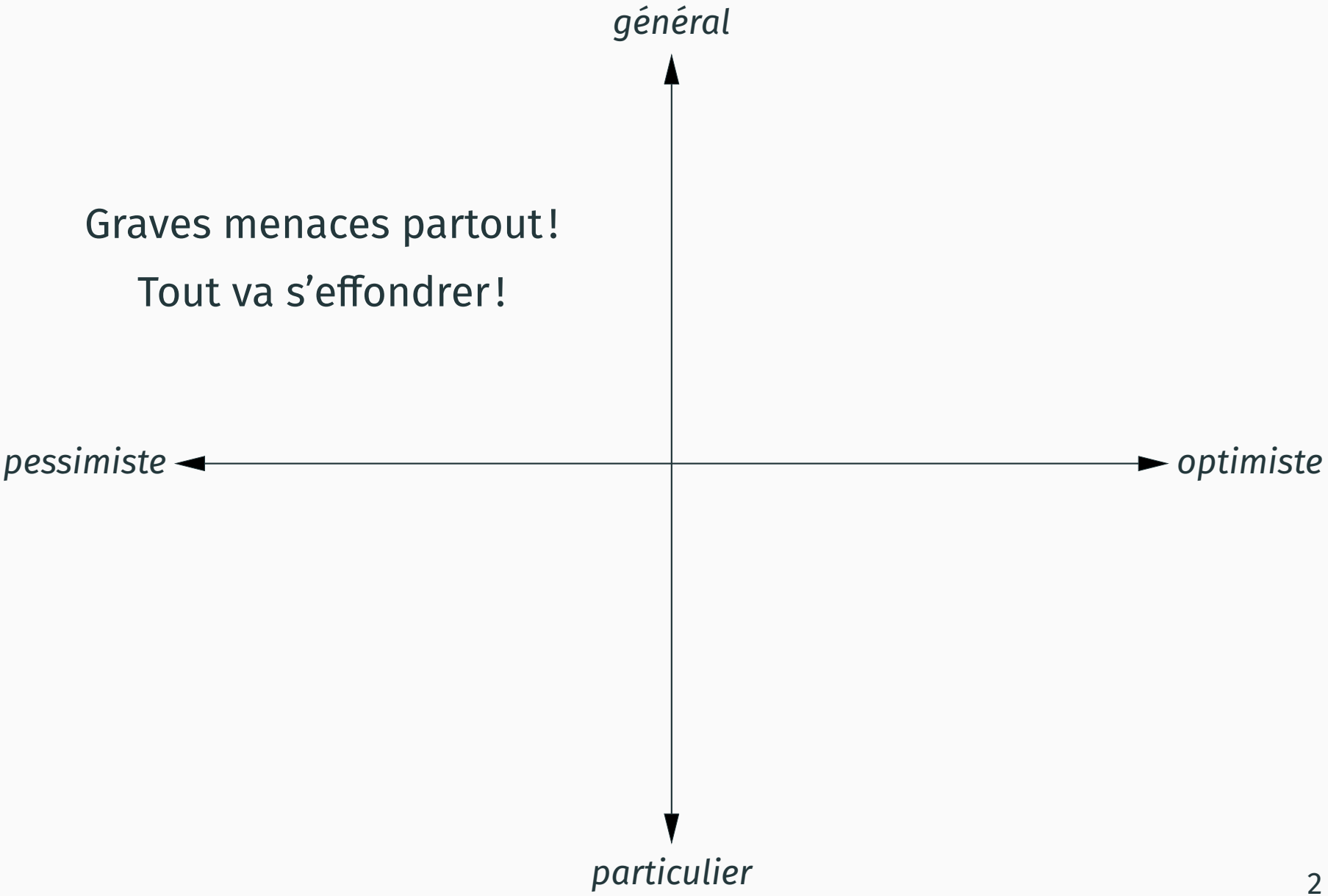
2022-04-21

Collège de France, chaire de sciences du logiciel
xavier.leroy@college-de-france.fr

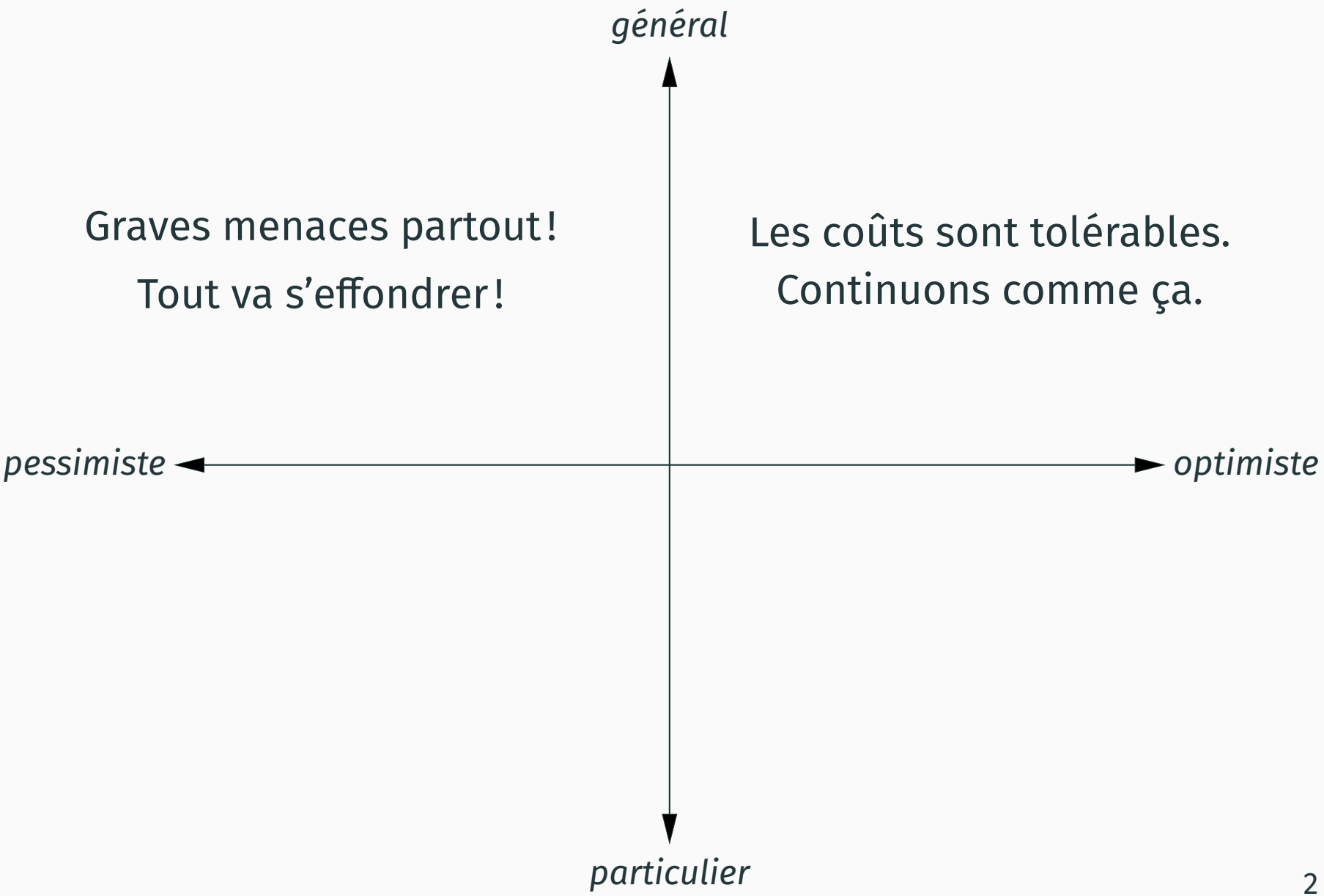
Quatre discours sur la sécurité informatique



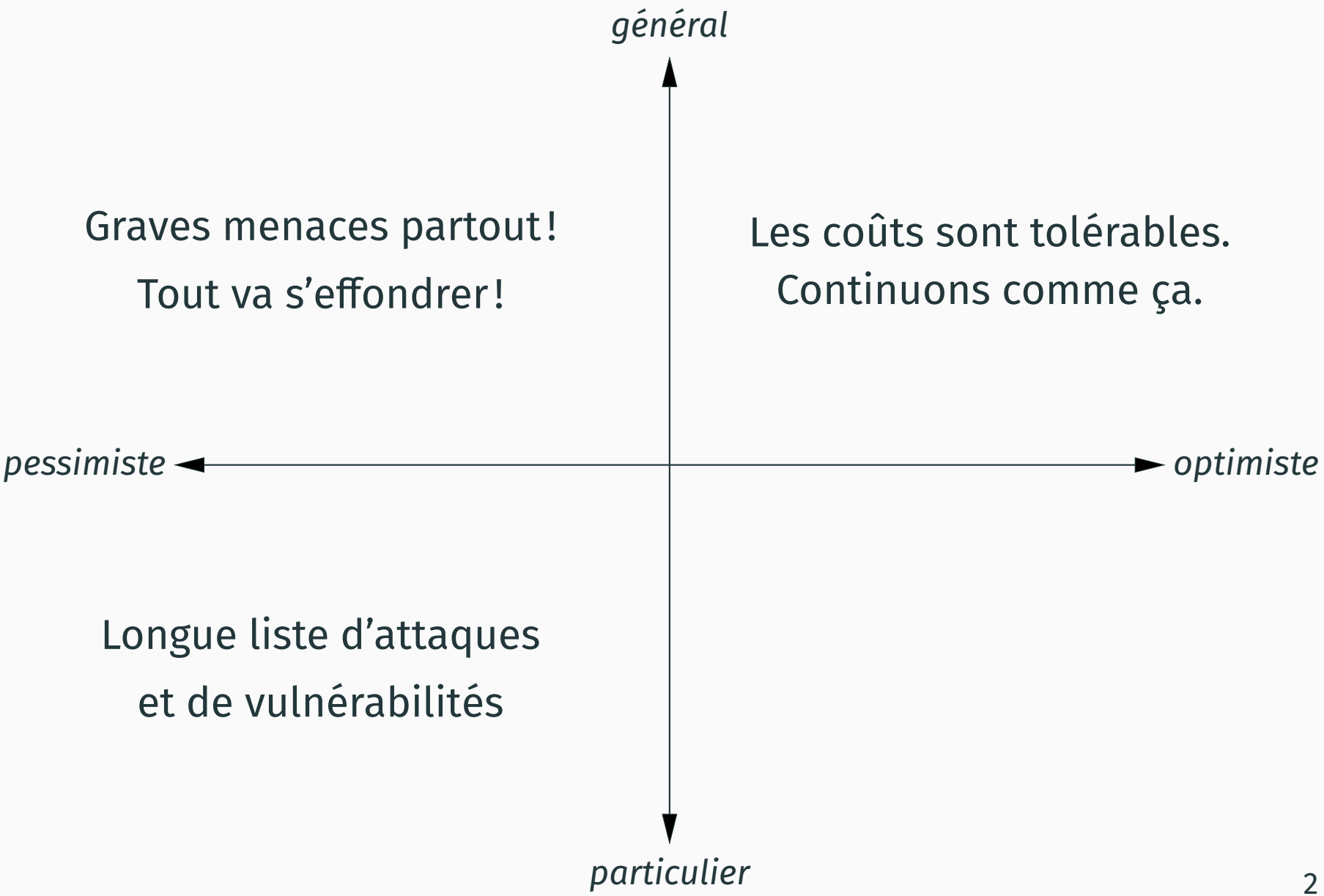
Quatre discours sur la sécurité informatique



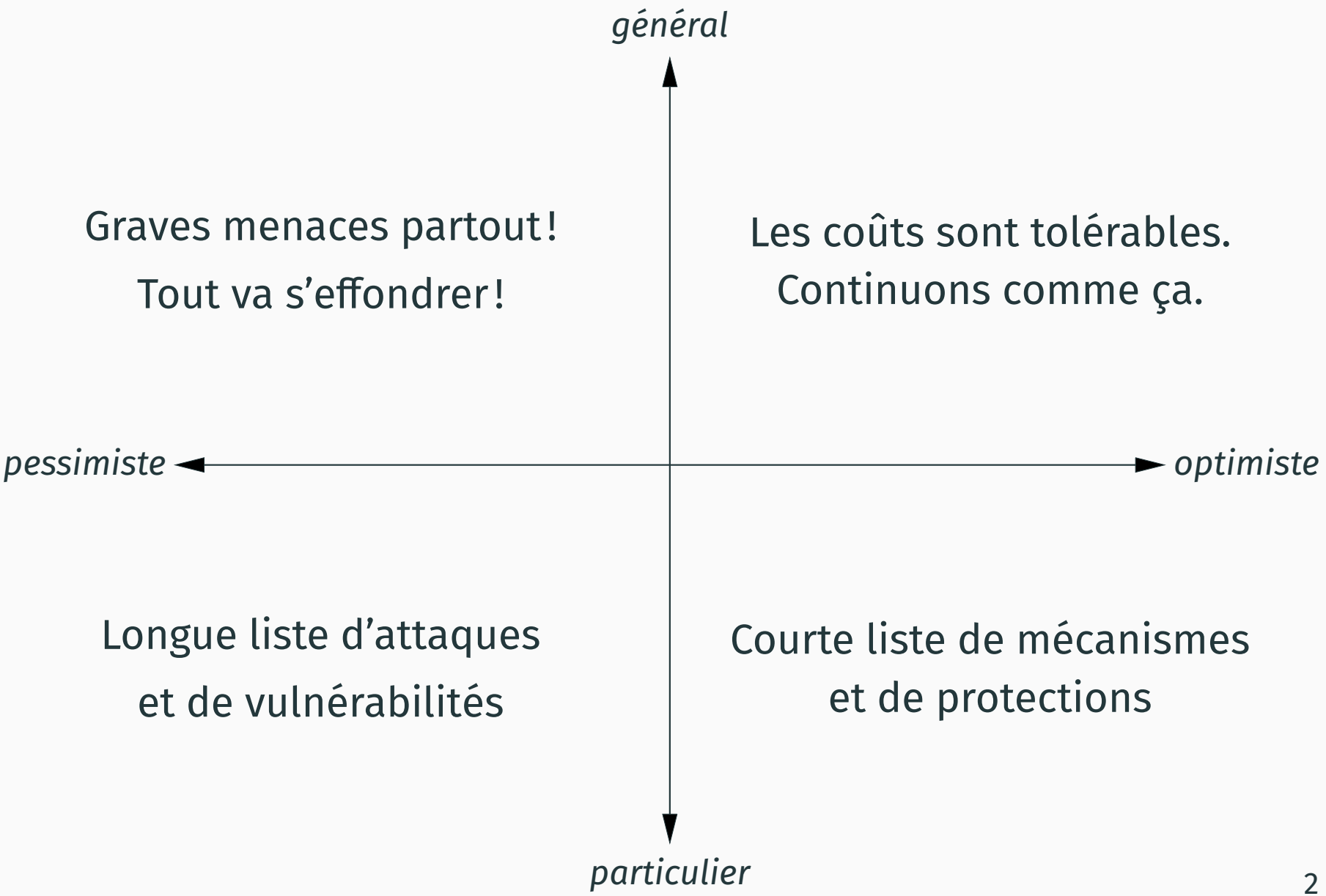
Quatre discours sur la sécurité informatique



Quatre discours sur la sécurité informatique



Quatre discours sur la sécurité informatique



Les composantes de la sécurité informatique

Utilisateurs

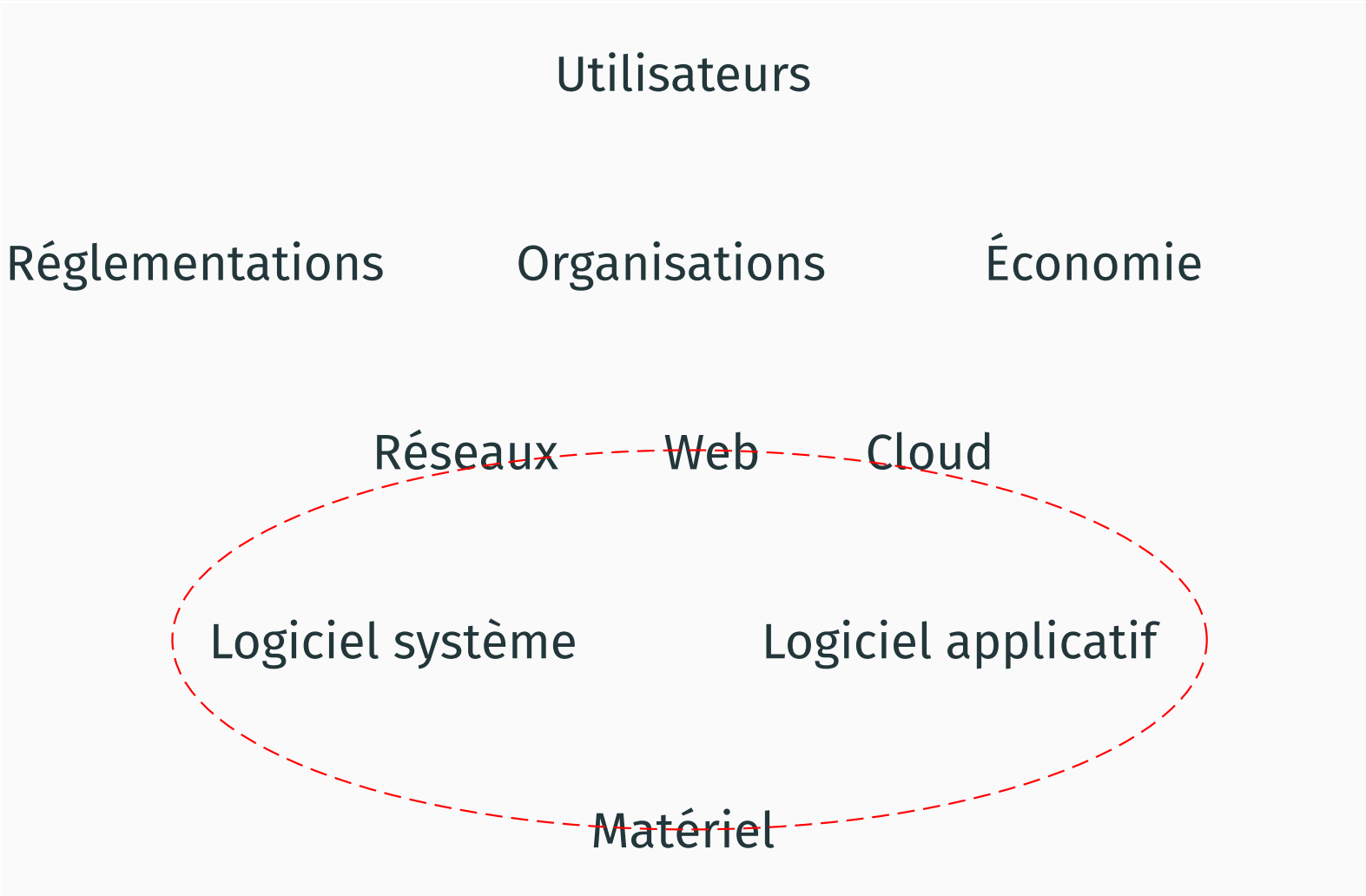
Réglementations Organisations Économie

Réseaux Web Cloud

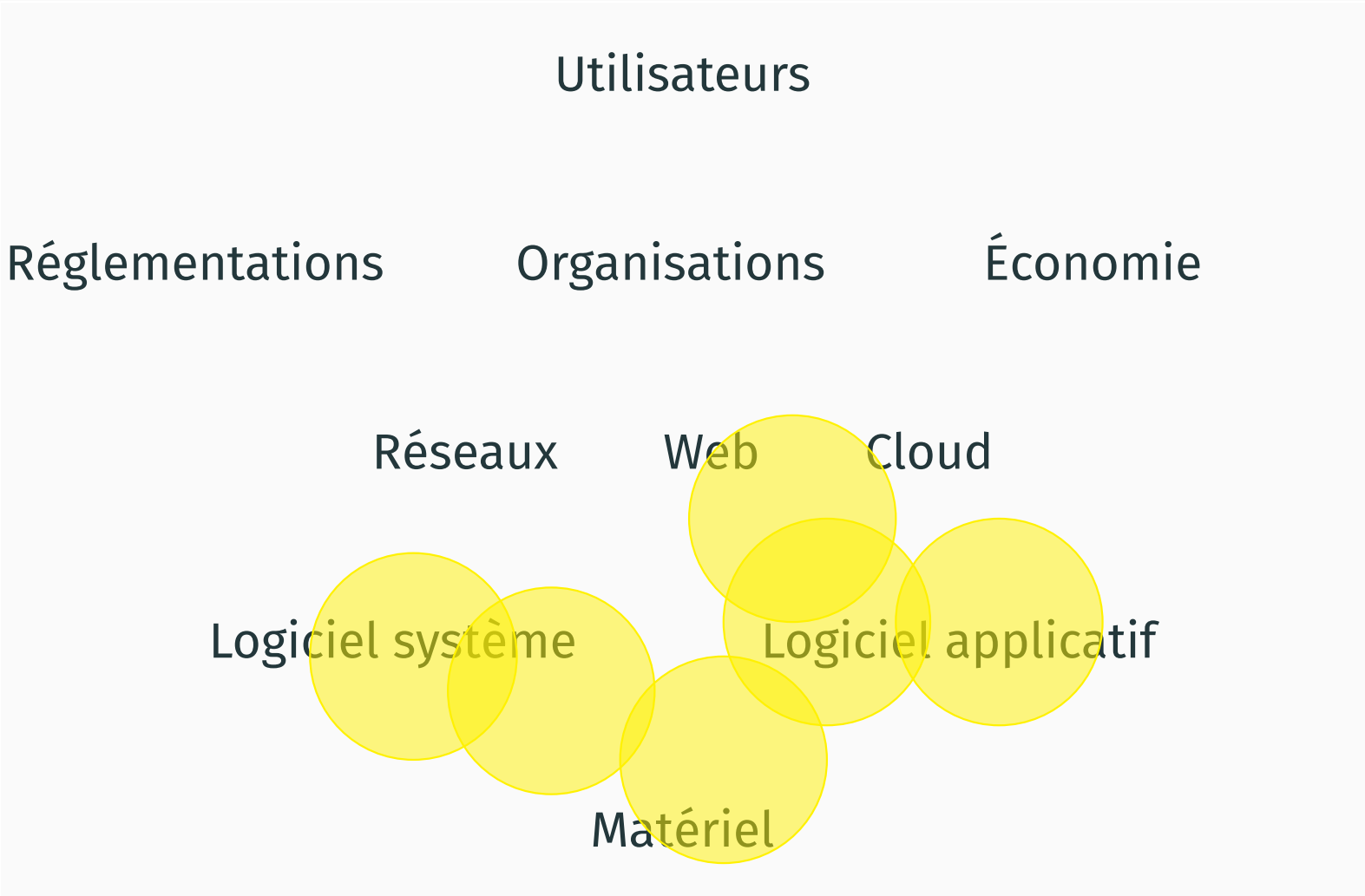
Logiciel système Logiciel applicatif

Matériel

Les composantes de la sécurité informatique



Les composantes de la sécurité informatique



Quel rôle pour les langages et outils de programmation ?

Un rôle essentiel :

- La sûreté de l'exécution : donner des garanties de base sur l'intégrité des structures de données et des flux de contrôle.

D'autres contributions ponctuelles à la sécurité, p.ex. :

- Contrôle des flux d'information (cours 2)
- Isolation logicielle des fautes (cours 3)
- Programmation ≪en temps constant≫ (cours 4)
- Vérification du code mobile ; code auto-certifiant (cours 5)
- Protections contre des attaques bas-niveau (cours 6)
- Etc.

La programmation au défi de la sécurité

De mauvaises pratiques de programmation qui n'aident pas à sécuriser les logiciels :

- la performance ne passe pas avant tout ;
- le test ne suffit pas.

Programming Satan's computer
(Ross Anderson)

Difficile de raisonner formellement au-delà de la correction des exécutions !

- confidentialité, *privacy* ;
- disponibilité, résilience ;
- fautes et fuites dans le matériel.

FIN