A retenir…

- Algorithme publique $\Rightarrow$ notion de clés
  - espace des clés suffisamment grand pour échapper force brute
- Structure PKI pour se garantir identité/pk
- Le paramètre de sécurité $\lambda \Rightarrow$ taille des clés
- **Se prémunir des risques d'intrusion**
- Différentes notions de sécurité : dépendant des pouvoirs de l'attaquant
- Sécurité repose sur un problème que l'on ne sait pas résoudre rapidement
  - $\Rightarrow$ pas de sécurité absolue tant que l'on ne saura pas si P≠NP
  - Dépend des paradigmes de calculs, i.e. informatique quantique
- Perspectives : propriétés homomorphes, signatures, calcul-multi-parties

Cours de cryptographie – sécurité

Mif 29

Contact :

gavin@univ-lyon1.fr

romuald.thion@univ-lyon1.fr

Attentes de la cryptographie

- **Transmettre de l'information de manière sécurisée**
  - https
  - Payer ses impôts sur internet
- **Calcul multi-parties sécurisé**
  - Vote électronique (ne consiste pas seulement à sécuriser la transmission)
  - Machine learning
  - Sécuriser le cloud

…

Travaux actuels en cryptographie

- Proposition de nouveaux outils cryptographique
  - cryptosystèmes complètement homomorphes **(FHE)**
- Preuves de sécurité + tentative d'automatisation des preuves
- Recherche de failles dans protocoles existants + remèdes
- Anticiper les révolutions
  - Algorithmiques
  - Matériels informatiques, i.e. ordinateurs quantiques…

Informatique ou mathématique ?

- La réponse est claire…la cryptographie est une branche de l'informatique.
- Un protocole sera dit sûr s'il n'existe pas d'attaque de complexité polynomiale

  ⇒ on ne pourra rien prouver de manière absolue tant que l'on ne saura pas si P≠NP

  ⇒ on se contentera de preuves relatives du type « si tel problème classique est difficile alors le protocole est sûr »

En cryptographie…

…ne pas pouvoir résoudre un problème signifie souvent :

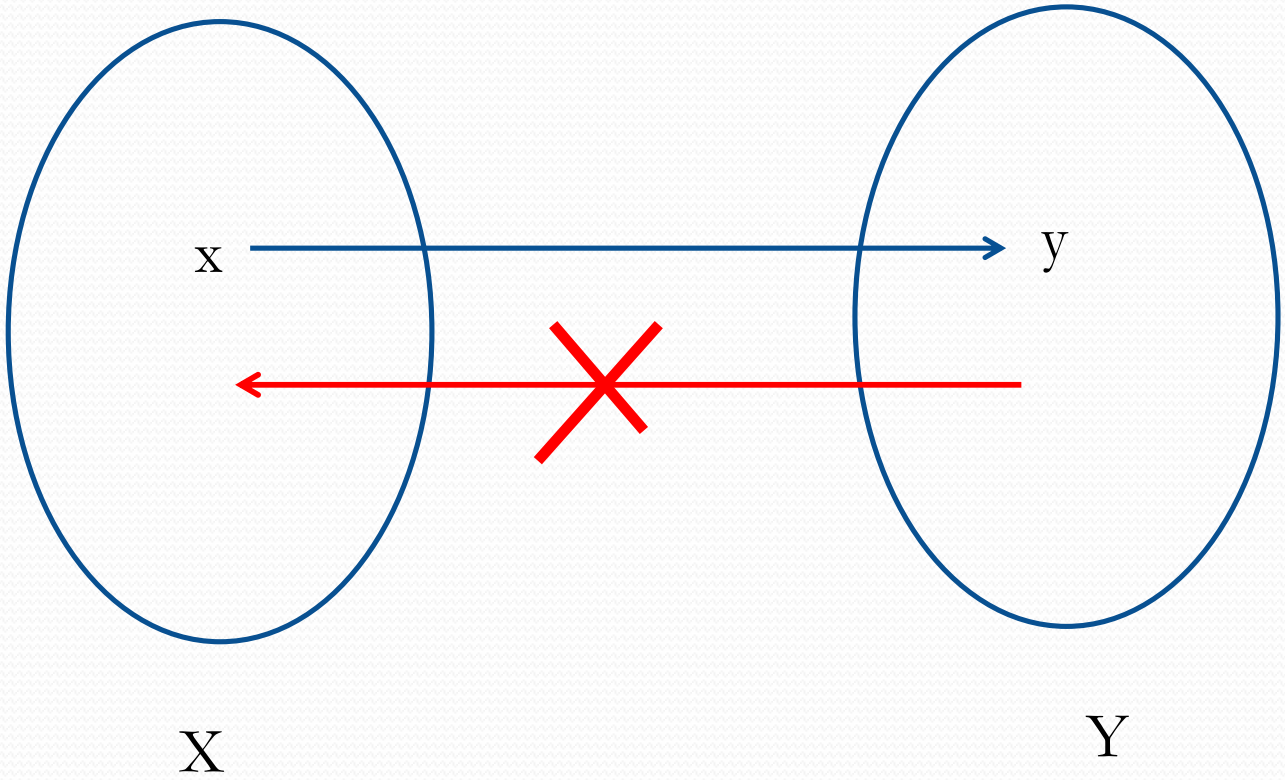
- il n'existe pas d'algorithme **polynomial/rapide** pour le résoudre
- Les paramètres sont choisis de manière à ce que la meilleure attaque connue nécessite plusieurs milliers d'années

Questions fondamentales : apport de la cryptographie moderne

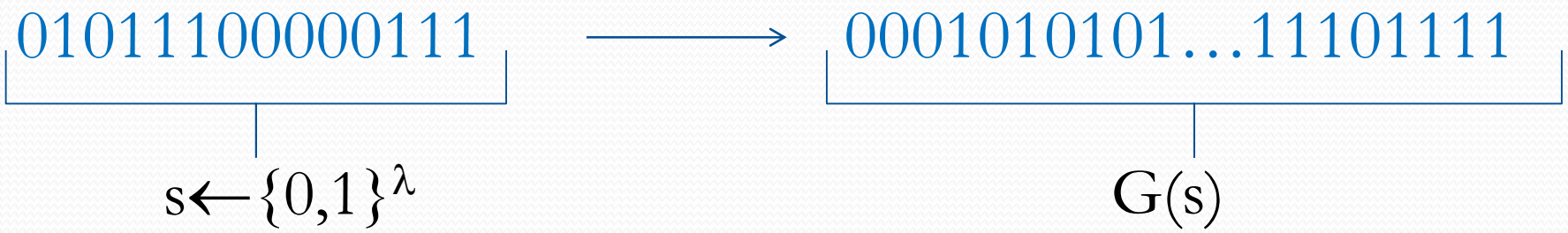
Existe-t-il :

- des fonctions sens uniques ?
  - $x \rightarrow g^x \bmod p$ (logarithme discret), $(p,q) \rightarrow p \times q$ (factorisation)
- des générateurs ou/et fonctions pseudo-aléatoires
  - AES,…
- fonctions de hachage résistantes aux collisions
  - Sha2, MD5….

Fonctions sens uniques

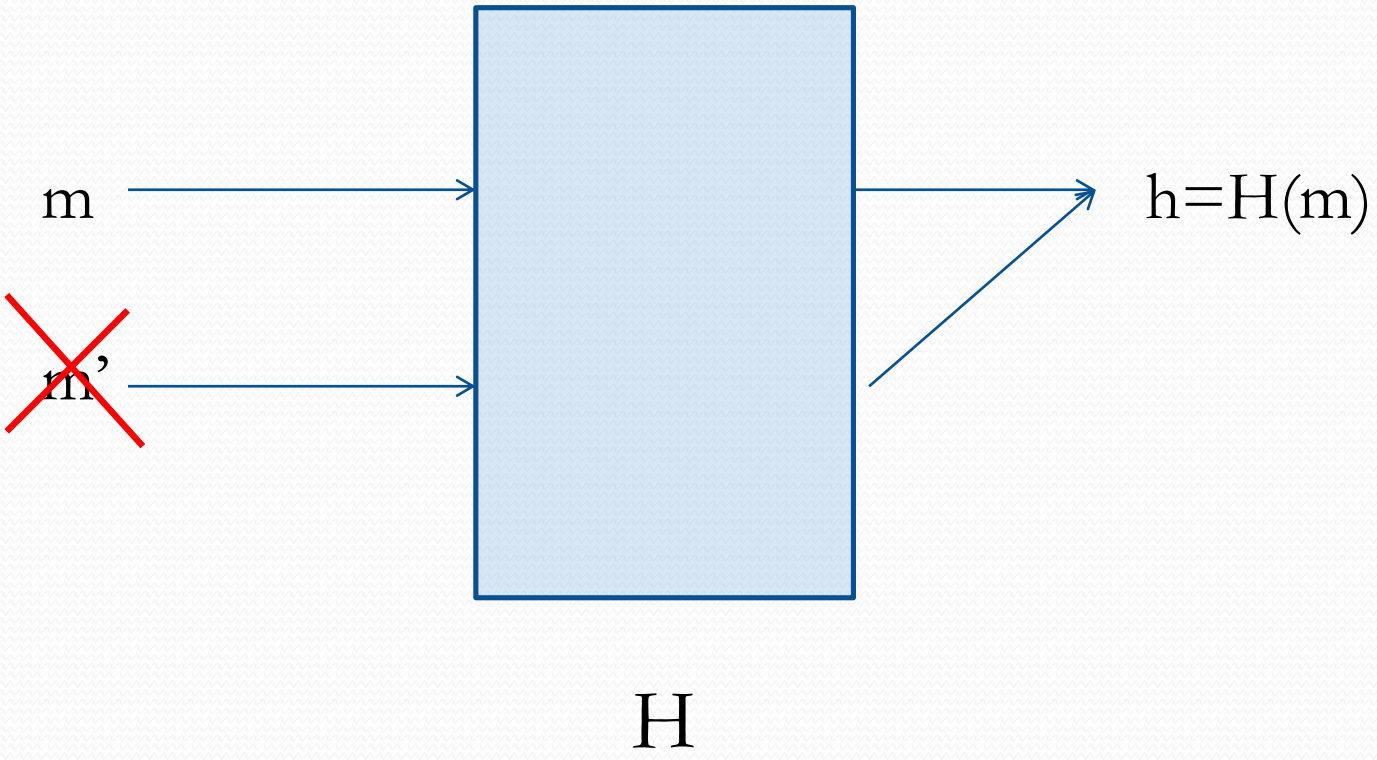


Générateurs pseudo-aléatoires G

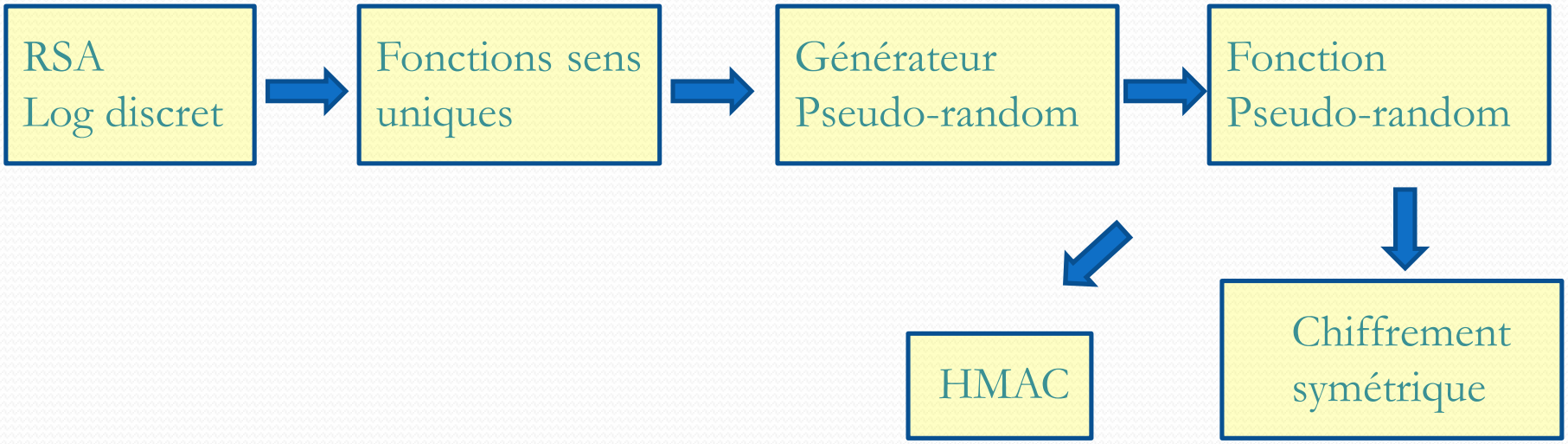


G(s) « ressemble » à une séquence aléatoire

Fonction hachage cryptographique H



Quelques résultats théoriques



Paramètre de sécurité λ

- λ est fixé de manière à garantir qu'aucun algorithme ne puisse effectuer 2^{λ} opérations « élémentaires »
  - Actuellement, $\lambda \approx 100$ ($2^{100} \approx 10^{30}$)
  - De manière duale on estime qu'une probabilité de $1/2^{\lambda}$ est négligeable.
- Il est fixé par des instances de recommandations
  - Analyse prospective sur l'évolution future des ordinateurs
  - N'a pas de dimension « algorithmique »

Rôle de λ

De manière générale, les cryptosystèmes seront paramétrés de manière à ce que toute attaque (connue) requiert plus de $\mathbf{2^{\lambda}}$ **opérations élémentaires.**

Vie réelle → λ → paramètres

IBM
Google
Samsung

$\lambda \approx 100$

e.g. clés de taille 2048 bits pour RSA

Partie 1

Sécuriser les communications face à des attaquants passifs : Cryptographie symétrique

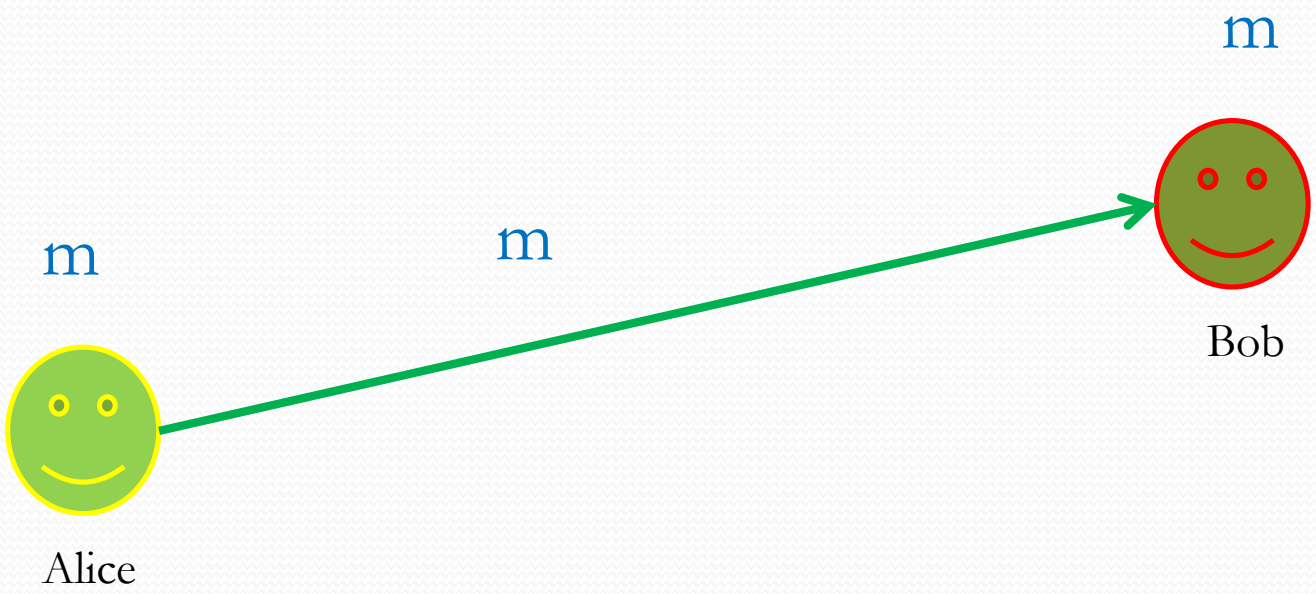
Première mission de la cryptographie : transmettre

message

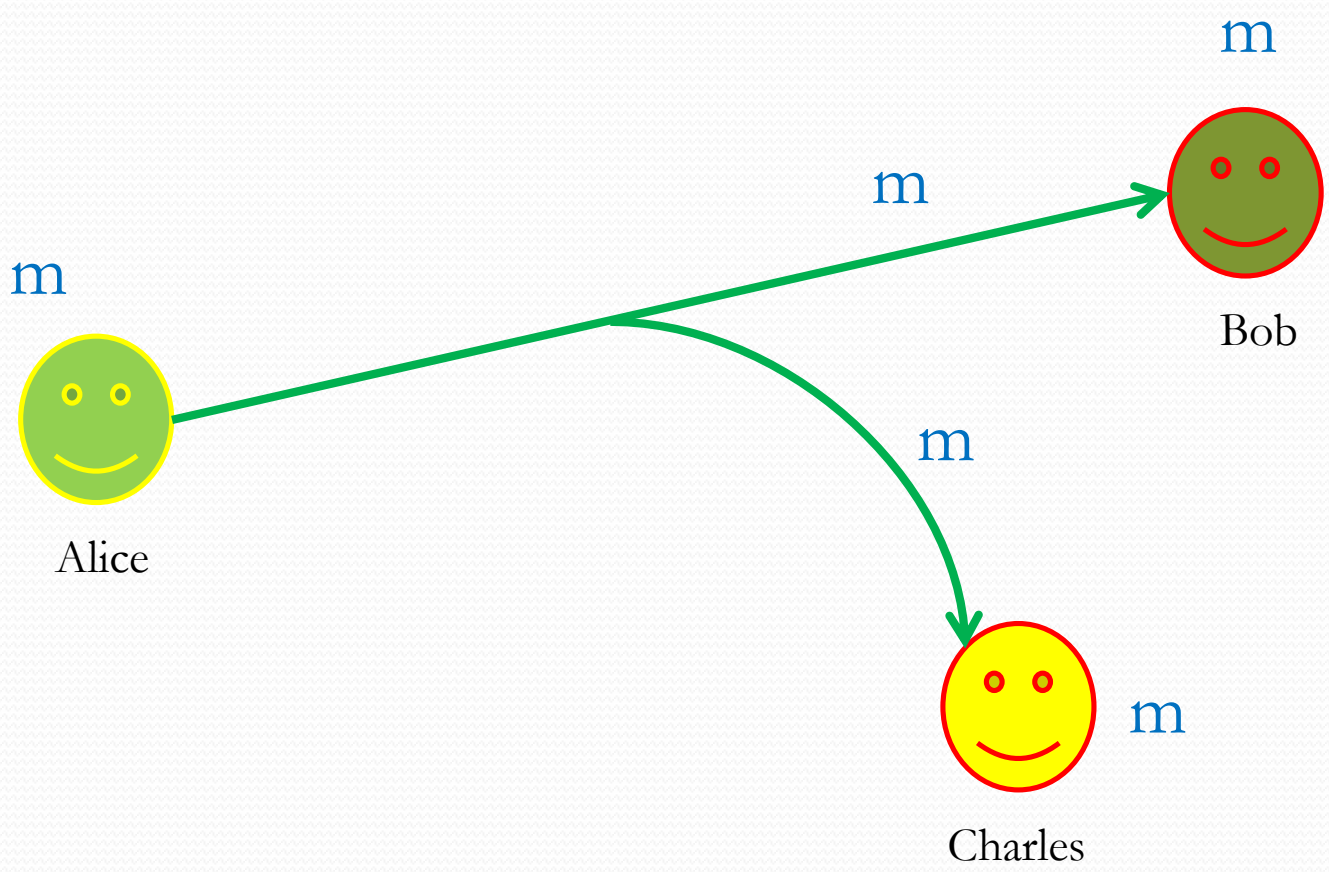
Alice

Bob

Première mission de la cryptographie : transmettre



Première mission de la cryptographie : transmettre



Attaquant

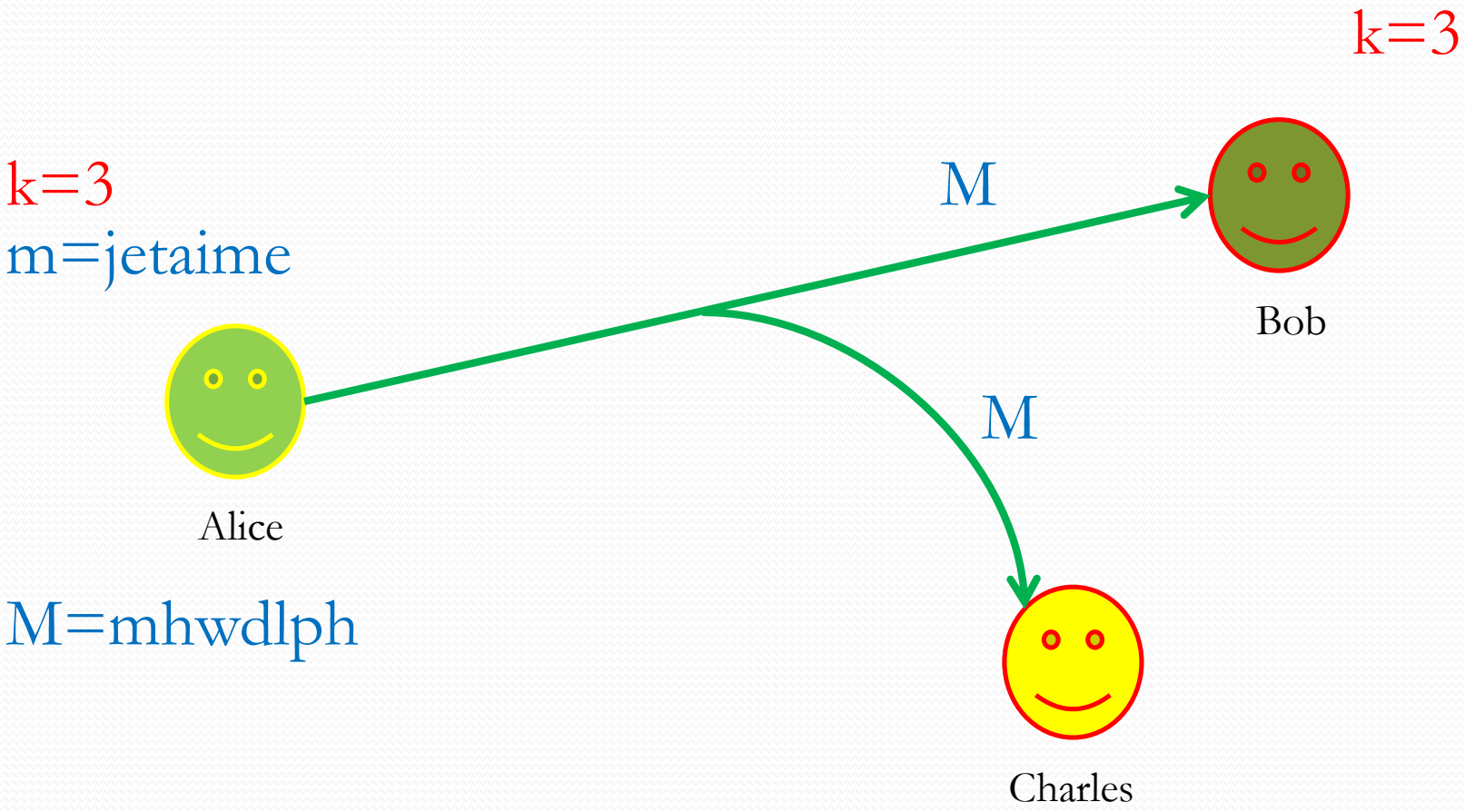
- Charles représente un attaquant (un algorithme)
  - Son objectif est de recouvrer/apprendre/découvrir m
- On supposera que Charles est un attaquant passif
  - Il écoute le réseau
  - Il ne peut pas « écrire » sur le réseau, i.e. modifier le flux
- Un schéma de transmission sera dit sûr/sécurisé si Charles ne peut pas retrouver m.
  - Le schéma précédent n'est évidemment pas sûr

Cryptographie symétrique

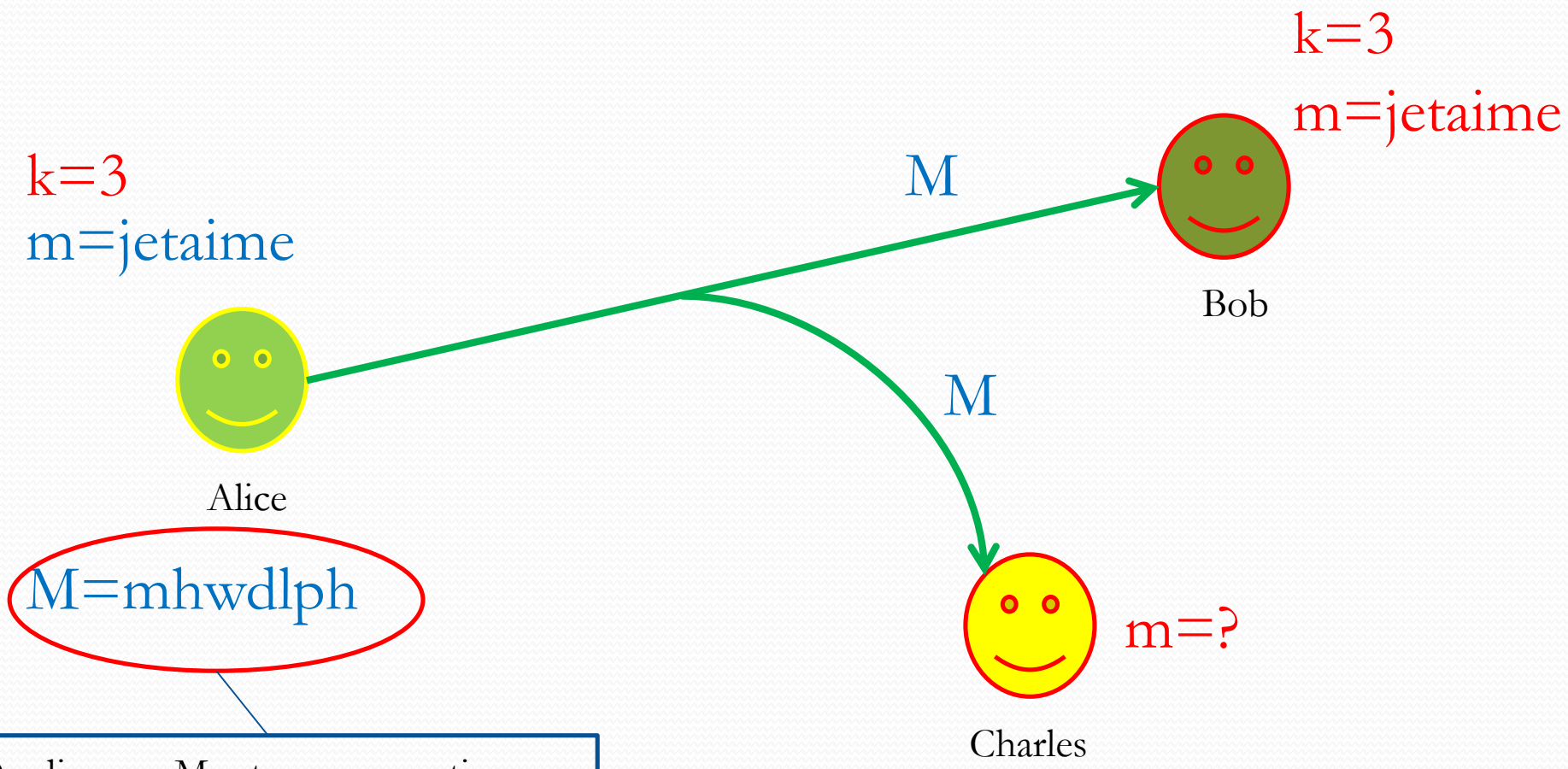
Code de César…

- César décalait les lettres…
  - a→d, b→e, c →f….

Code de César...



Code de César...



Premières conclusions

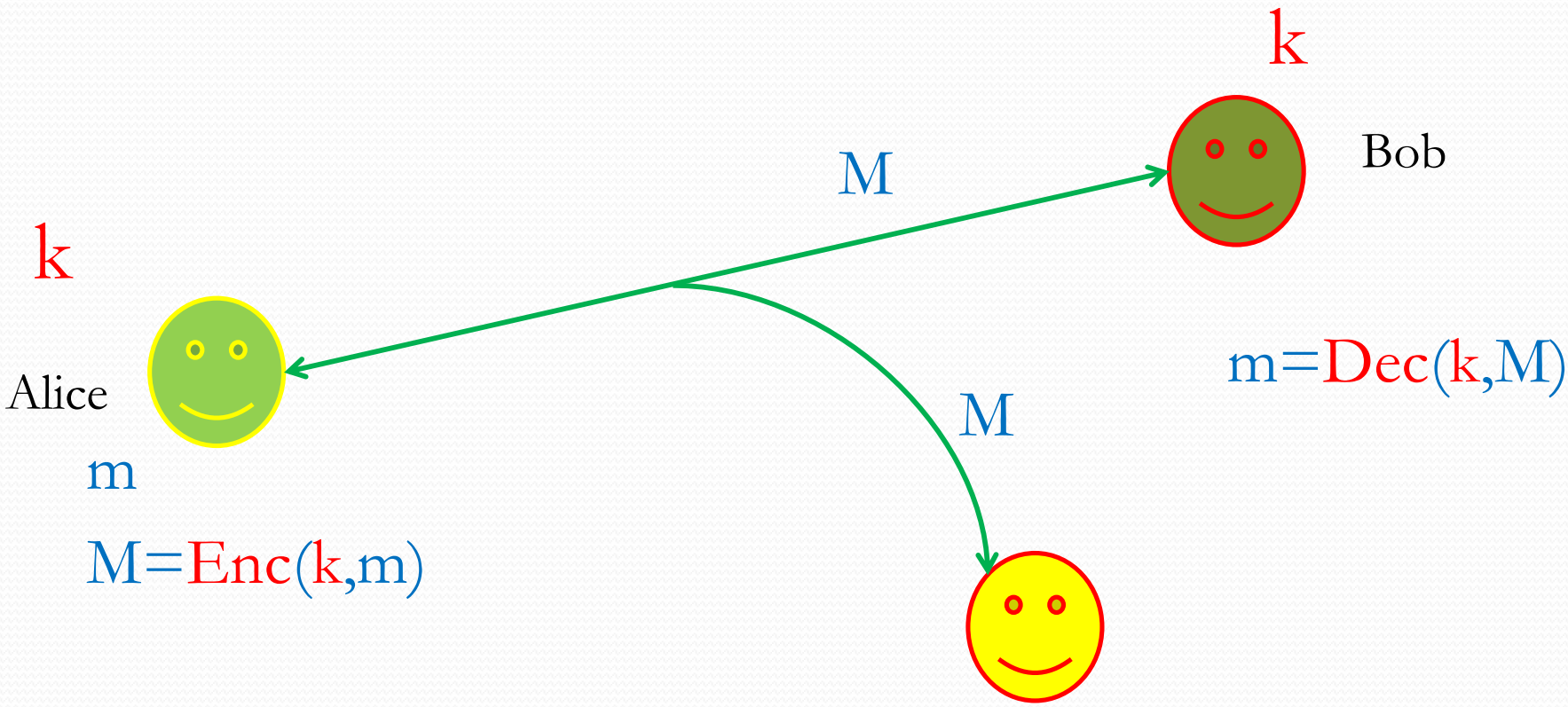
- Trop peu de choix de décalages (clés) k dans le code de césar
  - Un attaquant passif peut tous les essayer (rapidement)

  ⇒ César doit tenir secret son algorithme ⇒ ne peut pas être utilisé à grande échelle.

- Premières conclusions
  - algorithmes de chiffrement sont publiques
  - prennent en entrée une **clé (secrète) choisie aléatoirement dans un espace K suffisamment grand**

Setup. Alice et Bob choisissent secrètement $k \leftarrow Gen(\lambda)$

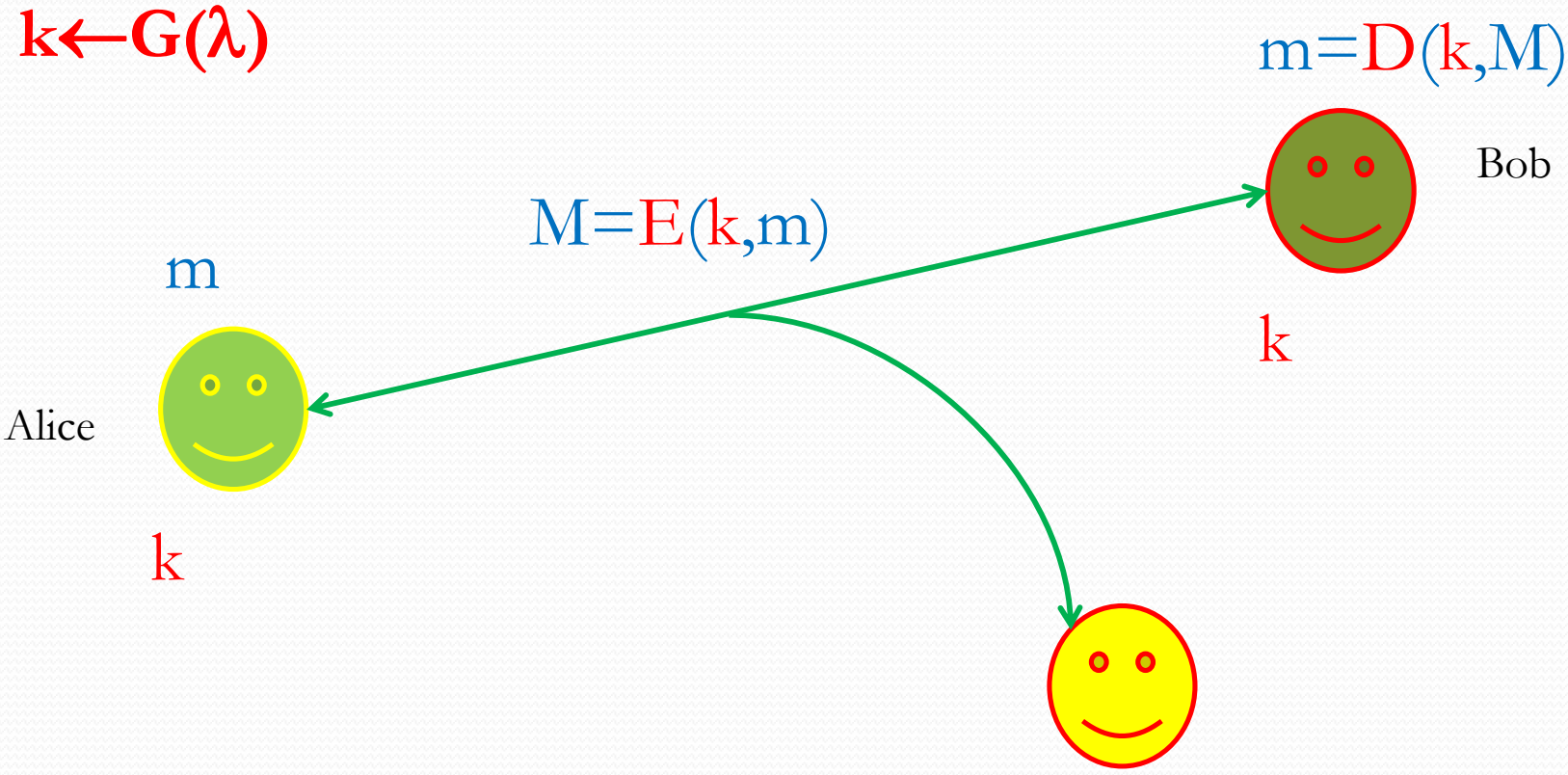
…Comment ?



Quelques remarques

- Le triplet (Gen,Enc,Dec) (ou plus simplement (G,E,D)) est appelé **cryptosystème symétrique publique**.
  - Gen : fonction de génération de clés
  - Enc : fonction d'encryption
  - Dec : fonction de décryption
- La clé $k \leftarrow G(\lambda)$ est secrète (partagée par Alice et Bob)
  - k choisie aléatoirement dans un ensemble de clés $K(\lambda)$ (ou K)
- Connaissant $M=Enc(k,m)$ et k on peut retrouver rapidement **m**
- Connaissant seulement M, on ne peut pas retrouver **m en temps raisonnable.**

Que faire?...



Charles peut tester toutes les clés…

Taille de K

- Charles peut tester toutes les clés
  - Pas de sécurité absolue
- Quelle doit être la taille K ?
  - Suffisamment grande pour qu'on ne puisse pas toutes les tester en temps raisonnable.
- Pour être sur qu'un attaquant ne puisse pas tester toutes les clés par force brute, le cryptosystème devra être paramétré comme suit :

$$|\mathbf{K}| \geq 2^{\lambda}$$

Améliorations du code de césar

- Ne pas se limiter aux permutations circulaires…
  - a→d, b→z, c →u….
  - |K| = ?
- Autre amélioration…ne pas se limiter à une lettre…
  - aaa→dio, aab→ety, aac →fga….

Ces évolutions ne sont pas suffisantes…

- ces « cryptosystèmes » sont tous **vulnérables** aux attaques par analyse fréquentielle des lettres.

Première proposition pour (G,E,D)

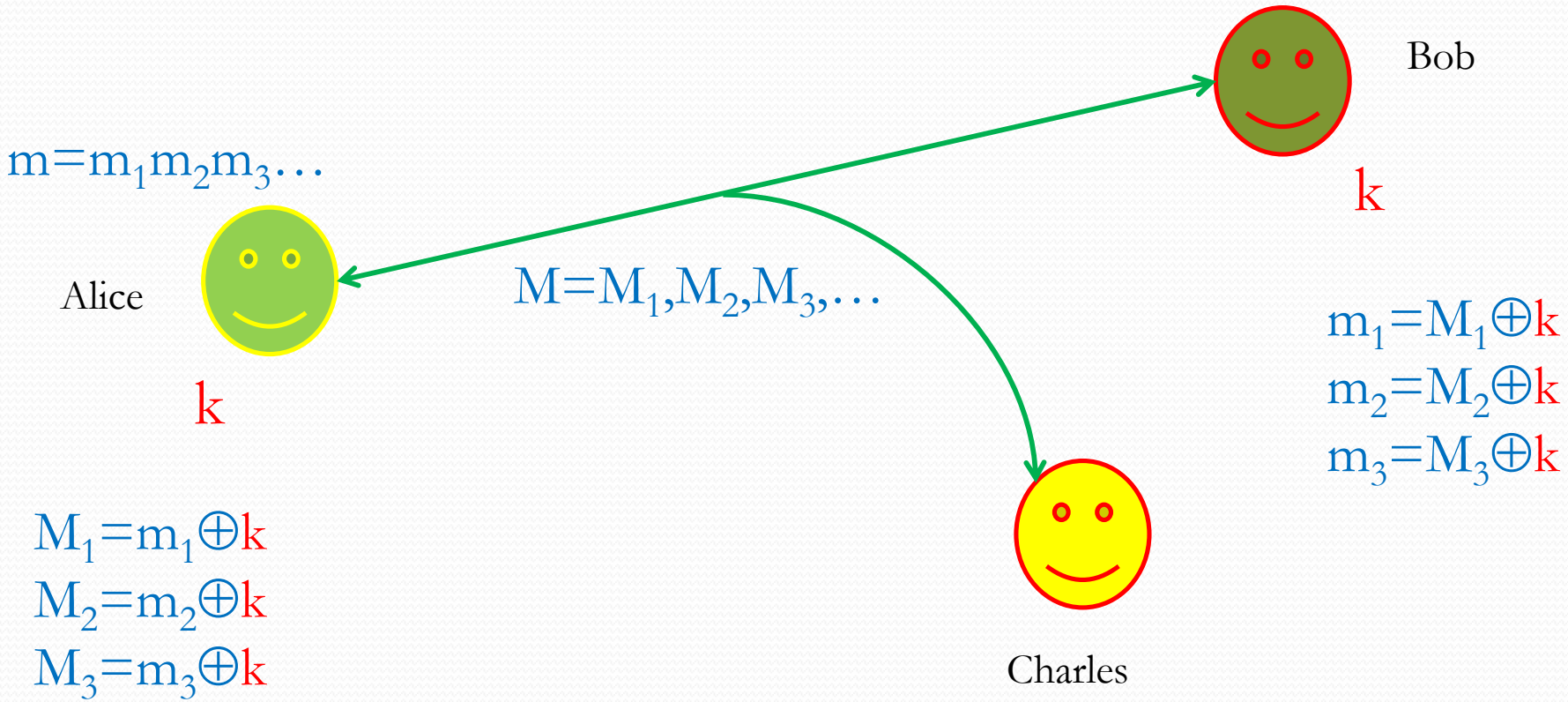
- $K(\lambda)=\{0,1\}^t$ = ensemble des mots binaires de taille $t\geq\lambda$
- On binarise le message

  $m=10011101|00101010|10\ldots.$
- On découpe m en blocks $m_1,m_2,m_3\ldots$de longueur t

  $m= m_1m_2m_3\ldots.$
- Alice envoie $M_1=m_1\oplus k$, $M_2=m_2\oplus k$, $M_3=m_3\oplus k$ …

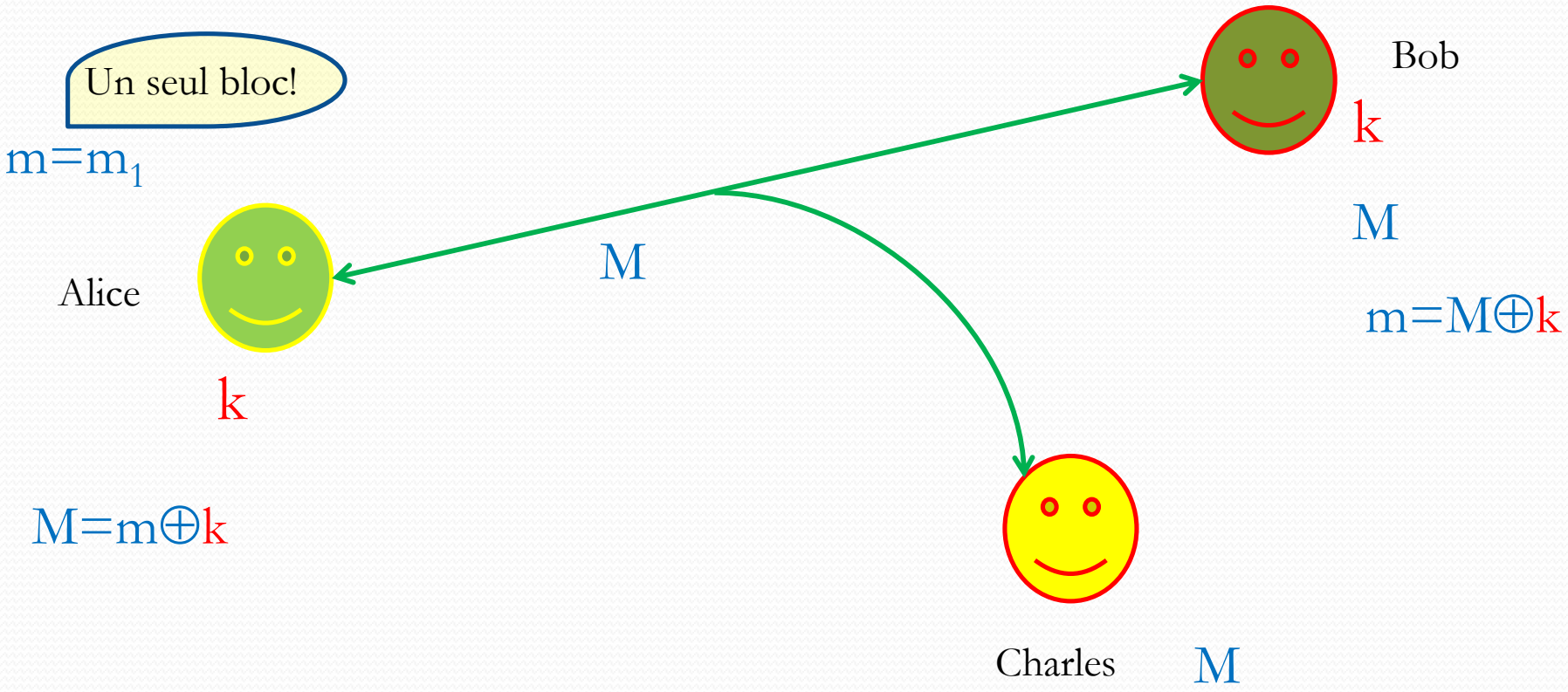
Première proposition pour (G,E,D)

$k \leftarrow G(\lambda)$



Première proposition pour (G,E,D)

$k \leftarrow G(\lambda)$



Est-ce sûr ?

- $|K| \geq 2^{\lambda}$
  - $t \geq \lambda$ car $|\{0,1\}^t| = 2^t$
- Est-ce suffisant pour affirmer la sécurité de ce cryptosystème ?
- D'ailleurs, qu'est ce qu'un cryptosystème sûr ?
  - Intuitivement, ceci signifie que l'attaquant ne peut pas retrouver m à partir de M

Cryptanalyse

- Donner à l'attaquant un pouvoir théorique (qui dépasse celui qu'il pourrait avoir en réalité)
- Peut-il retrouver m avec ce pouvoir ?
- Exemples.
  - L'attaquant à un ordinateur quantique
  - L'attaquant a un *a priori* sur le message, e.g.. il sait que $m \in \{m_0, m_1\}$
- **Remarque.** On supposera toujours que l'attaquant connait (G,E,D) et l'encryption M.

Attaques à messages clairs choisis (CPA)

Pouvoirs classiquement donnés à l'attaquant :

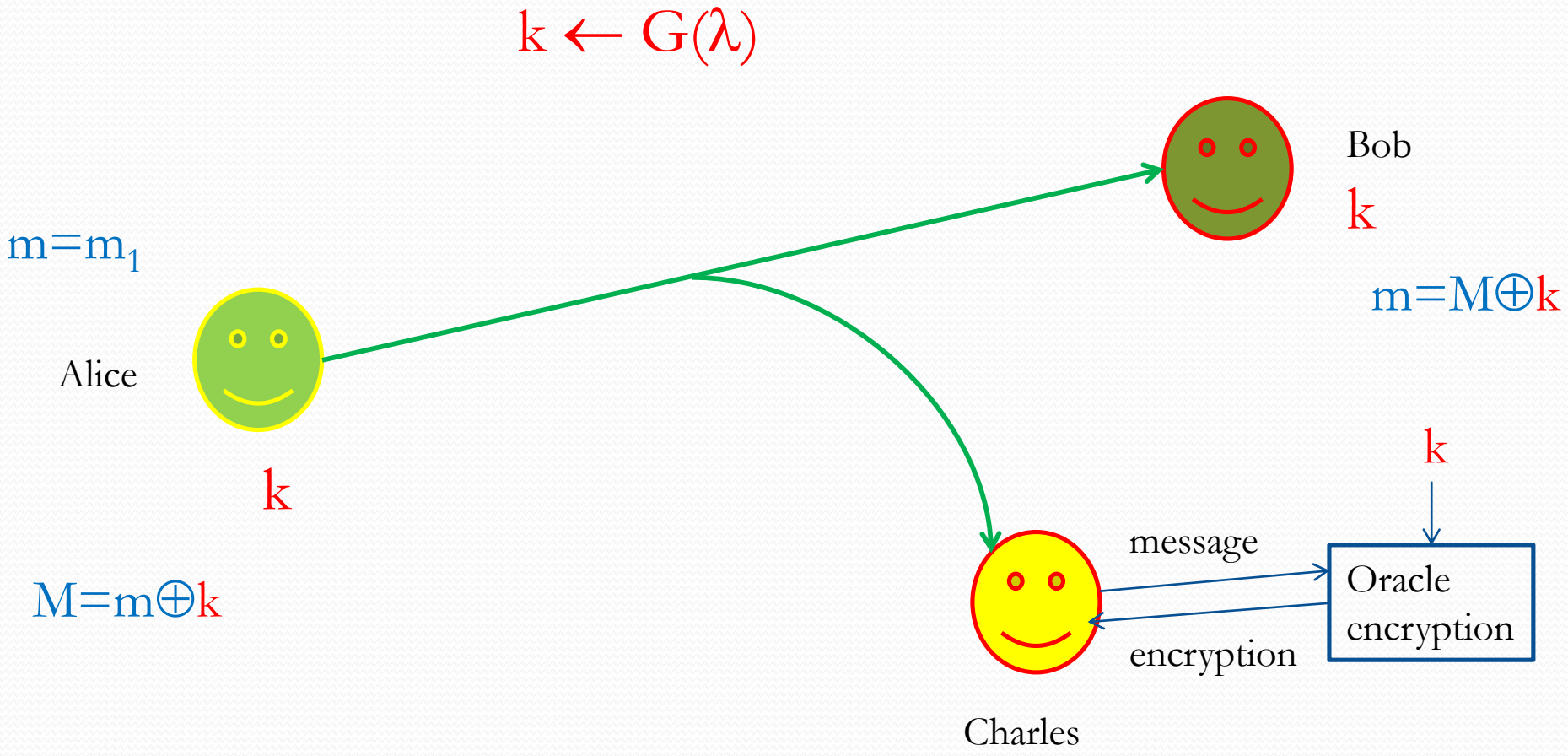
- Il a des ressources classiques…il ne peut pas effectuer plus de 2^{λ} opérations…
  - Cryptographie post quantum…il a un ordinateur quantique
- Il a accès à un oracle d' encryption

$\Rightarrow$ Il peut choisir des messages et obtenir leur encryption.

Classiquement, un cryptosystème est sûr contre tout attaquant ayant de tels pouvoirs est dit CPA-sûr.

Exercice. Proposer un scénario où de tels pouvoirs sont réalistes

Revenons à notre cryptosystème



Est-il CPA-sûr ?

Est-il CPA-sûr ?

Réponse. Non

Est-il CPA-sûr ?

Réponse. Non

Attaque.

1. Charles choisit un message m' arbitrairement
2. Obtient son encryption M' en requêtant l'oracle d'encryption
3. …

Est-il CPA-sûr ?

Réponse. Non

Attaque.

1. Charles choisit un message m' arbitrairement
2. Obtient son encryption M' en requêtant l'oracle d'encryption
3. Calcule $k = m' \oplus M'$
4. Calcule $m = M \oplus k$

Seconde proposition pour (G,E,D)

Idée : permuter les bits

- K = ensemble des permutations de {1,…,t}
- $E(\sigma,m)=(b_1b_2,\ldots b_t))=b_{\sigma(1)}b_{\sigma(1)}\ldots b_{\sigma(t)}$

Exercice. Montrer que ce cryptosystème n'est pas sûr contre des attaques a messages clairs choisis

Seconde proposition pour (G,E,D)

Idée : permuter les bits

- K = ensemble des permutations de {1,…,t}
- $E(\sigma,m)=(b_1b_2,\ldots b_t))=b_{\sigma(1)}b_{\sigma(1)}\ldots b_{\sigma(t)}$

Exercice. Montrer que ce cryptosystème n'est pas CPA-sûr contre des attaques a messages clairs choisis

Indice. Demander à l'oracle une encryption 1000….00, 0100…00, 0010…00, …, 0000…01

En pratique

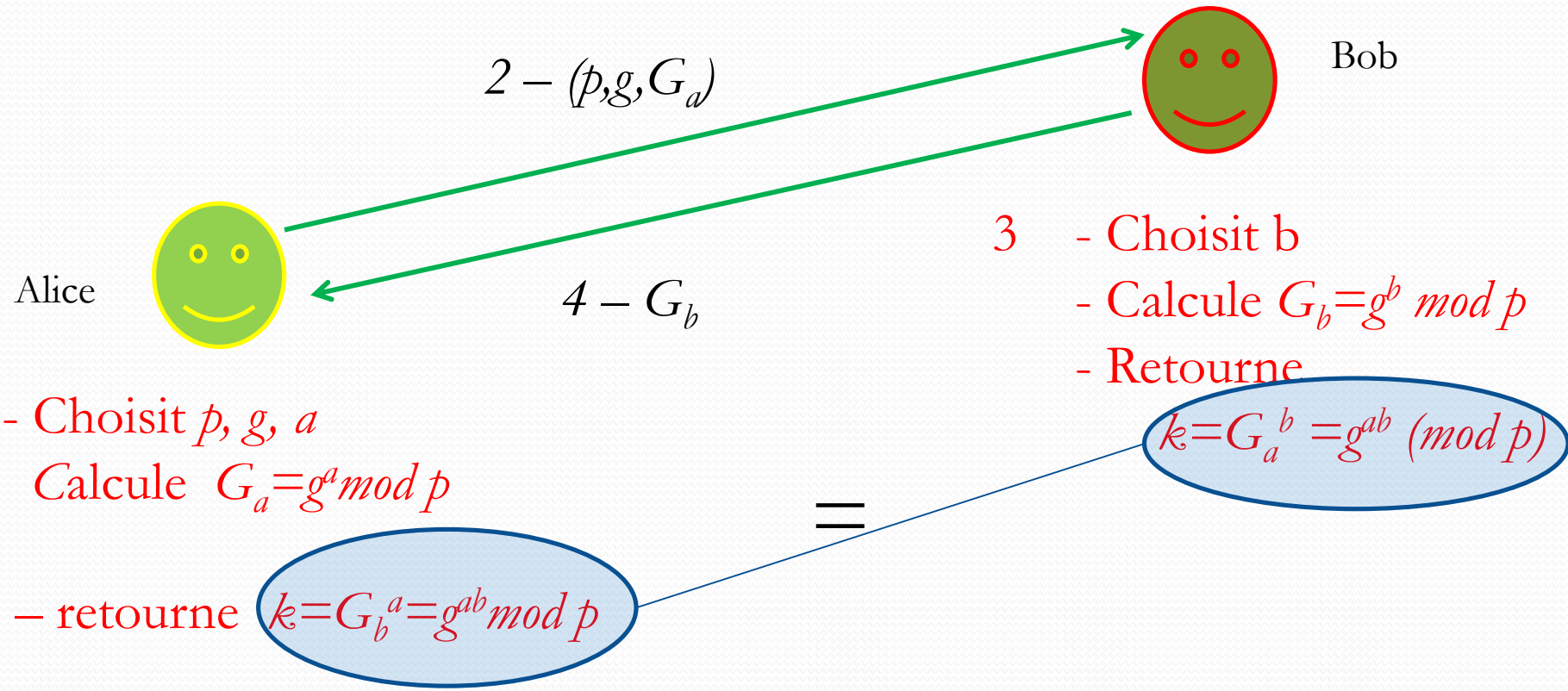
- Les cryptosystèmes utilisés en pratique, e.g. AES reprennent les idées de nos deux cryptosystèmes *maison*
  - Appliquer un masque aléatoire
  - Permuter aléatoirement
  - faire les 2 étapes précédentes un certains nombre de fois (tours)
- Pour obtenir une sécurité plus forte on peut ajouter de l'aléatoire aux messages
  - e.g. on encrypte m'=m||**r** **(r choisi aléatoirement)**

Echange de clés

Echange de clés

- Il reste à résoudre le problème de l'échange de clés *k*
- Diffie-Hellman (1976) propose une solution basée sur le problème DDH ≤ logarithme discret
- Problème du logarithme discret (supposé difficile):
  - Soient
    - Soit p un entier premier grand $(p \geq 2^{\lambda})$
    - $g \leftarrow Z/pZ$
    - $a \leftarrow \{1,\ldots,p-1\}$
  - Retrouver a connaissant $(p,g,g^a \bmod p)$

Echange de clés Diffie-Hellman



Analyse

Sécurité basée sur le problème DDH

Problème DDH

Retrouver $k=g^{ab} \bmod p$ *connaissant* (p,g,G_a,G_b) *est un problème difficile*

Ce que voit passer l'attaquant

Est-ce suffisant?

- A-t-on résolu le problème de la transmission avec un cryptosystème symétrique CPA-sûr ?

Est-ce suffisant?

- A-t-on résolu le problème de la transmission avec un cryptosystème symétrique CPA-sûr ?

Oui si l'attaquant est passif….

Est-ce suffisant?

- A-t-on résolu le problème de la transmission avec un cryptosystème symétrique CPA-sûr ?

Oui si l'attaquant est passif….

Et si les hypothèses faites sont vraies

- cryptosystème CPA-sûr
- DDH est un problème difficile

Est-ce suffisant?

- A-t-on résolu le problème de la transmission avec un cryptosystème symétrique CPA-sûr ?

Oui si l'attaquant est passif….

Et si les hypothèses faites sont vraies

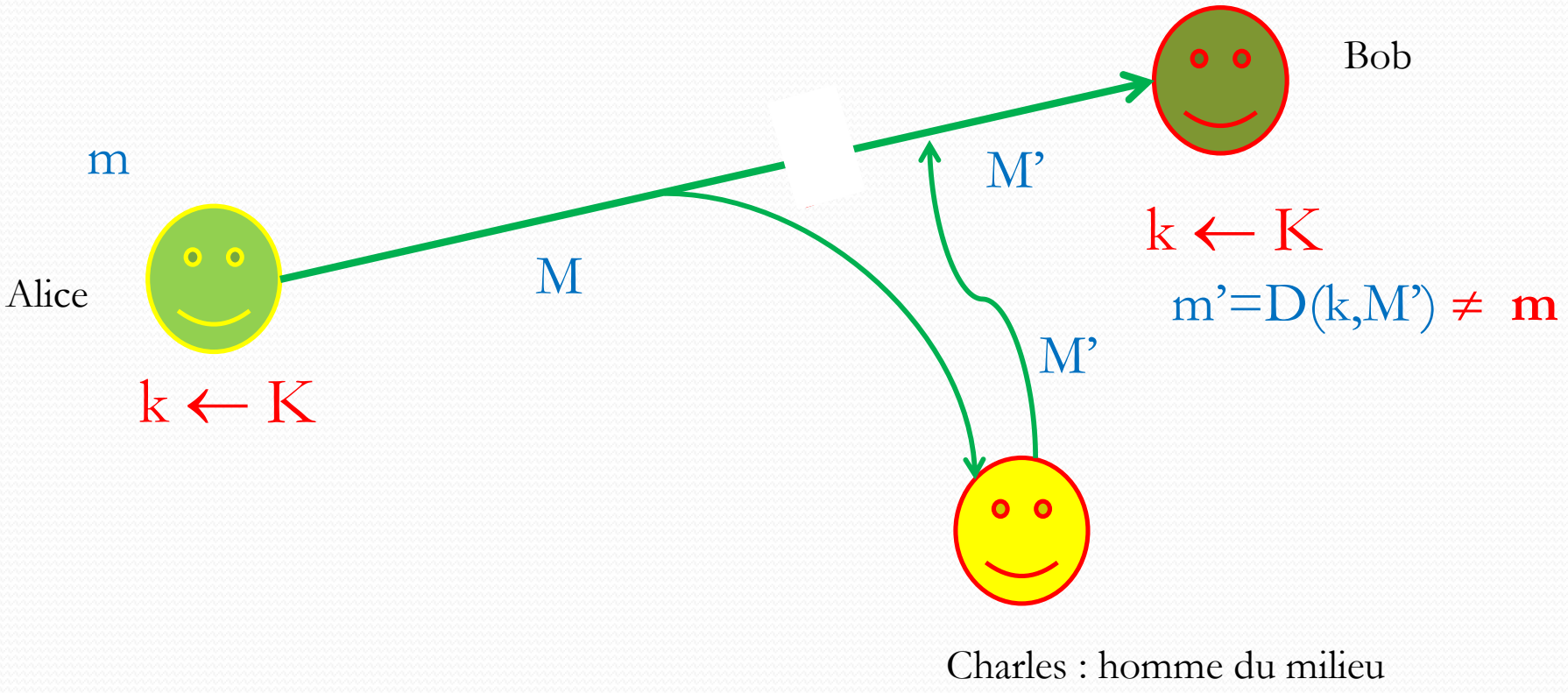
- cryptosystème CPA-sûr
- DDH est un problème difficile

Aucune preuve absolue tant que l'on ne saura pas si P≠NP

Attaquant actif

- Attaquant actif = attaquant pouvant modifier les messages
- Construction précédente n'est plus sûre
  - Perturbe l'échange de clés
  - Authentification de l'expéditeur du message plus garantie

Attaquant actif



Partie 2

Sécuriser les communications face à des attaquants actifs

- MAC
- Cryptographie asymétrique
- signature numérique
- https

MAC

Homme du milieu

- Typiquement un attaquant actif sera appelé « l'homme du milieu »
- Il peut couper la communication entre Alice et Bob
  - Il peut intercepter et remplacer tous les messages qu'il souhaite

⇒Comment Bob peut-il être certain de communiquer avec Alice ?

Problème d'authentification

HMAC

- Supposons qu'Alice et Bob **aient réussi à partager une clé secrète $k \leftarrow K$**
- Supposons qu'il existe une fonction connue de tous (publique)

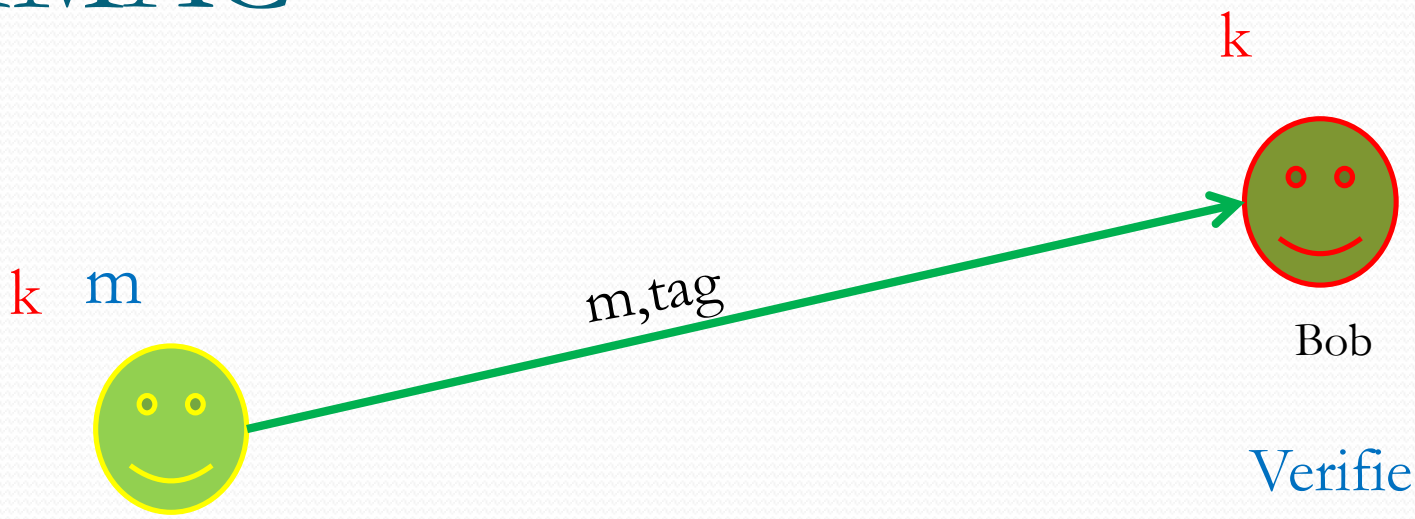
$$H : \{0,1\}^* \times K \rightarrow \{0,1\}^{t \geq \lambda}$$

telle que :

  - $H(m,k)$ soit difficile à calculer sans connaitre k
  - Difficile même si l'on connait $H(m_1,k),\ldots, H(m_r,k)$ pour des messages choisis $m_1,\ldots,m_r$ différents de m

$\Rightarrow$ Alice et Bob pourraient s'identifier mutuellement : HMAC

HMAC

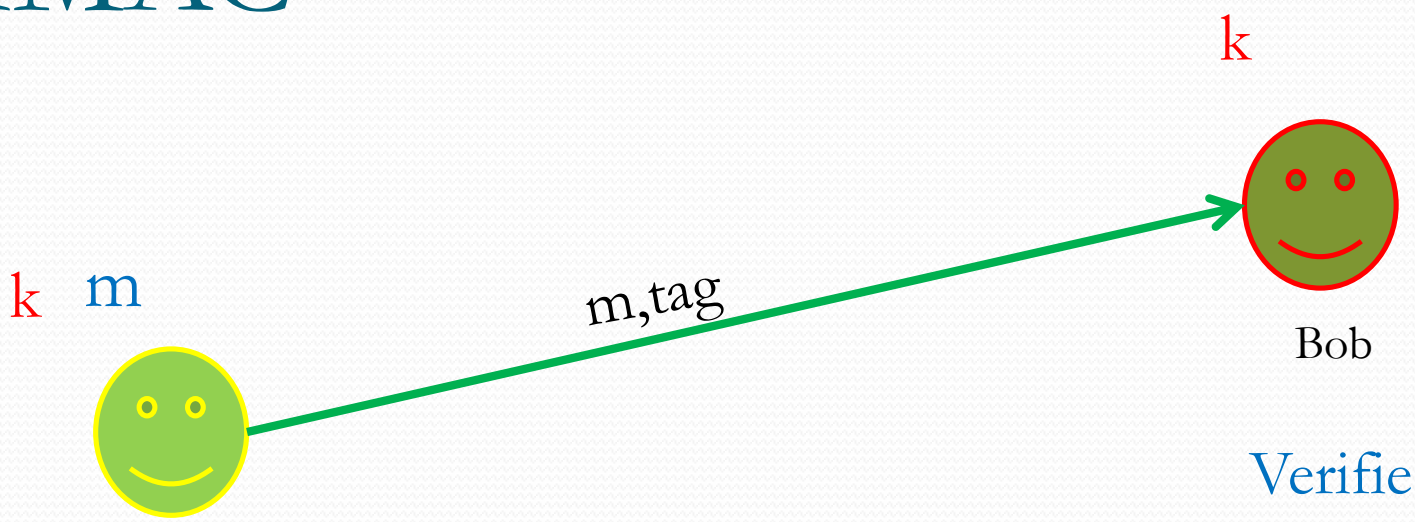


Verifie que :

tag==H(m,k)

tag=H(m,k)

HMAC



Verifie que :

tag==H(m,k)

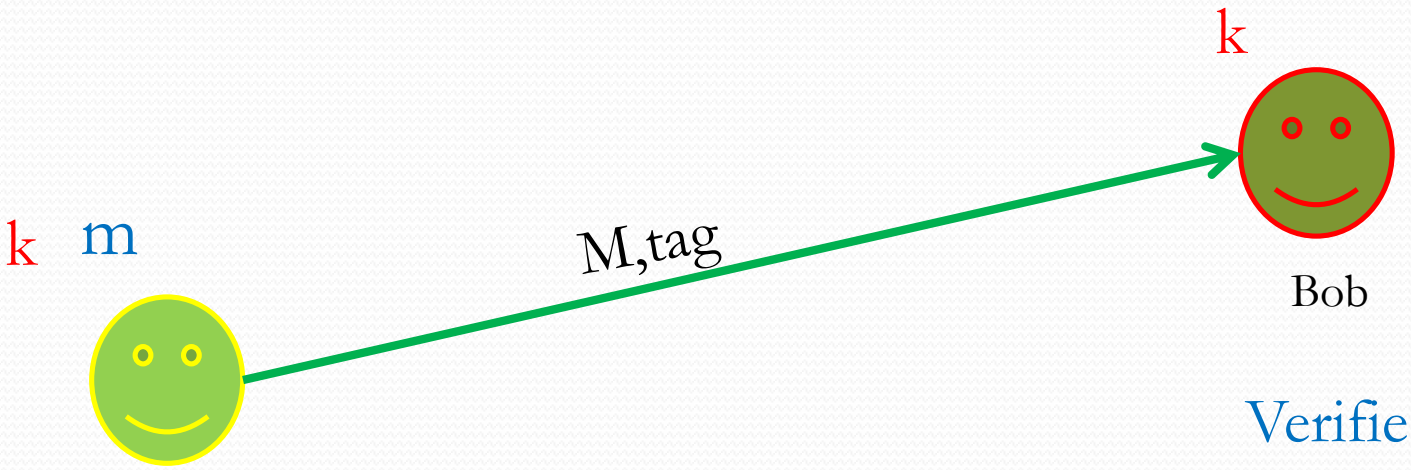
tag=H(m,k)

Si vérification réussie, Bob est certain que le message provienne d'Alice

HMAC

- Sous certaines hypothèses, de telles fonctions H existent
- Des fonctions de hachages cryptographiques ou des fonctions pseudo-aléatoires seront au cœur de la construction de H
  - e.g. SHA 2
- En couplant un chiffrement symétrique avec un HMAC on sécurise un canal de communication
  - sous réserve qu'Alice et Bob partagent une clé secrète k

Chiffrement symétrique + HMAC



Verifie que :

tag==H(M,k)

m=D(k,M)

M=E(k,m)

tag=H(M,k)

Si vérification réussie, Bob est certain que le message provienne d'Alice

Est-ce suffisant ?

- Efficace en pratique
  - utilisé dans https
- N'évite pas les attaques par re-jeu
  - Remplacer (m,tag) par un ancien couple (m^{ancien},tag^{ancien})
  - Une solution ?

Est-ce suffisant ?

- Efficace en pratique
  - utilisé dans https
- N'évite pas les attaques par re-jeu
  - Remplacer (m,tag) par un ancien couple (m^{ancien},tag^{ancien})
  - Numéroter les messager ou utiliser des time-stamps

Est-ce suffisant ?

- Efficace en pratique
  - utilisé dans https
- N'évite pas les attaques par re-jeu
  - Remplacer (m,tag) par un ancien couple (m^{ancien},tag^{ancien})
  - Numéroter les messager ou utiliser des time-stamps
- Cependant on n'a pas sécurisé (contre un attaquant actif)

l'échange de clés

Est-ce suffisant ?

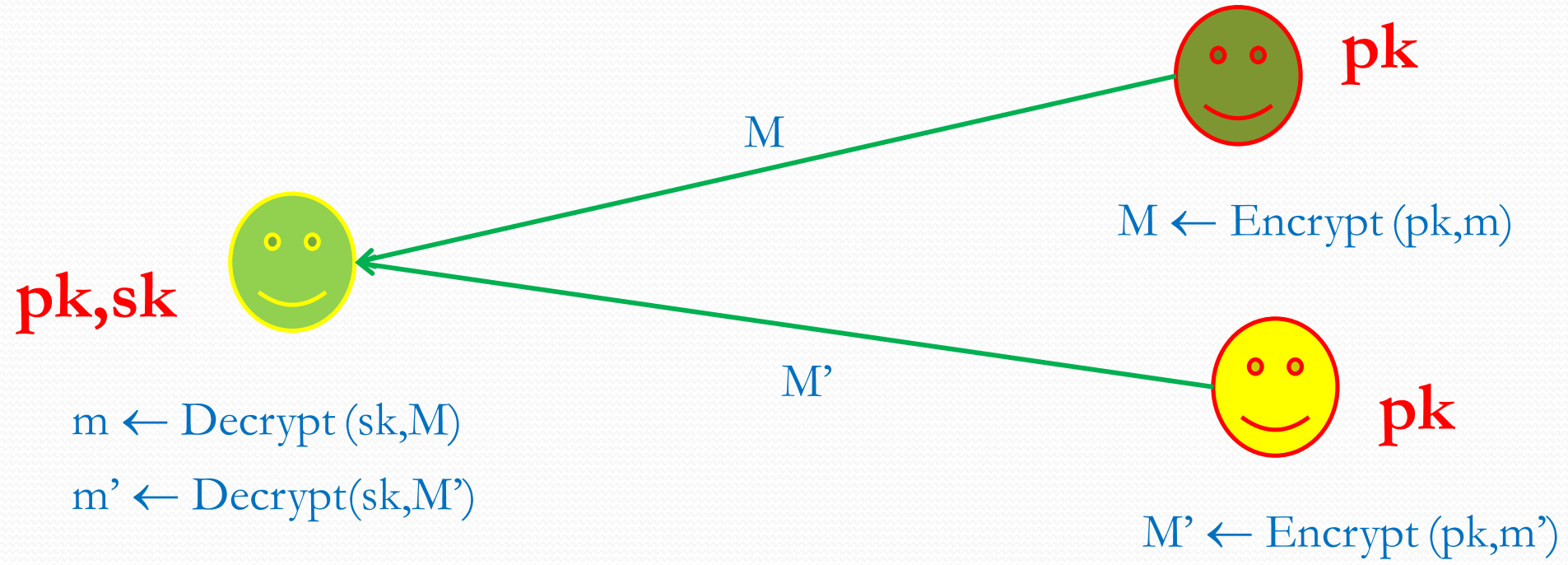
- Efficace en pratique
  - utilisé dans https
- N'évite pas les attaques par re-jeu
  - Remplacer (m,tag) par un ancien couple (m^{ancien},tag^{ancien})
  - Numéroter les messages ou utiliser des time-stamps
- Cependant on n'a pas sécurisé (contre un attaquant actif)

l'échange de clés

⇒ chiffrement asymétrique

Chiffrement asymétrique

Cryptographie asymétrique



Idées

- Alice génère un couple de clés (pk,sk)
  - pk est la clé publique (donc publiée)
  - sk est la clé secrète **(ne passe pas sur le réseau)**
- Pour écrire à Alice, Bob (et les autres) chiffre leur message grâce à pk
- Seule Alice pourra les décrypter

Idées

- Alice génère un couple de clés (pk,sk)
  - pk est la clé publique (donc publiée)
  - sk est la clé secrète (ne passe pas sur le réseau)
- Pour écrire à Alice, Bob (et les autres) chiffre leur message grâce à pk
- Seule Alice pourra les décrypter

Cryptosystème à clé publique

Un cryptosystème à clé publique est un ensemble de 3 fonctions :

- **- KeyGen(λ) :** génère un couple de clés (pk,sk)
- **- Encrypt(pk,x)** : permet d'encrypter un nombre x connaissant la clé publique pk.
- **- Decrypt(sk,c) :** permet de décrypter.

Evidemment, Decrypt(sk,Encrypt(pk,m))=m

Exemples de cryptosystèmes à clé publique

- Merkle-Hellman (Problème du sac à dos)
  - **Cassé par Shamir**
- RSA (problème de la factorisation)
  - **rapide mais pas probabiliste**
- El Gamal (problème du logarithme discret)
  - **Probabiliste et partiellement homomorphe**
- Paillier (problème de la factorisation)
  - **probabiliste et additivement homomorphe**

Cryptographie asymétrique vs symétrique

La cryptographie asymétrique **n'a que des avantages** par rapport à la cryptographie symétrique

Cryptographie asymétrique vs symétrique

La cryptographie asymétrique **n'a que des avantages** par rapport à la cryptographie symétrique

….sauf l'efficacité (rapidité, taille des clés, …)

Cryptosystème sûr?

Premier niveau de sécurité : On ne doit pas pouvoir retrouver sk à partir de pk…en temps raisonnable !!

⇔ il ne doit pas exister d'algorithme polynomial calculant sk à partir de pk

On ne doit pas pouvoir (rapidement) décrypter si l'on ne connait pas sk …recouvre plusieurs notions de sécurité!!

Plusieurs niveaux de sécurité

- Sécurité sens unique (one-way)
  - RSA
- Sécurité sémantique
  - Paillier
- Sécurité avec oracle de décryption
  - One-pad RSA

Securité de base : sens unique

Le prouveur

1 - (pk,sk)←KeyGen(λ)
m ← Domaine d'encryption
M ← Encrypt(pk,m)

Attaquant A

Securité de base : sens unique

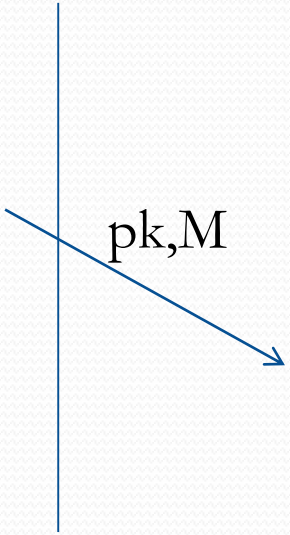
Le prouveur

Attaquant A

1 - (pk,sk)←KeyGen(λ)

m ← Domaine d'encryption

M ← Encrypt(pk,m)



Securité de base : sens unique

Le prouveur

1 - (pk,sk)←KeyGen(λ)
m ← Domaine d'encryption
M ← Encrypt(pk,m)

Attaquant A

pk,M

2 - m'←A(M,pk)

Securité de base : sens unique

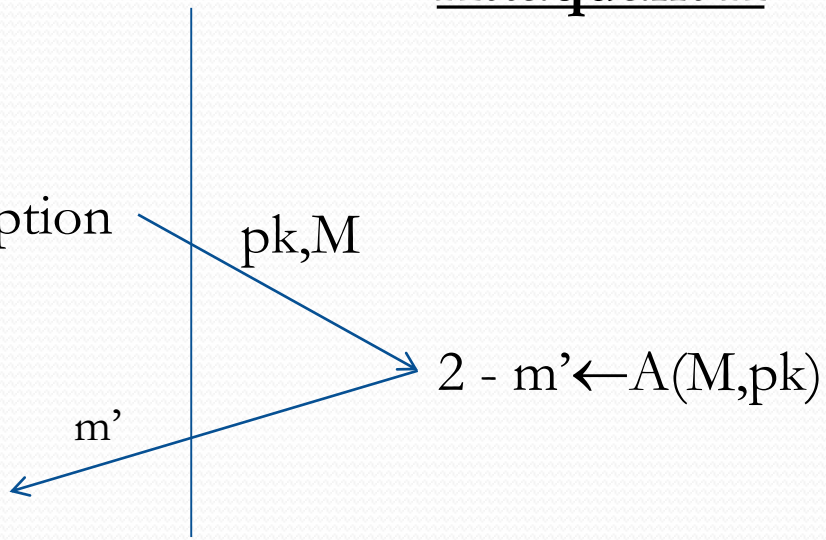
Le prouveur

Attaquant A

1 - (pk,sk)←KeyGen(λ)

m ← Domaine d'encryption

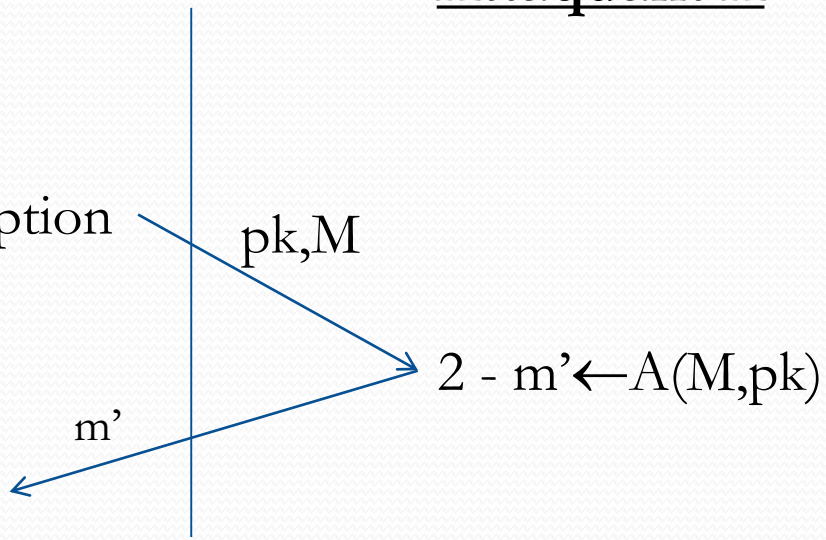
M ← Encrypt(pk,m)



Securité de base : sens unique

Le prouveur | **Attaquant A**

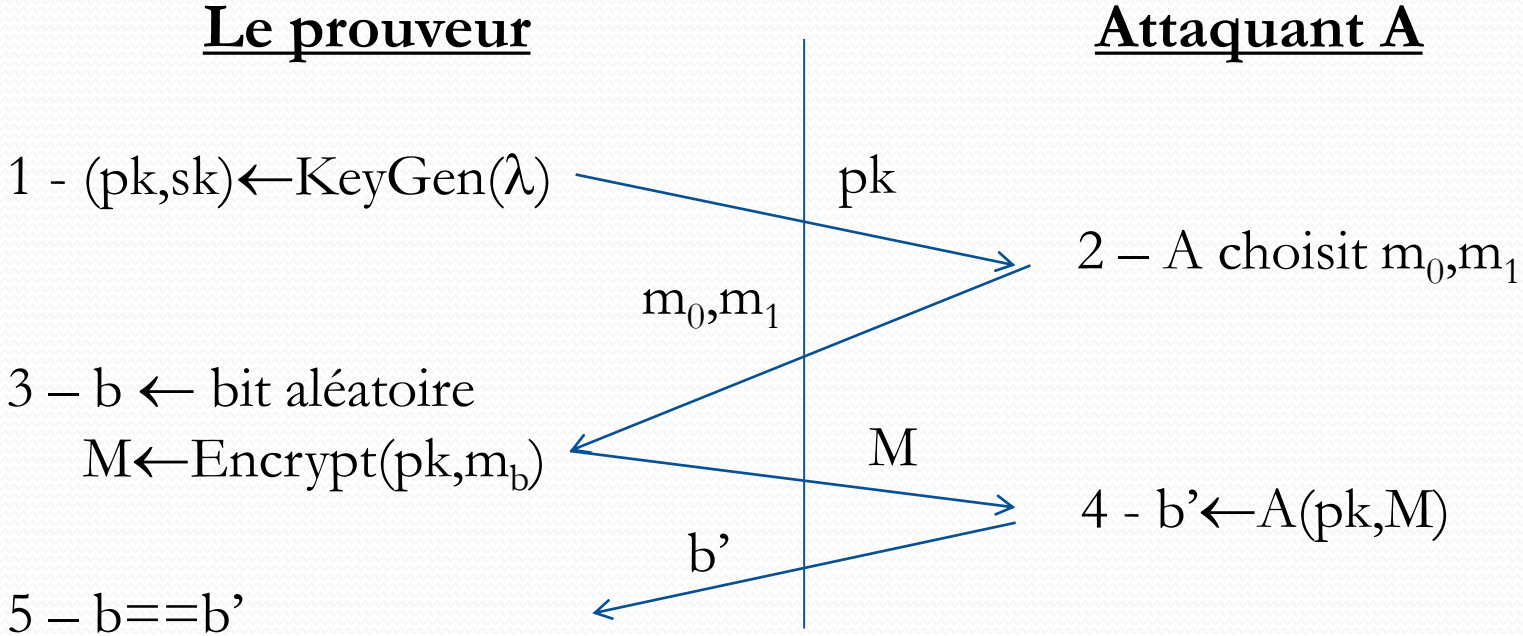
1 - (pk,sk)←KeyGen(λ)
m ← Domaine d'encryption
M ← Encrypt(pk,m)



3 – m==m'

S'il n'existe pas d'attaquant polynomial tel que m==m ' avec une probabilité non-négligeable alors on parle de **sécurité sens unique**.

Securité sémantique



S'il n'existe pas d'attaquant polynomial tel que b==b' avec une probabilité significativement supérieure à 1/2 alors on parle de **sécurité sémantique**.

Questions

- Quelque doit être le niveau de sécurité d'un cryptosystème utilisé pour faire du vote électronique?
- RSA est-il sémantiquement sur? Pourquoi?

Et encore….

Les cryptosystèmes homomorphes sont à la base de beaucoup d'autres applications

- Signatures de groupe
- Calcul multi-parties
  - Vote électronique
  - Transfert inconscient
  - …
  - Machine learning
- Externalisation de calculs
- …

Un peu d'arithmétique Modulaire…

Modulo (mod)

- Deux significations du terme « mod »
- a mod n = reste de la divivion de a par n
  - 27 mod 5 = 2
  - -28 mod 5 = 2
- $a \equiv b \bmod n$ signifie $a=b+kn$ $(k \in \mathbf{Z})$
  - $27 \equiv 2 \bmod 5$
  - $27 \equiv -28 \bmod 5$

mod $\Leftrightarrow$ mod

- Les deux significations se rejoignent!

Proposition. Soient a et b deux entiers

$$a \equiv b \bmod n \Leftrightarrow a \bmod n = b \bmod n$$

- **Exemple**

$12 \equiv 17 \bmod 5 \Leftrightarrow 12 \bmod 5 = 17 \bmod 5 \ (=2)$

Calculs modulo

Proposition. Si $a \equiv a' \bmod n$ et $b \equiv b' \bmod n$ alors

1. $a+b \equiv a'+b' \bmod n$
2. $a \times b \equiv a' \times b' \bmod n$

Exemple.

$12 \equiv 17 \bmod 5$ et $11 \equiv 1 \bmod 5$

$\Rightarrow 12+11 \equiv 17+1 \bmod 5$

$\Rightarrow 12 \times 11 \equiv 17 \times 1 \bmod 5$

Calculs modulo

Proposition fondamentale. Soient a et b deux entiers

1. $a+b \bmod n = (a \bmod n)+ (b \bmod n) \bmod n$
2. $a \times b \bmod n = (a \bmod n) \times (b \bmod n) \bmod n$

Cette proposition est le fondement de l'arithmétique modulaire

Exemples

- $12 \times 8 + 10 \bmod 5$

$= 106 \bmod 5$

$=1$

- $12 \times 8 + 10 \bmod 5$

$= (12 \bmod 5) \times (8 \bmod 5) + (10 \bmod 5) \bmod 5$

$= 2 \times 3 + 0 \bmod 5$

$= 1$

Exemples (suite)

$(2059 \times 9876 \times 1234+1004) \bmod 5$

$= (4 \times 1 \times 4+4) \bmod 5$

$= (16 \bmod 5+4 \bmod 5) \bmod 5$

$= (4+1) \bmod 5 = 0$

Remarque. On met des « modulos » où l'on veut pourvu qu'on en mette un à la fin

Structure $\mathbf{Z}/n\mathbf{Z}$ (ou $\mathbf{Z}_n$)

- $\mathbf{Z}/n\mathbf{Z}$ est une structure arithmétique finie donc manipulable informatiquement
  - Découle directement des propositions précédentes
  - Mêmes propriétés que dans $\mathbf{Z}$ ou $\mathbf{R}$
- L'intérêt est de travailler dans $\mathbf{Z}/n\mathbf{Z}$ est d'alléger les notations
  - **Plus besoin de mod, ≡**
  - **Calculer dans $\mathbf{Z}/n\mathbf{Z}$ signifie « calculer modulo n »**

$\mathbf{Z}/n\mathbf{Z}$ (ou $\mathbf{Z}_n$)

- $\mathbf{Z}/n\mathbf{Z}$ est l'ensemble des entiers $\{0,1,2,\ldots,n-1\}$ dans lequel on définit l'addition $\oplus$ et la multiplication $\otimes$ comme suit :
  - $\mathbf{a \oplus b = a+b \bmod n}$
  - $\mathbf{a \otimes b = a \times b \bmod n}$
- Les opérations $\oplus$ et $\otimes$ ont les mêmes propriétés que l'addition et la multiplication dans $\mathbf{Z}$ ou $\mathbf{R}$
  - **Commutativité**
  - **Associativité**
  - **Distributivité de la multiplication / addition**

Addition dans $\mathbb{Z}/5\mathbb{Z}$

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Multiplication dans $\mathbb{Z}/5\mathbb{Z}$

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Calculs dans $\mathbf{Z}/n\mathbf{Z}$

- L'addition et la multiplication ont les mêmes propriétés que dans **Z** ou **R**
  - $\mathbf{Z}/n\mathbf{Z}$ un anneau
- Les opérations $\oplus$ et $\otimes$ seront simplement notés $+$ et $\times$
- Lorsque l'on calcule dans Z/nZ on n'a pas besoin d'utiliser les symboles $\equiv$ et mod

Exemples

- Si l'on précise que l'on travaille dans Z/5Z, on peut écrire que :
  - $2+4=1$
  - $3\times3=4$
  - $4^8 = 4^2 \times 4^2 \times 4^2 \times 4^2 = 1 \times 1 \times 1 \times 1 = 1$
- Lorsque l'on calcule dans $\mathbf{Z}/n\mathbf{Z}$ on peut calculer comme dans $\mathbf{Z}$ et « finir par un modulo n »
  - dans Z/5Z, $4^8 = 65536 \bmod 5 = 1$
  - Cependant faire dés modulos après chaque opérations évite que les nombres deviennent trop grands

Exemples

Plaçons nous dans $\mathbf{Z}_5$

- $2+3 = 0$
- $4 \times 4 = 1$
- $4 \times 3 \times 2 + 3^4$

$= (24 \bmod 5) + (81 \bmod 5) \bmod 5$

$= 4+1 \bmod 5$

$= 0$

$= (4 \times 3) \times 2 + 3 \times (3 \times (3 \times 3))$

$= 2 \times 2+3 \times (3 \times 4)$

$= 4 + 3 \times 2$

$= 4+ 1$

$=0$

Exercice. Calculer $4^{2021} \bmod 5$

Exemples

Plaçons nous dans $\mathbf{Z_{15}}$

- $3 \times 5 = 0$ (!)
- $4 \times 4 = 1$
- $4 \times 3 \times 2 + 3^4$

$= (24 \bmod 15) + (81 \bmod 15) \bmod 15$

$= 9 + 6 \bmod 15$

$= 0$

Exercice. Que vaut $4 \times 3 \times 2 + 3^4$ dans $\mathbf{Z_{17}}$?

Comme dans **R** ou **Z** mais…quelques bizarreries!!!

- Diviseurs des 0
  - Dans $\mathbf{Z}/15\mathbf{Z}$, $3\times5 = 0$
  - Pas de diviseur dans $\mathbf{Z}/n\mathbf{Z}$ si n est premier
- Pour tout $x \in \mathbf{Z}/7\mathbf{Z}$, $x^7-x = 0$
  - Théorème de Fermat
- Pour tout $x \in \mathbf{Z}/30\mathbf{Z}$, $x^5-x = 0$
  - Une équation de degré 5 ayant 30 solutions !!!

Eléments inversibles

- Plaçons nous dans $\mathbf{Z}_n$
- On dit que y est l'inverse de x (que l'on notera x^{-1}) si $xy=1$

Exercice.

1 – Quel est l'inverse de 3 modulo 7?

2 – Quel est l'inverse de 3 modulo 15

On note $\mathbf{Z}_n^*$ ou $(=\mathbf{Z}/n\mathbf{Z})^*$ l'ensemble des éléments qui ont un inverse dans $\mathbf{Z}_n$.

Exercice . Combien y-a-t il d'éléments inversibles dans $\mathbf{Z}_7$, $\mathbf{Z}_{15}$?

Fonction d'Euler ϕ

- $\mathbf{Z}_n^*$ contient tous les éléments $x \in \{0,\ldots,n-1\}$ tel que

  $pgcd(x,n)=1$

- $Card(\mathbf{Z}_n^*) = \phi(n)$: fonction d'Euler
- Soit $n=\Pi\, p_i^{ei}$ la décomposition en nombre premiers de n.

  $\phi(n)=\Pi\, p_i^{ei-1}\,(1-p_i)$

- Si n premier alors $\phi(n)=(n-1)$
- Si $n=pq$ alors $\phi(n)=(p-1)(q-1)$

Euclide pour trouver un inverse…

- Grâce à Euclide….
  - Et aux coefficients de Bezout
- (d,u,v)←EuclideEtendu(a,n)
  - d=pgcd(a,b)
  - ua+vn=d
- Si pgcd(a,b)=1
  - ua=1-vn ⇒ ua ≡ 1 mod n ⇒ (u mod n) a ≡ 1 mod n

⇒ **a^{-1}=(u mod n)**

Petit théorème de Fermat-Euler

Théorème. $\forall a \in \mathbf{Z}_n^*$,

$$\mathbf{a^{\phi(n)}=1}$$

Formulations équivalentes :

1. Si pgcd(a,n)=1 alors $a^{\phi(n)} \bmod n = 1$
2. Si pgcd(a,n)=1 alors $a^{\phi(n)} \equiv 1 \bmod n$
3. Si pgcd(a,n)=1 alors $a^{\phi(n)} = 1+kn$
4. Si pgcd(a,n)=1 alors $a^{\phi(n)}-1$ est un multiple de n

Exemples

- $2^{\phi(7)} \bmod 7 = 2^6 \bmod 7 = 64 \bmod 7 = 1$
- $2^{\phi(15)} \bmod 15 = 2^8 \bmod 15 = 256 \bmod 15 = 1$

Exemples

$96^{94} \bmod 47$

$=(96 \bmod 47)^{94}$

$=2^{94} \bmod 47$

or $\phi(47)=46 \Rightarrow 94=2\,\phi(47)+2$

d'où $2^{94} \bmod 47$

$=2^{2\,\phi(47)+2} \bmod 47$

$=2^2 \times (2^{\phi(47)})^2 \bmod 47$

$=4 \times 1^2 \bmod 47$

$=4$

Généralisation

$$a^b \bmod n = (a \bmod n)^{b \bmod \phi(n)} \bmod n$$

« les exposants peuvent être réduits modulo $\phi(n)$ »

Exercice. Que vaut $55555^{55555} \bmod 7$?

Généralisation

$$a^b \bmod n = (a \bmod n)^{b \bmod \phi(n)} \bmod n$$

« les exposants peuvent être réduits modulo $\phi(n)$ »

Exercice. Que vaut $55555^{55555} \bmod 7$?

Solution.

$55555^{55555} \bmod 7$

$= (55555 \bmod 7)^{55555 \bmod 6} \bmod 7$

$= 3^1 \bmod 7$

$= 3$

Exercice.

Soient $p = 17$ et $q = 11$ et $n = 11 \times 17$

1. Montrer que $3 \times 107 \equiv 1 \bmod 160$
2. Resoudre $m^{107} = 9$ dans $\mathbf{Z}_n$
3. Vérifier

Exercice.

Soient $p = 17$ et $q = 11$ et $n = 11 \times 17 = 187$

1. Montrer que $3 \times 107 \equiv 1 \bmod 160$
2. Resoudre $m^{107} = 9$ dans $\mathbf{Z}_n$
3. Vérifier

Solution.

1. $3 \times 107 = 1 + 2 \times 160$
2. $\phi(n) = 160$

   $9^3 = 168 = (m^{107})^3$ (dans $\mathbf{Z}_{187}$)

   Or $(m^{107})^3 = m^{1+2\times 160} = m^1 \times (m^{160})^2 = m \times 1^2 = m$

   Donc **$m = 168$**

Test de primalité

- Il s'agit que décider si un nombre n est premier ou non
- Une méthode : chercher un facteur de n
  - cela revient à factoriser n
- Est-ce efficace?
- Connaissez vous une méthode plus efficace ?

Test de Fermat

Idée.

$\phi(n)=n-1 \Leftrightarrow$ n est premier

$\Rightarrow$

- $2^{n-1} \bmod n = 1$ si n est premier
- il n' y a aucune raison que $2^{n-1} \bmod n = 1$ si n n'est pas premier

TestFermat(n)

retourner ($2^{n-1} \bmod n == 1$)

Efficace ?

Exponentiation modulaire

But : calculer $e=a^b \bmod c$

e.g. calculer $1345667897888^{7668899998880998766} \bmod 87546789998776$

1ère méthode :

Calculer $d=a^b$ puis $e=d \bmod c$

Problème.

Considérer $a=b=10^{20}$

Stocker $a^b=10^{2\times10^{21}}$ nécessite $\approx$ 100 milliards de Go

Qu'en est-il pour des nombres cryptographiques…..?

Exponentiation modulaire

2nde méthode :

Entrées: a, $b=(b_n \ldots b_0)_2$, c

e=1

Pour k=n à 0

e=e^2 mod c

if b_k=1

e=e*a mod c

Complexité ?

Exponentiation modulaire

2nde méthode :

Entrées: a, $b=(b_n \ldots b_0)_2$,c

a=a mod c

e=1

Pour k=n à 0

  $e=e^2$ mod c

  if $b_k=1$

    e=e*a mod c

Complexité ? $O(\log^2 c \log b + \log a \log c)$

RSA

Primalité/factorisation

- Résultats d'arithmétique modulaire permettent de tester efficacement si un entier est premier
  - test de Fermat, Rabin, AKS (2002)
  - algorithmes polynomiaux
- Il n'existe pas d'algorithme polynomiaux pour factoriser de grands entiers (mais pas NP-complet)

Facile ou difficile ?

- Test de primalité
- Générer deux nombres premiers p et q
- Calculer n=pq
- Factoriser des grands nombres
- Calculer $\phi(n)$
- Trouver l'inverse de $x \in Z_n$

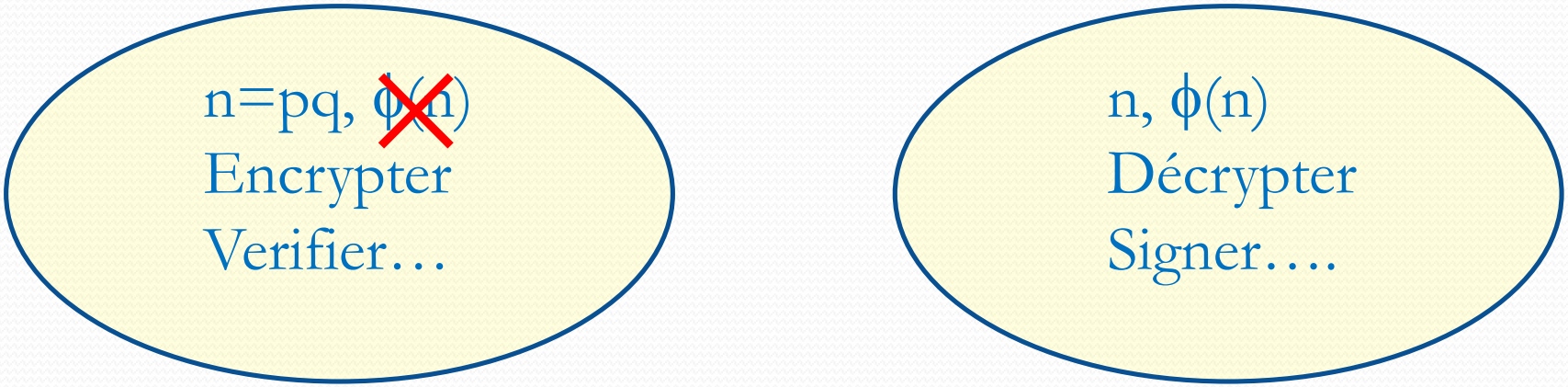
Idées sous-jacentes

- Soit n= pq produit (publique) de deux nombres premiers
- p et q sont tenus secrets

Idées

- On ne connait pas d'algorithme A rapide tel que $\phi(n) \leftarrow A(n)$
- Grâce au théorème de Fermat, connaitre $\phi(n)$ permet de faire des calculs que l'on ne peut pas faire sans.
  - e.g. décrypter

Idées sous-jacentes



Public

Privé

RSA

- KeyGen(λ) : générer $n=pq$ où p et q sont des nombres premiers de δ bits choisis aléatoirement. Soit e tq $pgcd(e, \phi(n))=1$
  - pk = (n,e)
  - sk $=d=e^{-1} \bmod \phi(n)$ // *$d \in \{1,\ldots, \phi(n)-1\}$ et $ed \bmod \phi(n)=1$*
- Encrypt(pk,x) = $x^e \bmod n$
- Decrypt (sk,c) = $c^d \bmod n$

Remarques.

1 - δ est choisi tq meilleure attaque requiert au moins 2^λ opérations élémentaires.

2 - La condition $pgcd(e, \phi(n))=1$ garantit que e a un inverse.

Encrypter en pratique

- Transformer un texte en une suite de nombres
  - Par exemple chaque lettre peut se représenter par un code formé de deux chiffres :

    « mot » → **131520**

  - Utiliser code ASCII

- Encrypter chacun des nombres

RSA est-il correct?

$(x^e)^d = x^{ed} = x^{1+k\phi(n)} = x.\ x^{k\phi(n)} = x.\ (x^{\phi(n)})^k = x.(1)^k = x$

Exercice. Montrer que $(pk_1 = (91;11); sk_1 = 59)$ et $(pk_2 = (91;5); sk_2 = 29)$ sont des clés RSA

RSA est-il correct?

$(x^e)^d = x^{ed} = x^{1+k\phi(n)} = x.\ x^{k\phi(n)} = x.\ (x^{\phi(n)})^k = x.(1)^k = x$

Exercice. Montrer que $(pk_1 = (91;11); sk_1 = 59)$ et $(pk_2 = (91;5); sk_2 = 29)$ sont des clés RSA

Solution.

91=7*13 est bien le produit de 2 nombres premiers. f(91) = 6×12 = 72. Il reste à vérifier que $11 \times 59 \equiv 5 \times 29 \equiv 1 \bmod 72$.

Complexité

- Les fonctions d'encryption et de décryption consistent à calculer une exponentiation modulaire
  - **Complexité ?**

- Grace aux tests de primalité rapide (i.e. test de Fermat) les clé peuvent se générer rapidement.
  - **Comment? Complexité ?**

Securité sens unique

- Pas de preuve formelle.
- Basée sur le problème de la factorisation.
- Si le problème de la factorisation n'est pas difficile alors RSA n'est pas sur.
- La réciproque est est-elle vraie???
- On va prouver que retrouver la clé privée d à partir de la clé publique (n,e) est un problème difficile si la factorisation est un problème difficile

Difficulté de retrouver *sk* pour un attaquant

- Soit A(n,e) un algorithme capable de retrouver d.
- Nous allons construire un algorithme *de factorisation* B(n), utilisant A, qui soit efficace si A l'est (efficace)
  - B s'appelle une réduction
- Si l'on fait l'hypothèse qu'il n'existe pas d'algorithme de factorisation efficace alors **l'algorithme A ne peut pas être efficace.**
  - Si l'on arrive à construire B on dit que **retrouver d est aussi difficile que de factoriser n.**

Construction de B

B(n)

d ← A(n)

pour k=1 à 2

$\phi \leftarrow (3d-1)/k$

$S \leftarrow n-\phi+1$

$p \leftarrow (S+(S^2-4n)^{1/2})/2$ *//p solution de $x^2-Sx+n=0$*

si p|n **alors retourner** p

retourner ⊥

Montrer que B est efficace si A est efficace et que B(n) retourne un facteur de n.

B(n) retourne un facteur de n

preuve.

On sait que ed mod $\phi(n)=1$

$\Rightarrow 3d=1+k\phi(n)$ avec $k\in\{1,2\}$ car $0<d<\phi(n)$

$\Rightarrow \phi(n)=(3d-1)/k$

Donc à l'itération k; $\phi=\phi(n)$

$\Rightarrow$ **S** $=pq-\phi+1=p+q$

Connaissant q+q et pq
$\Rightarrow$ p,q

Or p et q solution de l'équation $x^2-(p+q)x+pq=x^2-Sx+n=0$

Donc la résolution de $x^2-Sx+n=0$ donne p et q

Donc à l'itération k, on retourne bien un facteur de n

Voyante australienne

Une voyante australienne réputée prétend avoir devine les 6 numéros gagnants (compris entre 1 et 49) du prochain tirage du loto. Pour des raisons que nous n'expliciterons pas ici, elle souhaite vous en faire profiter (et seulement vous).

Etant d'un naturel peu partageur, vous souhaitez sécuriser la communication. Pour ceci, vous décidez d'utiliser le cryptosystème RSA en transmettant a cette voyante une clé publique (n,e) de taille 2048 (bits).

Sans consigne de votre part, la voyante vous envoie une encryption de chacun des 6 numéros.

Expliquez pourquoi cette façon de procéder n'est absolument pas sécurisée.

RSA en pratique

RSA est déterministe ⇒ pas de sécurité sémantique

- En pratique on randomise le message avant de l'encrypter, i.e.,
  - On choisit un nombre r aléatoirement (suffisamment grand)
  - On encrypte r||m
- D'autres manières d'ajouter de l'aléatoire existent
  - RSA-OAEP

Quelques attaques RSA

Modification de KeyGen

Pour des besoins cryptographiques non explicités ici, Alice doit générer 2 clés RSA différentes,

i.e. ($pk_1 = (n_1; e_1)$; $sk_1 = d_1$), ($pk_2 = (n_2 , e_2)$; $sk_2 = d$)

Afin de gagner du temps, elle décide de ne générer que 3 nombres premiers p_1,p_2,p_3 (au lieu de 4) et de définir

$$n_1 = p_1 p_2 \text{ et } n_2 = p_2 p_3$$

Pourquoi est ce dangereux?

Modification de KeyGen

Pour des besoins cryptographiques non explicités ici, Alice doit générer 2 clés RSA différentes, i.e. ($pk_1 = (n_1; e_1)$; $sk_1 = d_1$), ($pk_2 = (n_2 , e_2)$; $sk_2 = d$).

Afin de gagner du temps, elle décide de ne générer que 3 nombres premiers p_1,p_2,p_3 (au lieu de 4) et de définir

$$n_1=p_1p_2 \text{ et } n_2=p_2p_3$$

Pourquoi est ce dangereux? …. $p_2=pgcd(n_1,n_2)$

Choisir le même n pour 2 clés différentes

Il est dangereux de chiffrer le même message avec deux couples clé publique/clé secrète (e_1; d_1) et (e_2; d_2) associe chacun a un même module n et de l'envoyer à deux utilisateurs différents.

Pourquoi?

Choisir le même n pour 2 clés différentes

Il est dangereux de chiffrer le même message avec deux couples clé publique/clé secrète (e_1; d_1) et (e_2; d_2) associe chacun a un même module n et de l'envoyer à deux utilisateurs différents.

Pourquoi?

Soit M_1 et M_2 les deux chiffrés, i.e. $M_i = m^{e_i} \bmod n$

Un attaquant écoutant le réseau les connait

Si pgcd(e_1,e_2)=1 alors il existe u et v tel que $ue_1+ve_2=1$

$\Rightarrow (M_1)^u(M_2)^v \bmod n = m^{e1u}m^{e2v} = m^{e1u+e2v} = m$

Et d'autres…

- Choisir d petit, i.e. $d<n^{1/4}$
- Encrypter m et $m+\delta$ avec δ connu
- m et e sont tous les deux petits
- ….

Conclusion

- RSA…a résisté a 40 d'efforts pour le « casser »
  - Le meilleur argument
- RSA peut être considéré comme sûr si on ne prend pas de liberté par rapport :
  - aux fonctions KeyGen, Encrypt, Decrypt
  - aux préconisations des instances compétentes
- RSA ne survivra pas à l'informatique quantique

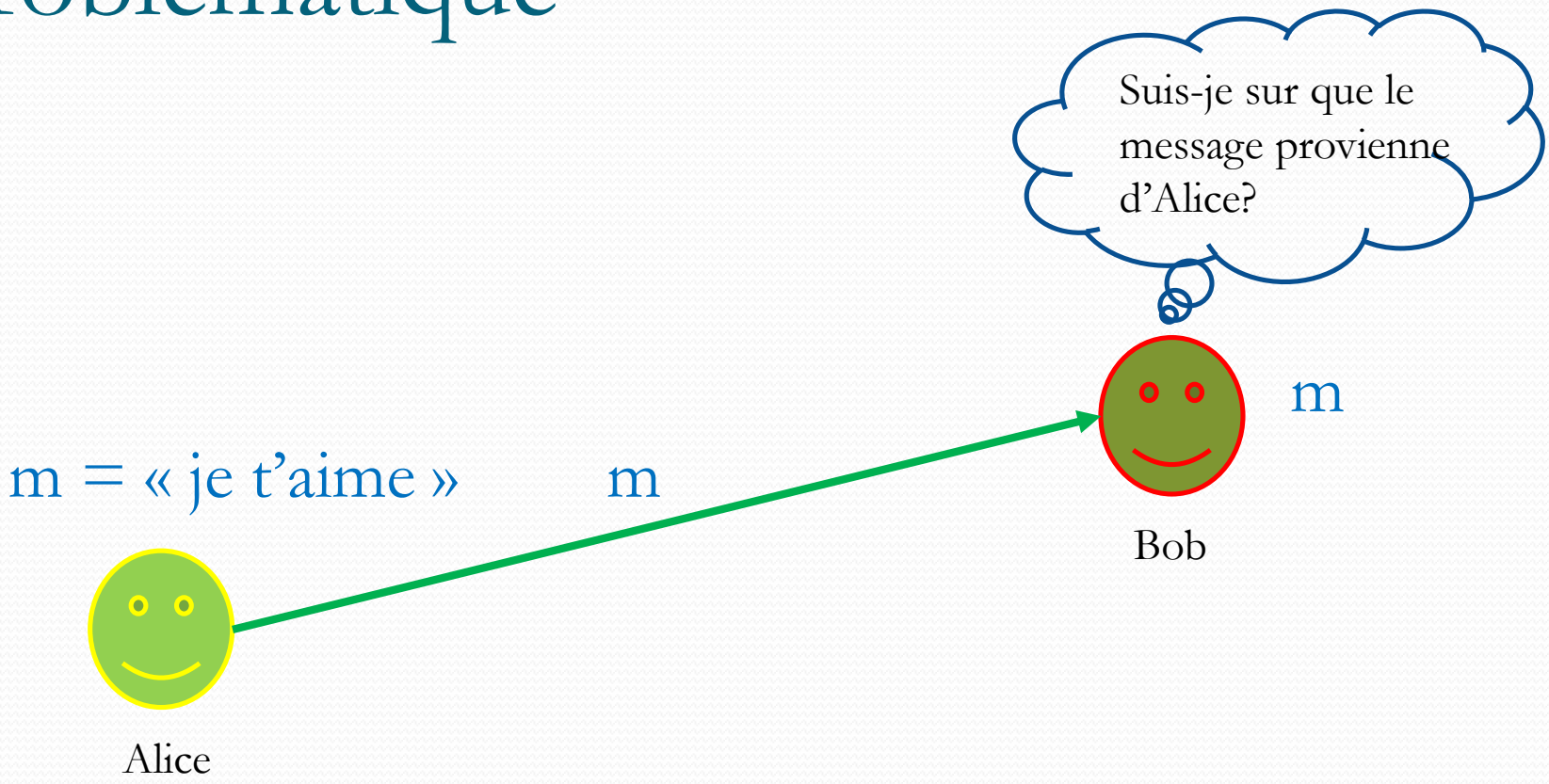
Signature Numérique

signature numérique

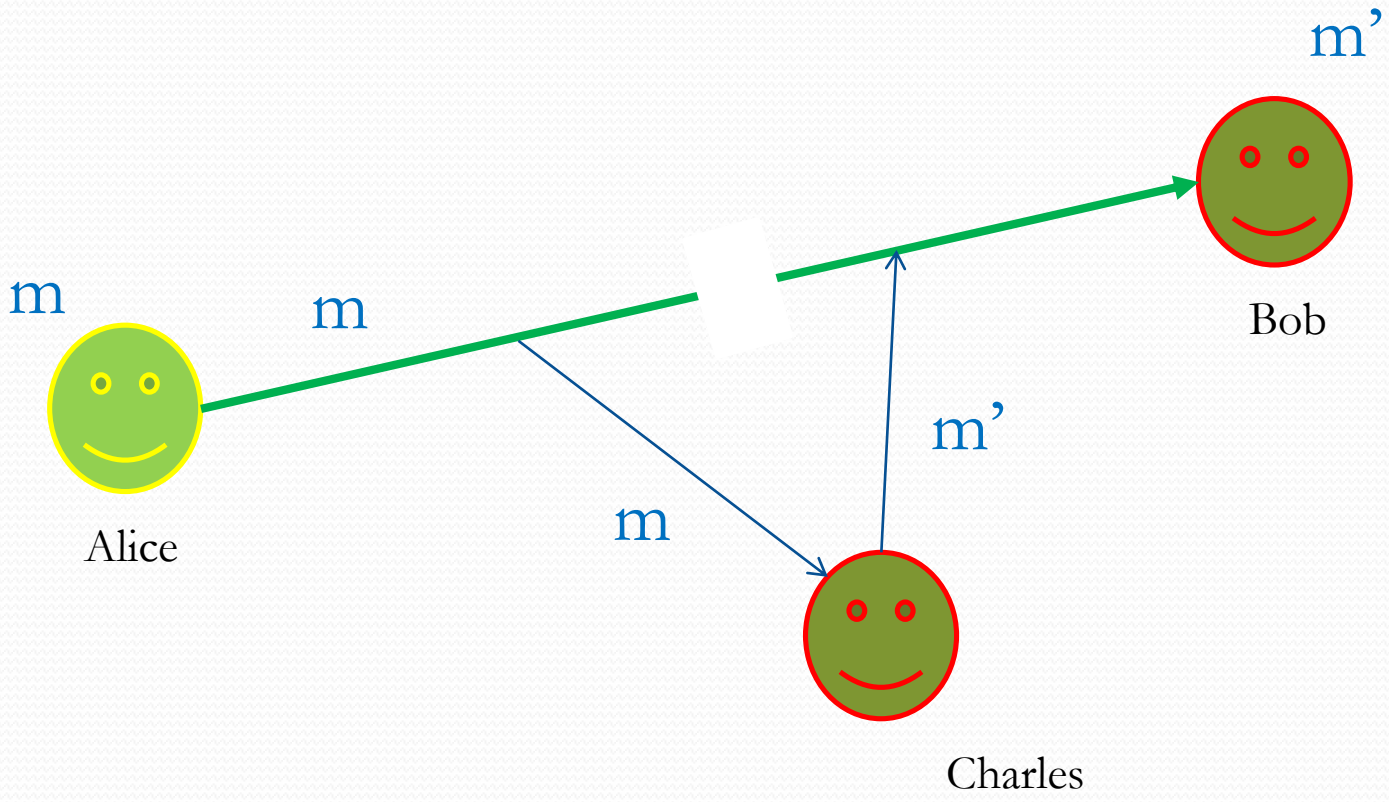
Vous souhaitez envoyer un message tel que le destinataire soit certain que ce message provient de vous….

Une solution?

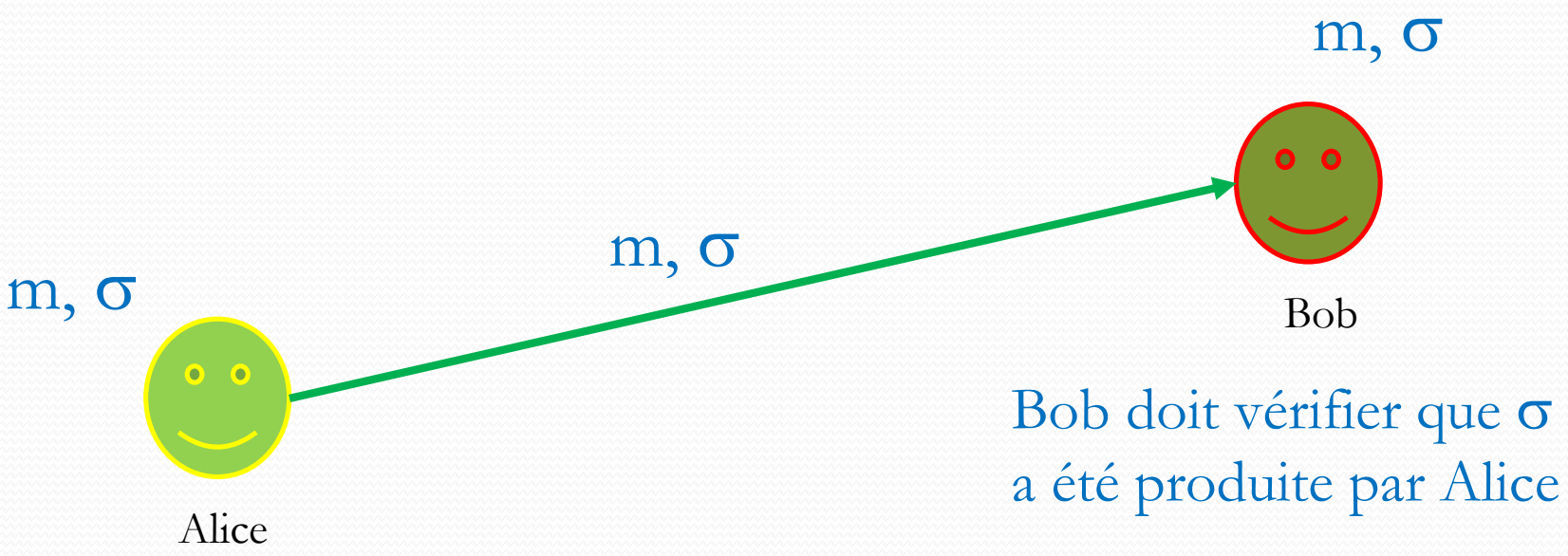
Problématique



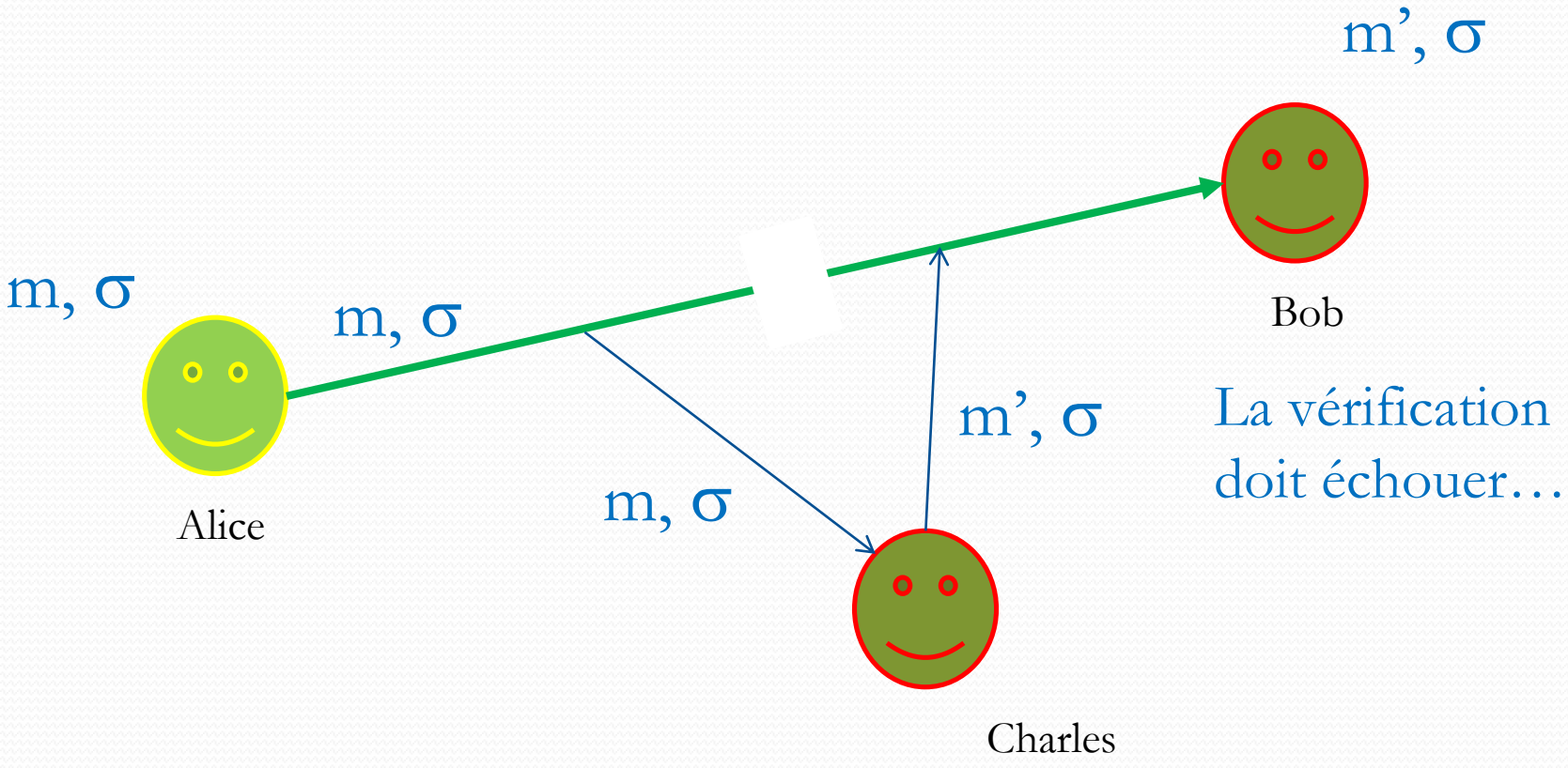
Problématique



Idée : adjoindre une signature

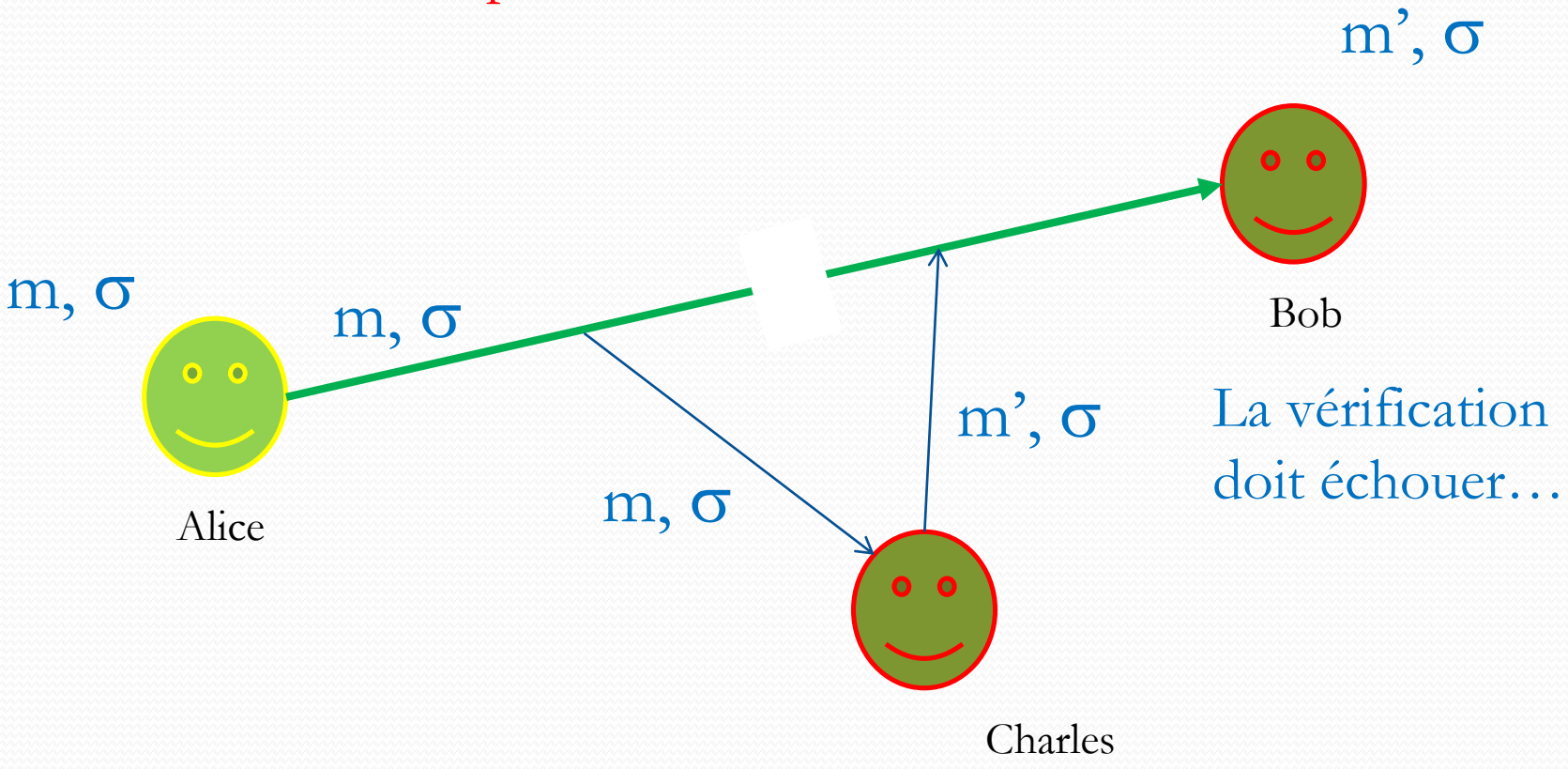


Idée : adjoindre une signature



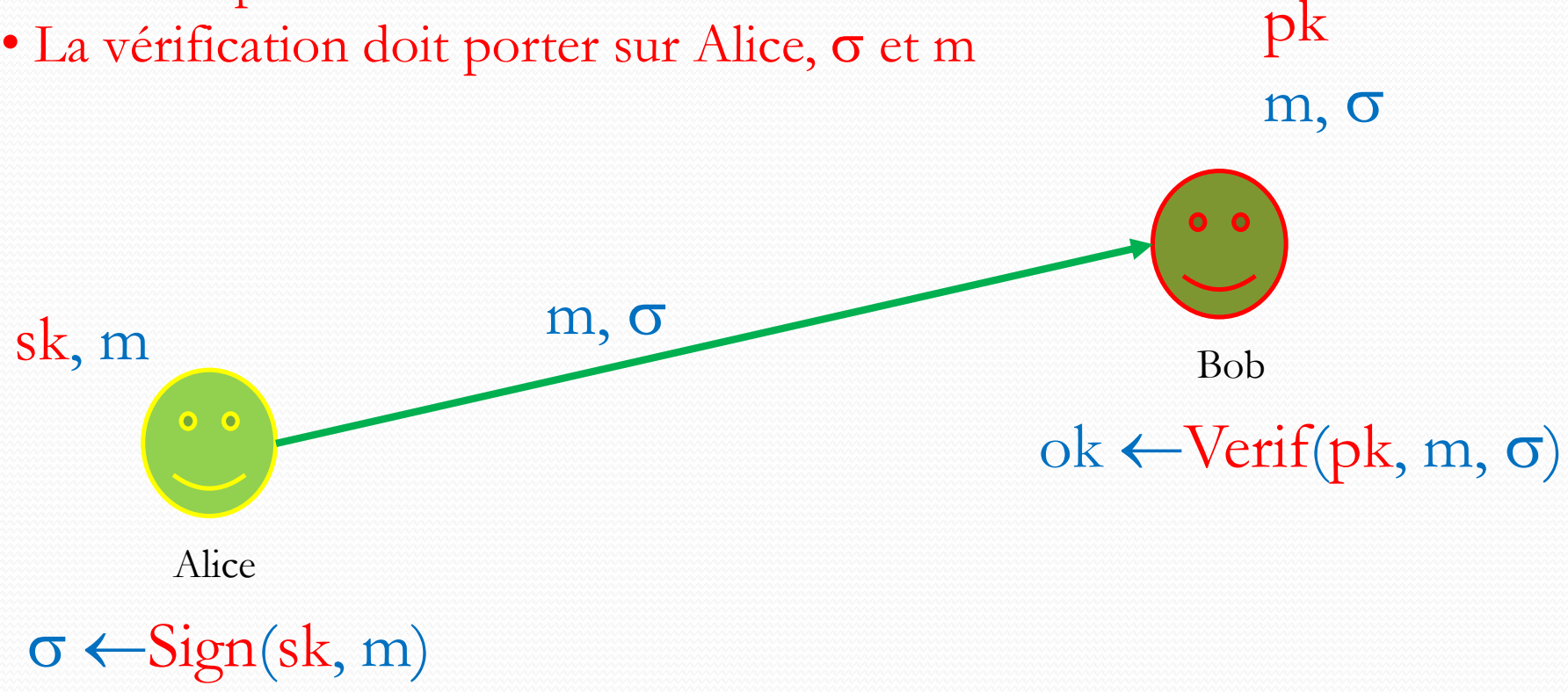
Premières conclusions :

- σ doit dépendre de Alice et de m
- La vérification doit porter sur σ et m



Premières conclusions :

- σ doit dépendre d'Alice et de m
- La vérification doit porter sur Alice, σ et m



Principe Général

Munissons nous d'un ensemble I_m de messages m, d'un ensemble fini I_s de signatures σ, d'un ensemble $I \subset I_{pk} \times I_{sk}$ de (couples) clés (pk,sk) valides.

Définition. Un procédé de signature est la donnée pour chaque $(pk,sk) \in I_{pk} \times I_{sk}$ de deux fonctions calculables en temps polynomial :

- Sign : $I_{pk} \times I_m \rightarrow I_s$: fonction de signature (secrète)
- Verif: $I_{sk} \times I_m \times I_s \rightarrow \{vrai , faux\}$: la fonction de vérification (publique) telles que si $(pk,sk) \in I$:

$$Verif(pk, m, \sigma) = vrai \Leftrightarrow \sigma = Sign(sk,m)$$

Remarque

Un procédé de signature n'est jamais inconditionnellement sûr :

Alice désirant signer le message m peut essayer une à une toutes les valeurs de $\sigma \in I_s$ et trouver une signature valide grâce à la fonction de vérification.

$\Rightarrow$ Un procédé de signature est toujours adossé à un problème supposé difficile.

Cahier des charges

- **Authentique** : l'identité du signataire doit pouvoir être retrouvée de manière certaine.
- **Infalsifiable** : la signature ne peut pas être falsifiée. Quelqu'un ne peut se faire passer pour un autre.
- **Non réutilisable** : la signature n'est pas réutilisable. Elle fait partie du document signé et ne peut être déplacée sur un autre document.
- **Inaltérable** : un document signé est inaltérable. Une fois qu'il est signé, on ne peut plus le modifier.
- **Irrévocable** : la personne qui a signé ne peut le nier.

Encrypter et signer : problèmes duaux

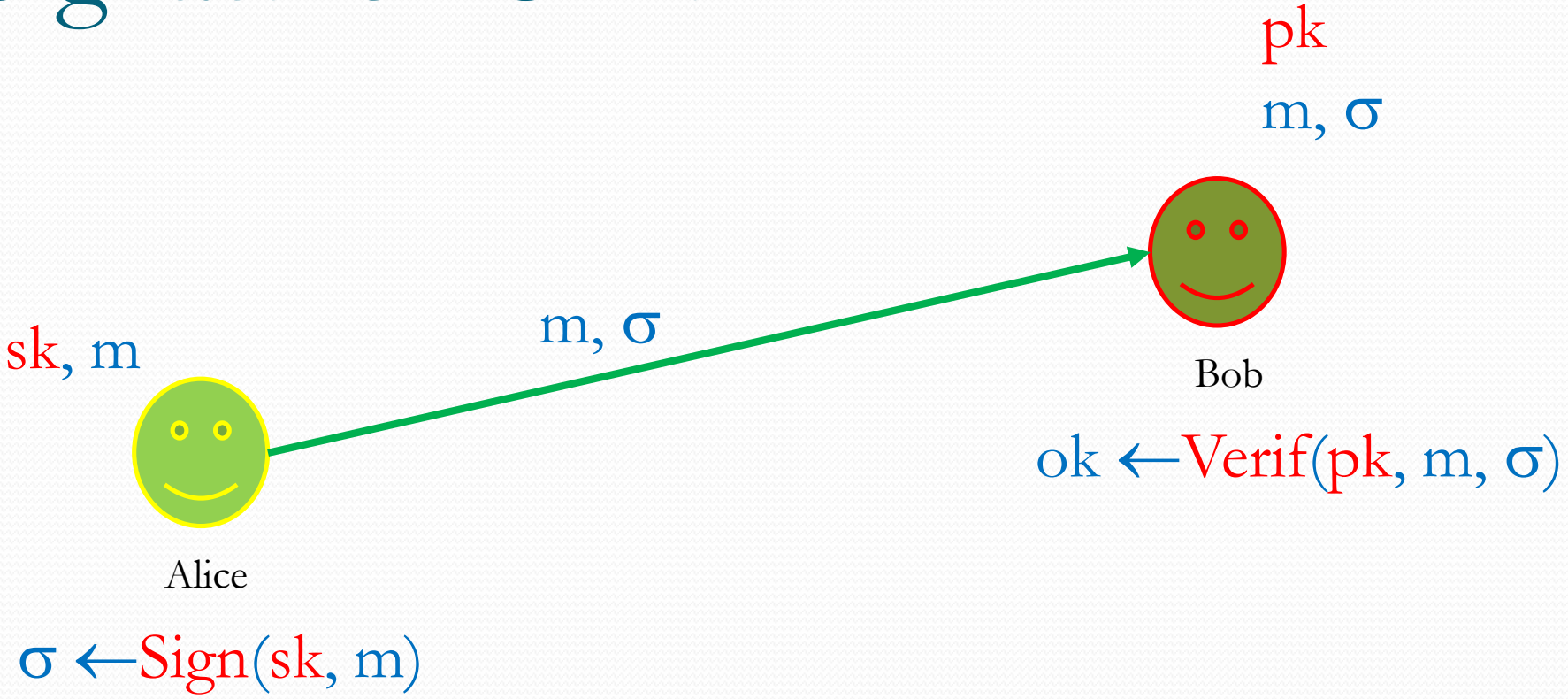
- Encrypter / verifier la signature
  - Tout le monde doit pouvoir le faire
- Décrypter / Signer
  - Réserver au possesseur d'une information secrète

Encrypter et signer : problèmes duaux

- Encrypter / vérifier la signature
  - Tout le monde doit pouvoir le faire
- Décrypter / Signer
  - Réserver au possesseur d'une information secrète

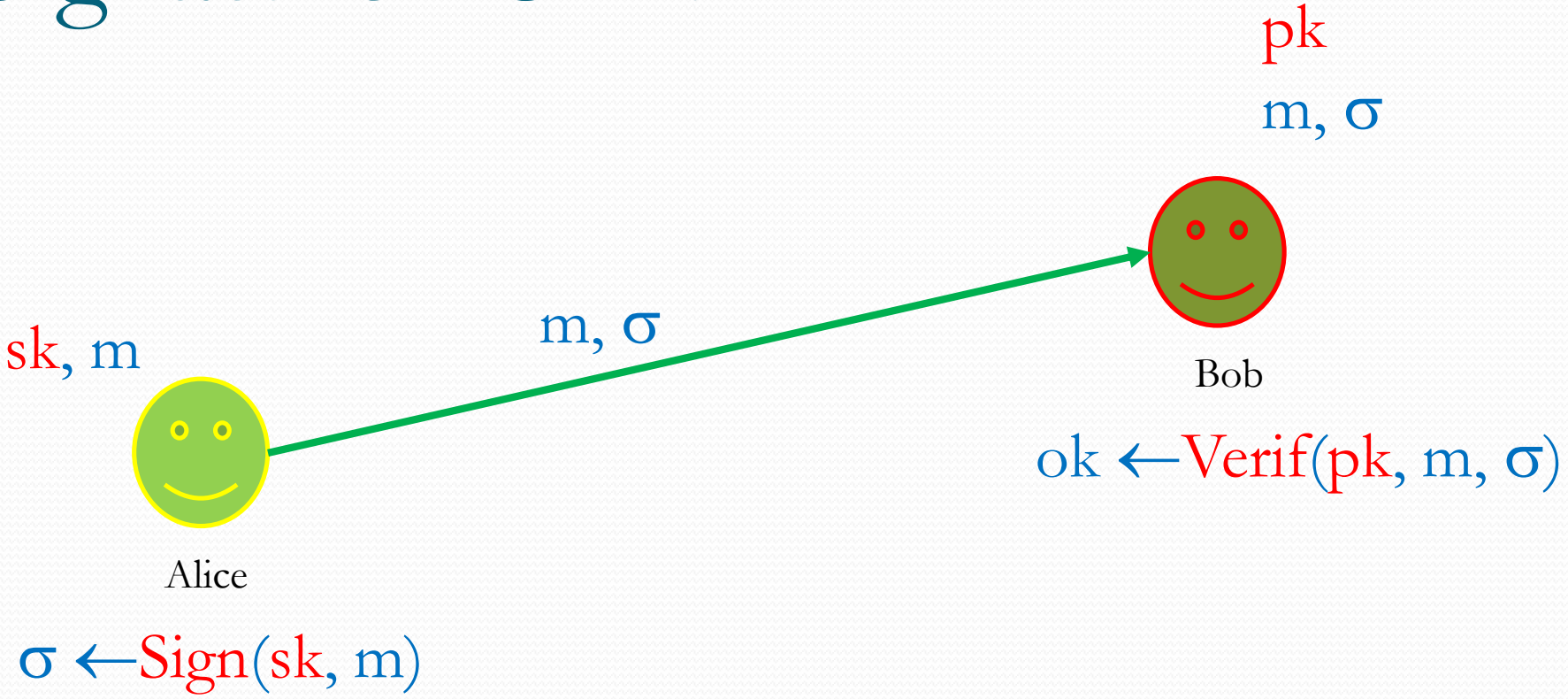
⇒ Certains cryptosystèmes à clé publique peuvent être transformés en schémas de signature.

Signature RSA…



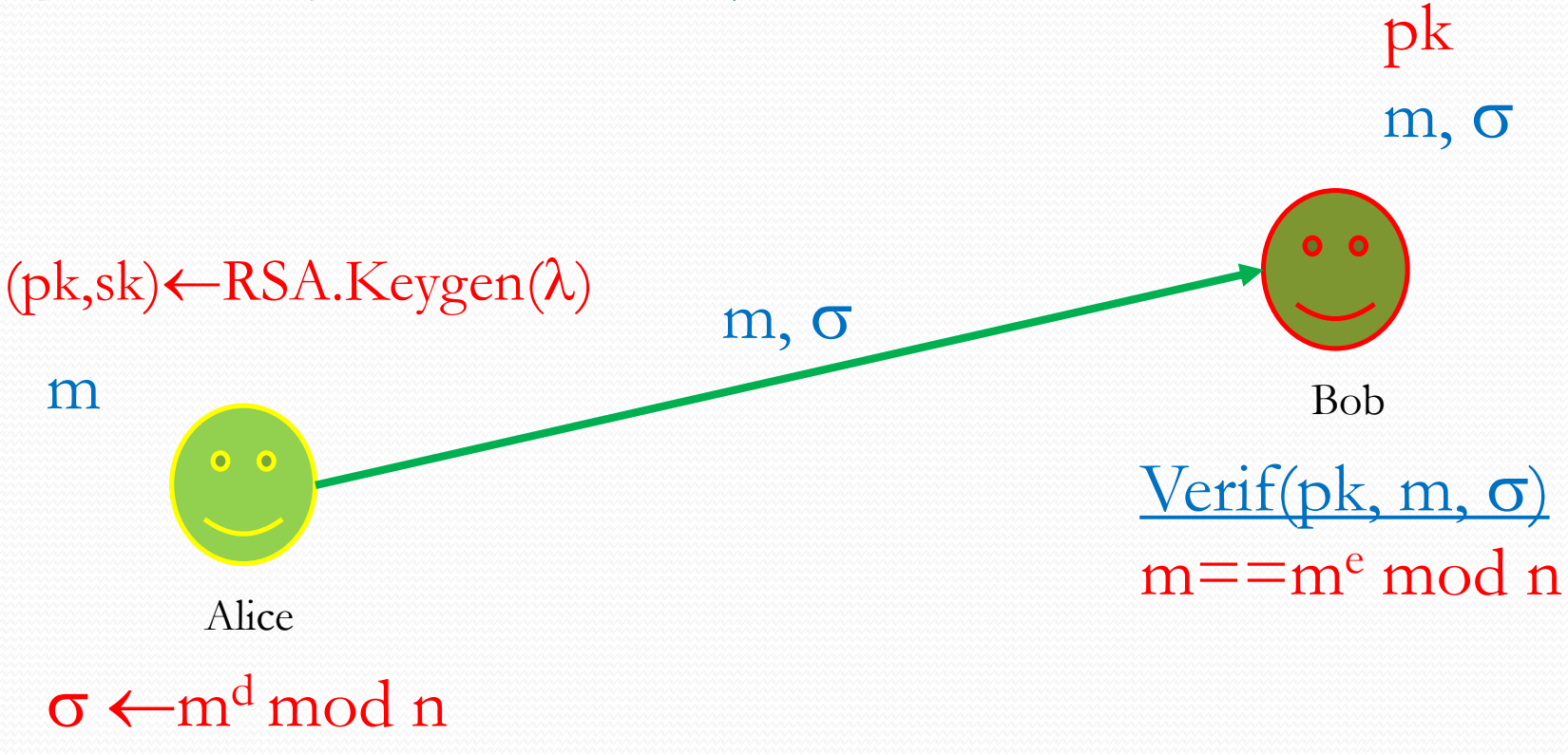
Une idée ?

Signature RSA…

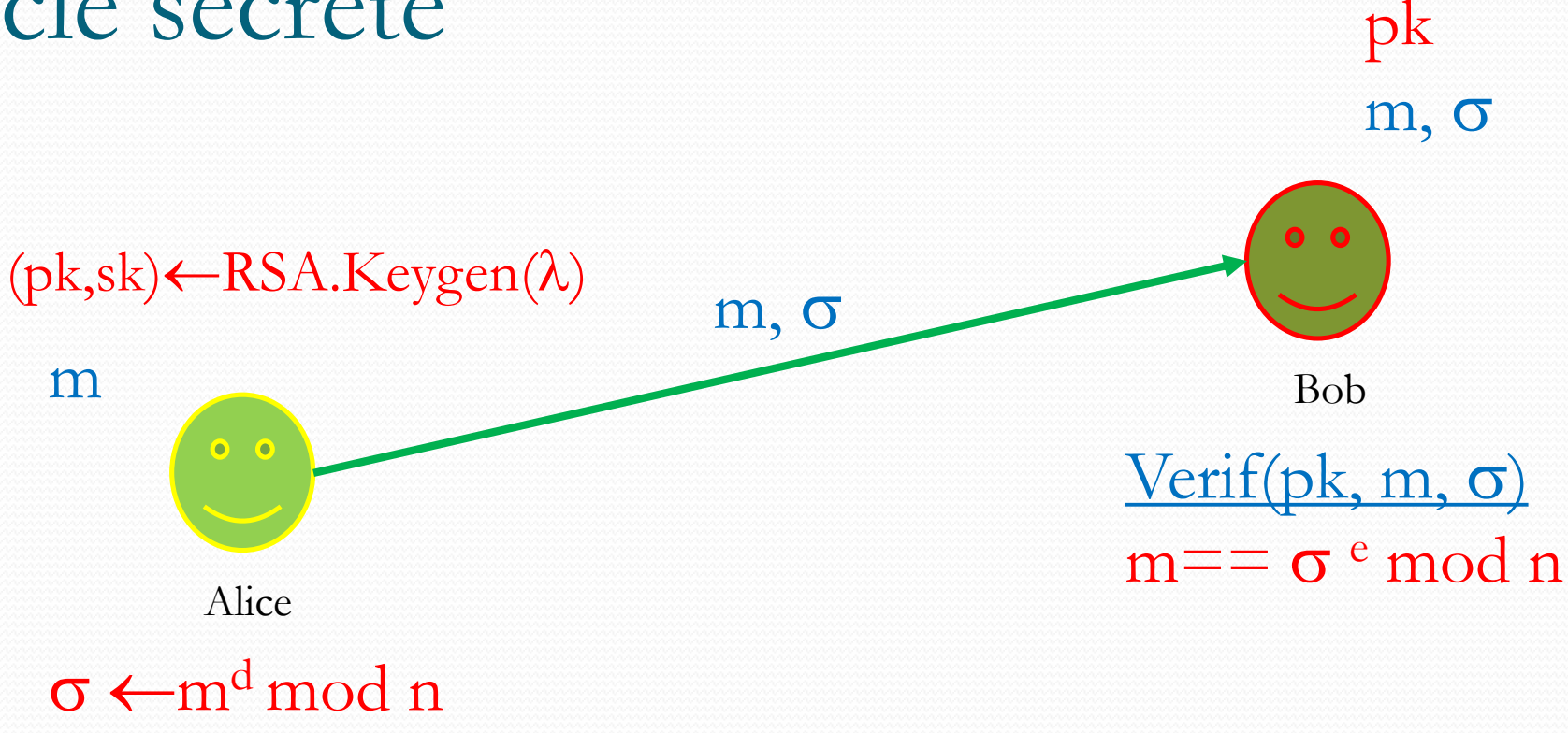


Une idée ?

Une idée…naive

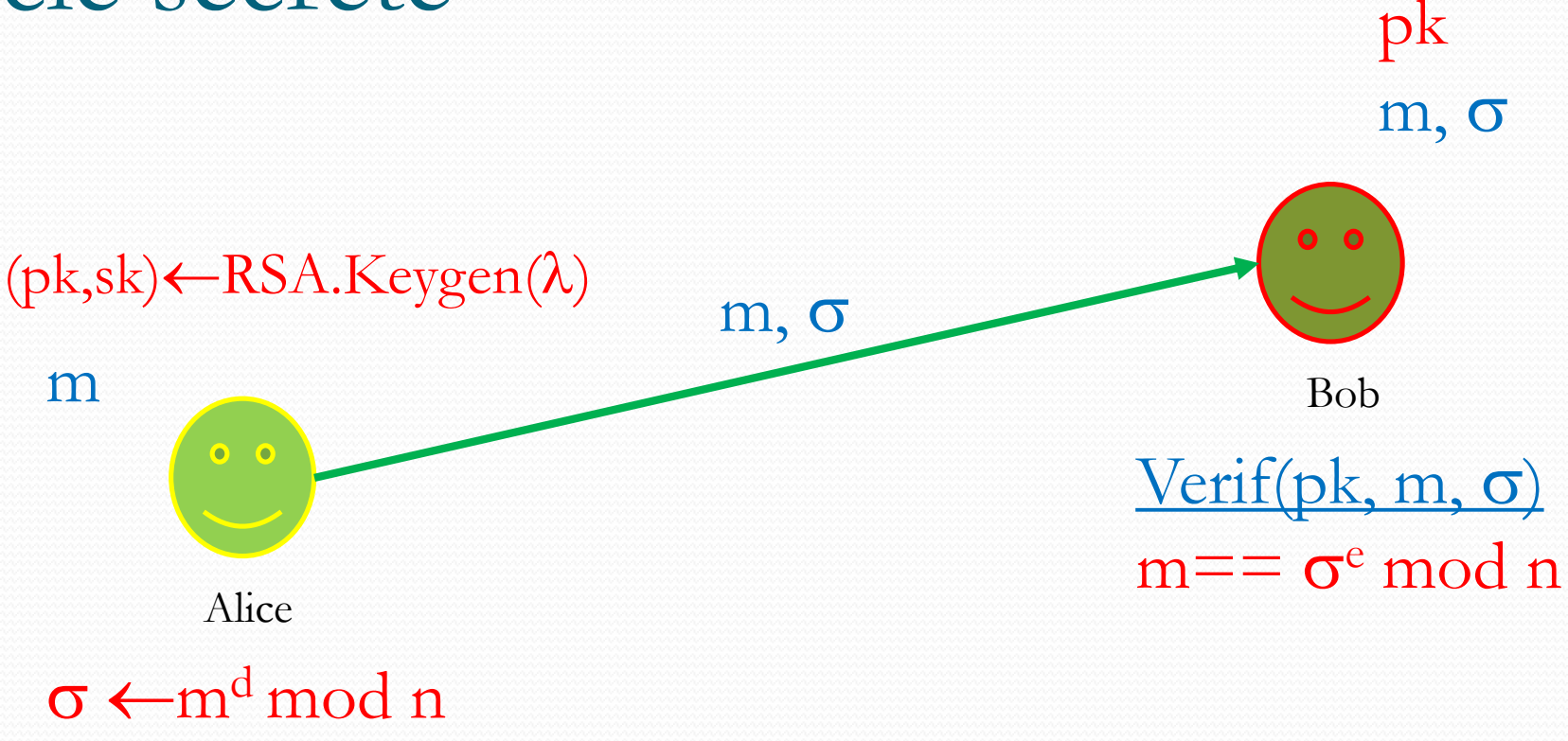


Une idée naive…encrypter avec la clé secrète



Problème. Charles peut produire un couple (m,σ) qui passe la vérification
Comment ?

Une idée naive…encrypter avec la clé secrète



Problème. Charles peut produire un couple (m,σ) qui passe la vérification
Comment ?
Choisir σ aléatoirement dans $\mathbf{Z}_n^*$ et calculer $m= \sigma^e \bmod n$

Hacher le message avant de l'encrypter

$H : \{0,1\}^* \rightarrow \{0,1\}^{t \geq \lambda}$ une fonction de hachage publique

pk

m, σ

$(pk,sk) \leftarrow RSA.Keygen(\lambda)$

m

m, σ

Bob

Alice

Verif(pk, m, σ)

$H(m) == \sigma^e \bmod n$

$\sigma \leftarrow H(m)^d \bmod n$

Signer un message avec RSA

- Soit H une table de hachage publique
- Générer un couple de clés (pk={e,n},sk={d}) avec la fonction *RSA.KeyGen*

Sign (sk,m)

1. générer h=H(m)
2. calculer $\sigma = h^d \bmod n$
3. envoyer (m, σ)

Verif(pk,m,σ)

1. calculer h=H(m)
2. calculer $t = \sigma^e \bmod n$
3. vérifier que h==t

Première attaque

Pour signer le message m Alice calcule $\sigma = \text{Sign}(H(m))$, si Charles réussit à trouver un autre message m_0 tel que

$$H(m_0) = H(m)$$

alors σ sera une signature valide de m_0 !

Définition. Une fonction H est dite *faiblement résistante* aux collisions si étant donné x, il est calculatoirement difficile d'obtenir x_0 tel que $H(x) = H(x_0)$.

Deuxième attaque

Charles cherche deux messages distincts m et m_0 tels que

$$H(m) = H(m_0)$$

Il persuade ensuite Alice de signer le message m et il obtient ainsi une signature valide de m_0.

Définition. Une fonction H est dite *fortement résistante* aux collisions s'il est calculatoirement difficile d'obtenir x et x' distincts tel que H(x) = H(x').

Troisième attaque

- Soit z choisi aléatoirement.
- Supposons que Charles soit capable de trouver m tel que H(m)=z.
- Alors Charles peut signer m (sans aucun contrôle sur m).

Comment ?

Troisième attaque

L'attaquant Charles:

- Choisit σ aléatoirement
- Calcule $z = \sigma^e \bmod n$
- z est distribué aléatoirement dans $\mathbf{Z}_n^*$ donc par hypothèse, Charles peut trouver m tel que H(m)=z
- Comme Verif(pk, m, σ)=true, Charles a trouver un message m et une signature valide σ

Par conséquent, la fonction H doit être à sens unique.

Fonction hachage cryptographique

- Une fonction de hachage cryptographique H doit offrir une résistance forte aux collisions :
  - $\Rightarrow$ résistance faible aux collisions
  - $\Rightarrow$ sens unique
- SHA2 est une fonction de Hachage classiquement utilisée
  - Il est admis que la signature RSA-SHA2 est sûre
  - La sécurité est prouvée dans des modèles idéalisés (random oracle model)

Fonction hachage cryptographique prouvée résistante aux collisions

- Soit p un nombre premier tel que q = (p-1)/2 soit également premier. Soit α et β deux éléments primitifs modulo p.
- On suppose qu.il est difficile de calculer $\log_\alpha(\beta)$: la valeur x telle que $\alpha^x = \beta \bmod p$.
- On définit la fonction de Chaum-Van Heist-Pfitzmann par

$$h : \{0,\ldots,q-1\}^2 \rightarrow Z_p^*$$
$$h(x,y) = x^\alpha y^\beta \bmod p$$

Proposition. Toute collision dans la fonction ci-dessus permet de calculer $\log_\alpha(\beta)$.

Preuve

Démonstration. Suppososns que l'on a une collision, c'est à dire deux couples distincts (x, y) et (x', y') tels que $\alpha^x \beta^y = \alpha^{x'} \beta^{y'}$ dans $\mathbb{Z}/p\mathbb{Z}$. On obtient $\alpha^{x-x'} = \beta^{y'-y}$. Soit $d = \text{pgcd}(p-1, y'-y)$ comme $p-1 = 2q$ avec q premier et que $y - y' < q$ on a $d = 1$ ou $d = 2$.

- Si $d = 1$, on calcule z : l'inverse de $y - y'$ modulo $p-1$ et $\alpha^{(x-x')z} = \beta$ donc $\log_\alpha(\beta) = (x - x')\, z\, [p-1]$.
- Si $d = 2$, on calcule z' : l'inverse de $y - y'$ modulo q donc $(y - y')\, z' = 1 + kq$ on a alors $\alpha^{(x-x')z'} = \beta \times (\beta^q)^k = \pm\beta$ car $\beta^q = -1$.

  $\log_\alpha(\beta) = (x - x')\, z\, [p-1]$ ou $\log_\alpha(\beta) = (x - x')\, z + q\, [p-1]$ car $\alpha^q = -1\, [p]$. Il est ensuite facile de vérifier pour trouver la bonne valeur de $\log_\alpha(\beta)$.

En pratique…

On vient de montrer que la fonction ci dessus est fortement résistante aux collisions mais :

- Elle est « trop lente » pour être performante en pratique.
- En pratique on utilise des fonctions de hachage très rapide à calculer mais dont la solidité n'est pas démontrée, comme SHA-2.
- Une autre fonction de hachage réputée : MD5 a été brisée : des cryptanalystes ont trouvé un moyen de générer des collisions en une heure.

Sécurisation des communications : Protocole **https**

https = httpsécurisé

Idée. Cryptographie symétrique bcp plus rapide que cryptographie asymétrique

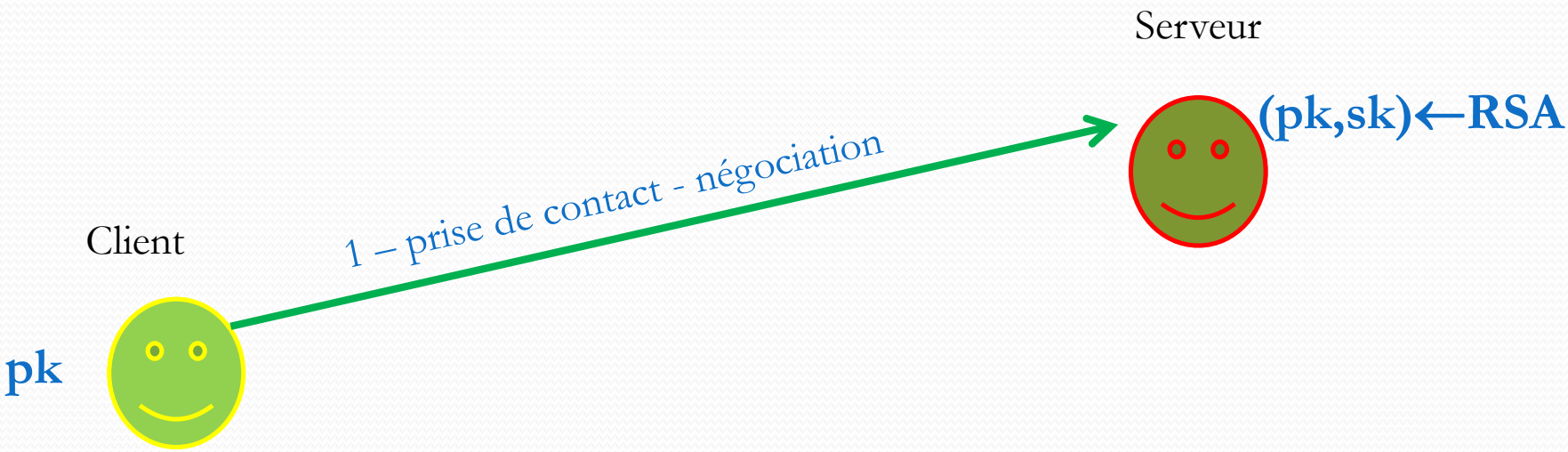
- Echange d'une clé secrète k avec cryptographie asymétrique
  - SSL 1.0 (1991) premier protocole proposé par netscape
  - TLS 1.3 (2018)
- Utiliser k pour échanger information de manière sécurisée
  - e.g. AES pour la confidentialité
  - HMAC pour l'authentification

Principe des protocoles SSL et TLS

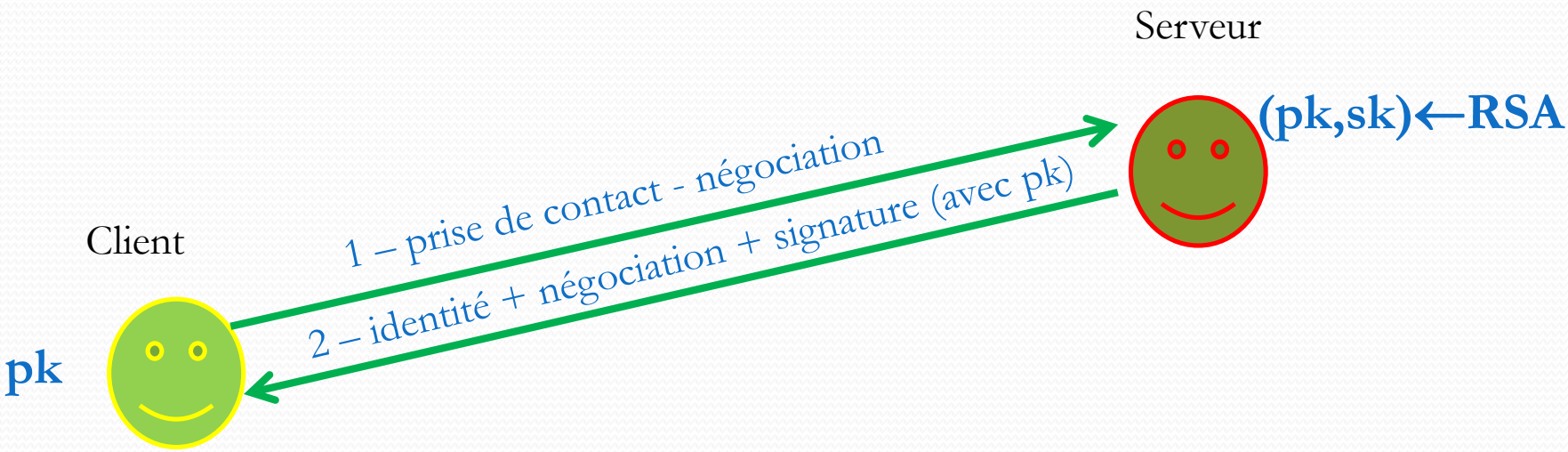
Permet

- d'authentifier un serveur (et non le client)
  - Un attaquant actif ne pourra pas se faire passer pour le serveur
- de sécuriser la communication ensuite
  - L'authentification du client pourra se faire via un mot de passe

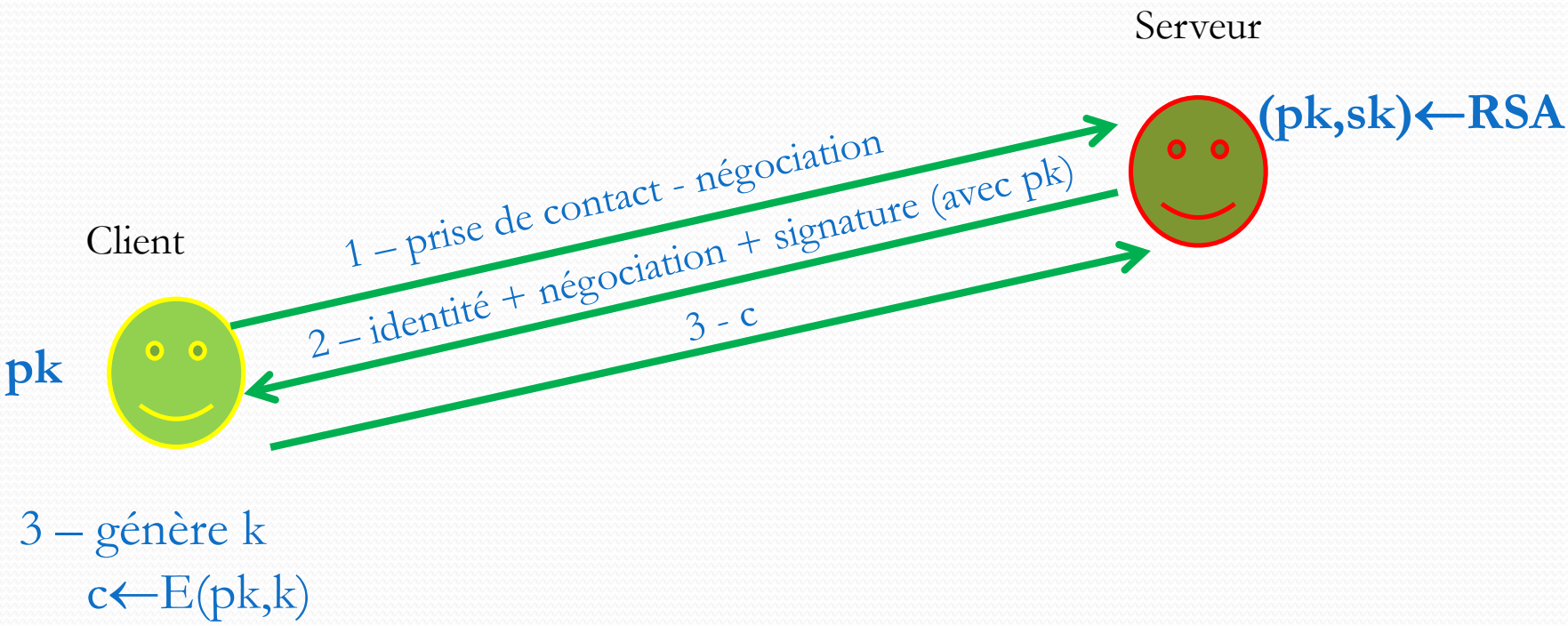
Principe des protocoles SSL et TLS



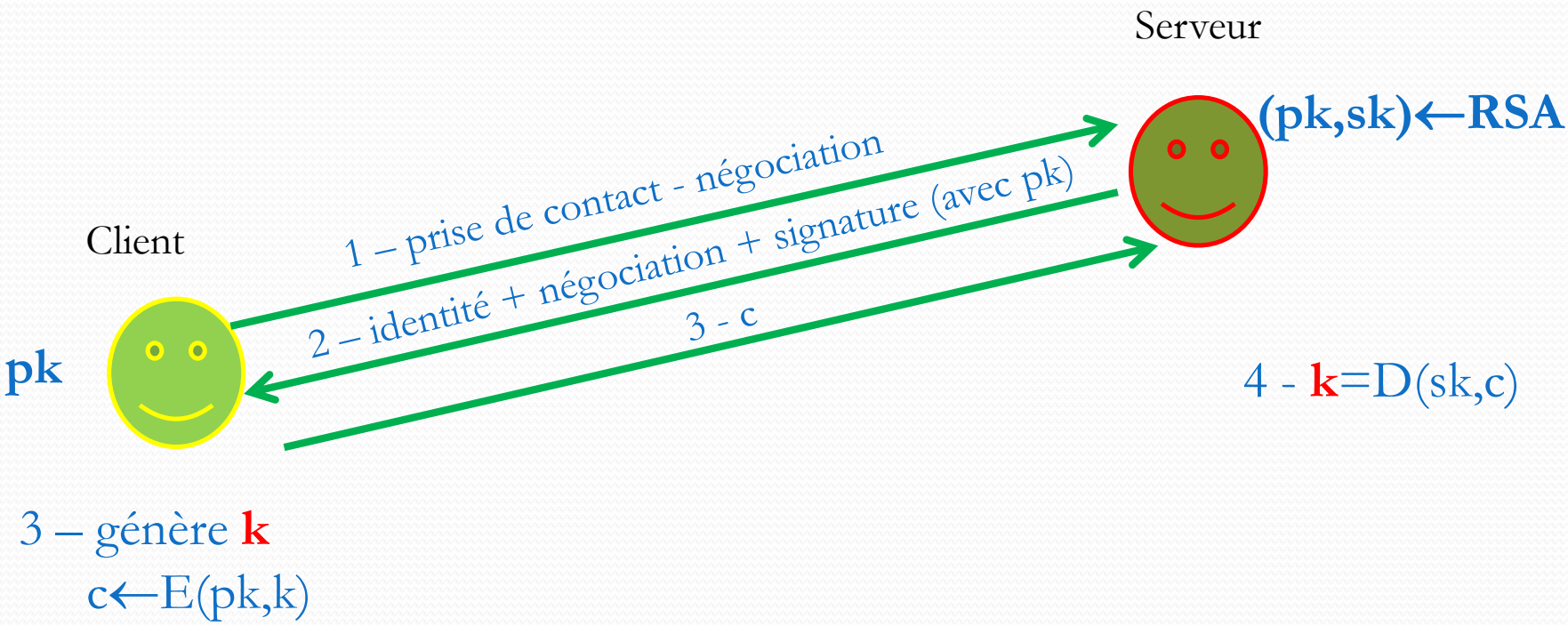
Principe des protocoles SSL et TLS



Principe des protocoles SSL et TLS



Principe des protocoles SSL et TLS



….puis communiquent avec k :

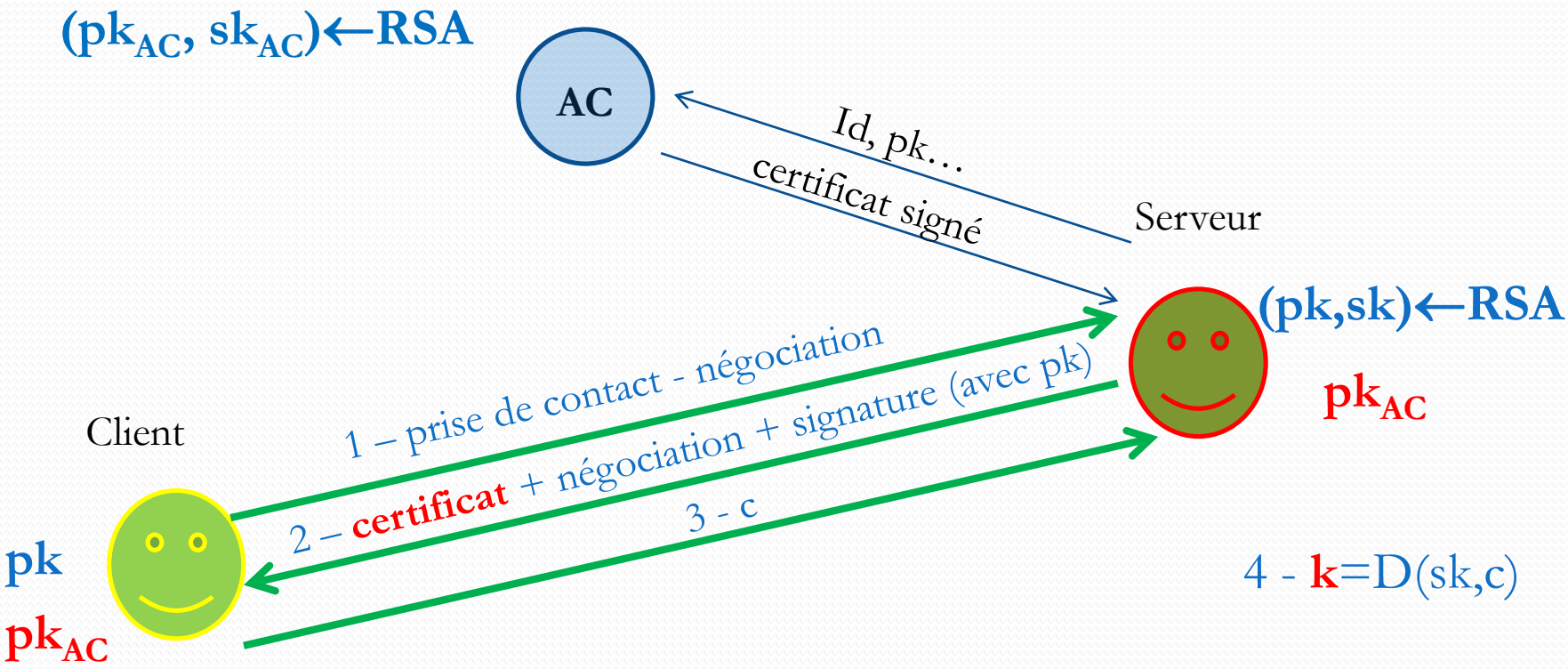
- **chiffrement symétrique, e.g. AES**
- **mécanismes authentification HMAC (non-étudiés ici)**

Infrastructure à clé publique

- Le client est sûr de communiquer avec le propriétaire de la clé publique pk….mais est-il sur qu'elle appartient au serveur ?
  - Non…elle peut appartenir à l'homme du milieu
- Besoin d'une autorité externe pour faire le lien pk / identité serveur
- Infrastructure à clé publique (PKI)
  - Signe des certificats avec une clé publique

Ce sera détaillé lors de la seconde partie de ce cours

Principe des protocoles SSL et TLS



3 – vérifie signature avec **pk** et certificat avec pk_{AC}
génère **k**
c←E(pk,k)

Exemple certificat

Certificat

*.univ-lyon1.fr | GEANT OV RSA CA 4 | USERTrust RSA Certification Authority

Nom du sujet

Pays	FR
État / Province	Auvergne-Rhône-Alpes
Organisation	UNIVERSITE CLAUDE BERNARD LYON 1
Unité organisationnelle	DSI
Nom courant	*.univ-lyon1.fr

Nom de l'émetteur

Pays	NL
Organisation	GEANT Vereniging
Nom courant	GEANT OV RSA CA 4

Validité

Pas avant	Mon, 27 Jun 2022 00:00:00 GMT
Pas après	Tue, 27 Jun 2023 23:59:59 GMT

Exemple certificat (suite)

Noms alternatifs du sujet	
Nom DNS	*.univ-lyon1.fr
Nom DNS	univ-lyon1.fr

Informations sur la clé publique	
Algorithme	RSA
Taille de la clé	2048
Exposant	65537
Module	E3:7B:EE:67:3E:E4:21:45:B3:04:24:FA:C8:2C:72:11:93:9C:D6:4F:BA:EF:AB:44:9E:75:6...

Divers	
Numéro de série	69:CE:FB:D6:A1:AB:DD:07:B8:F0:15:B2:5B:D1:B0:EF
lgorithme de signature	SHA-384 with RSA Encryption
Version	3
Télécharger	PEM (cert) PEM (chain)

Bonnes pratiques

- Le protocole https **n'analyse pas** l'identité
  - On peut imaginer un site qui s'appellerait **https://impots.gou.com**
  - dont l 'identité serait « escroquerie »
- Avant de fournir des données confidentielles
  - **Vérifier** *à la main* **l'identité fournie** par le certificat
  - IA pourrait le faire à votre place

Que peut faire l'homme du milieu ?

Que peut faire l'homme du milieu ?

Perturber :

- la diffusion de pk_{AC}
  - la/les clés pk_{AC} sont souvent importées lors de l'installation du navigateur
- s'approprier un faux certificat
  - L'autorité de certification doit vérifier l'identité et les informations du serveur

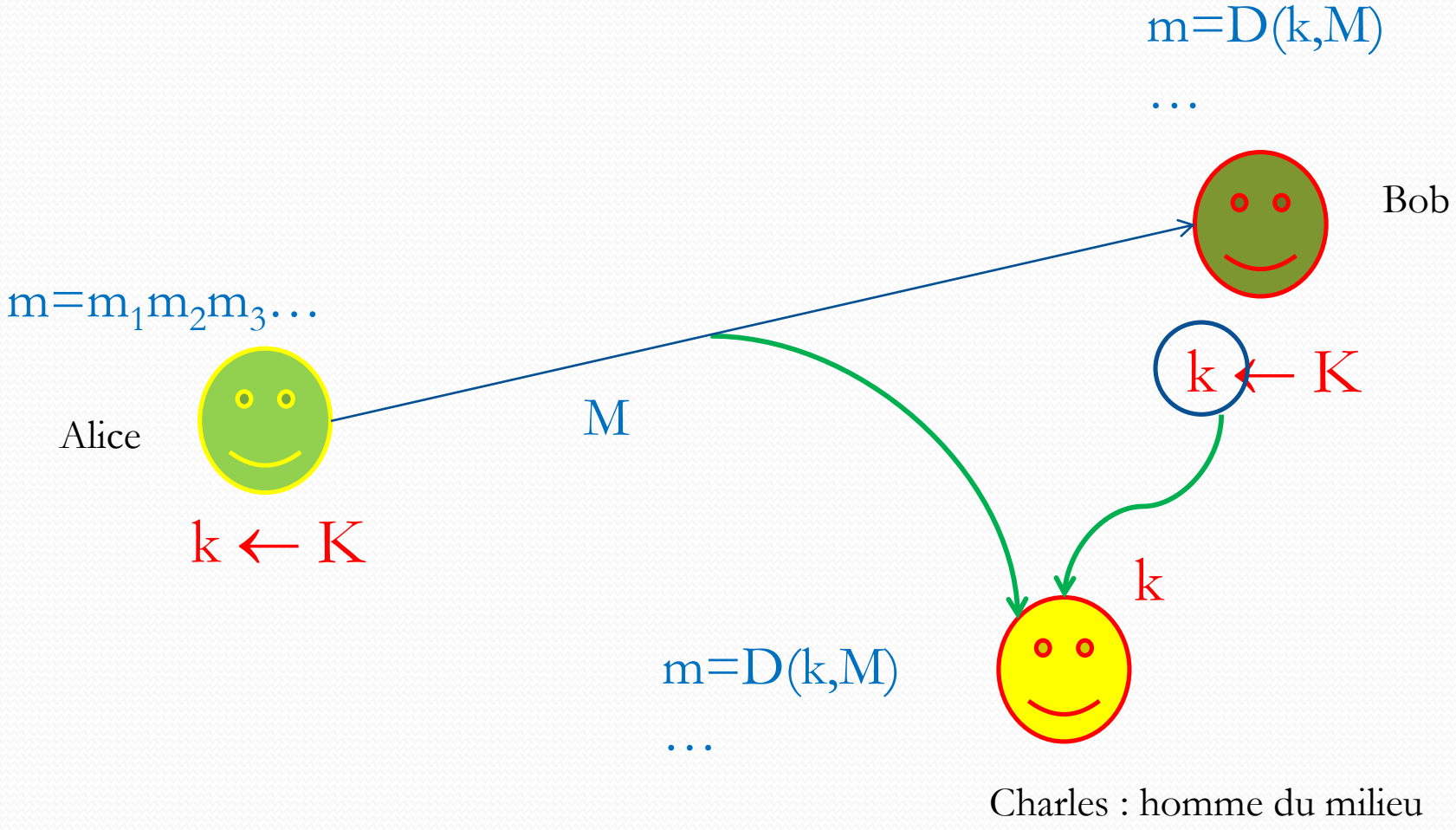
Authentification client

- Dans sa version de base https identifie le serveur
- Il peut être adapté pour identifier aussi le client
  - Communication entre instances officielles
- Généralement le client est authentifié par mot de passe
  - https permet de garantir la confidentialité du mot de passe
  - Le mot de passe n'est généralement pas stocké en clair sur le serveur…juste une empreinte

Limites

- Homme du milieu peut encore perturber communication client/PKI
  - Le client n'a pas la bonne clé publique de l'autorité de certification
- Navigateurs clients pas à jour ou mal configurés
- Aucune mesure contre des attaquants intrusifs (qui peuvent s'introduire sur votre pc)

Crypto ne peut rien contre les intrusions…



Sécurité informatique > cryptographie

L'attaquant peut récupérer la clé k par intrusion…que faire?

- La cryptographie ne peut rien!!!…nécessité de supposer que l'on peut garder un secret en local

Sécurité informatique plus vaste que la cryptographie

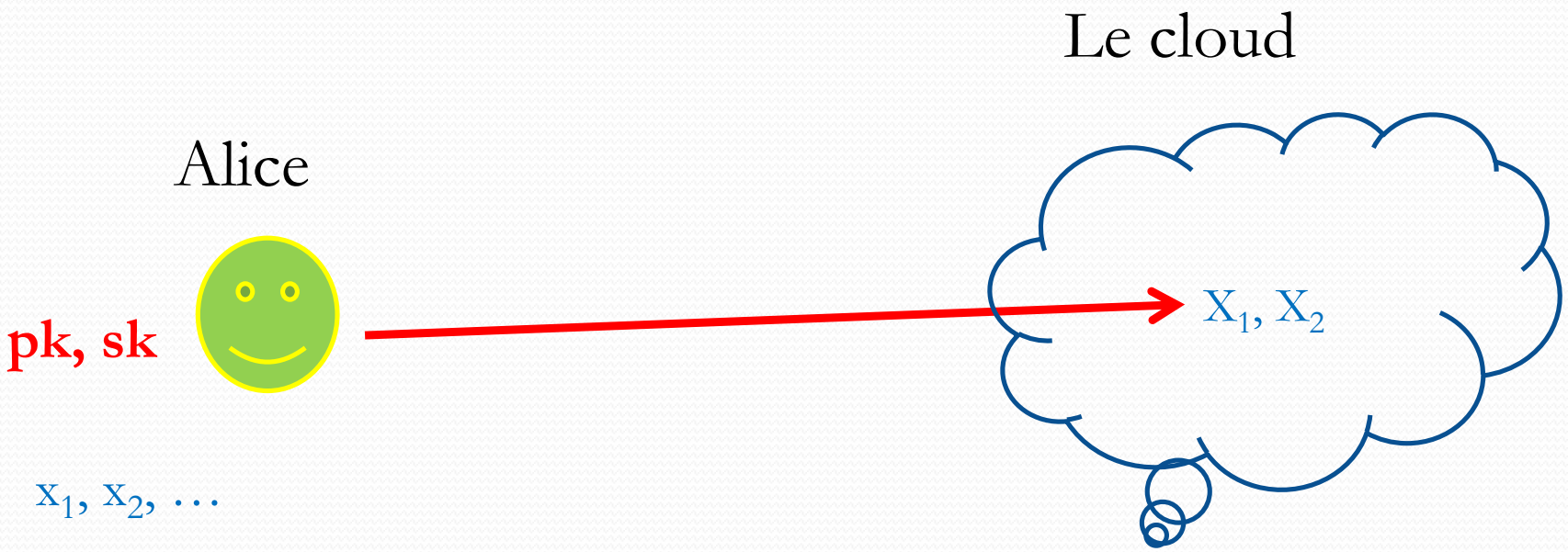
Partie 3

Calcul multi-parties sécurisé

Principe général

- Considérons t parties $p_1, p_2, \ldots, p_t$.
- **Les canaux de communication** sont supposés **sécurisés**
  - e.g. avec https
- Chaque partie p_i
  - a une valeur secrète x_i
  - souhaite obtenir $\mathbf{f_i(x_1,\ldots, x_t)}$
  - souhaite garder $\mathbf{x_i}$ **secret**

Cryptographie et cloud



Externaliser les requètes ???

- Modèle précédent permet d'externaliser les données
- Mais pour les traiter, il faut tout d'abord les télécharger, les décrypter
- Serait-il possible d'externaliser aussi les traitements?

Externaliser les requètes ???

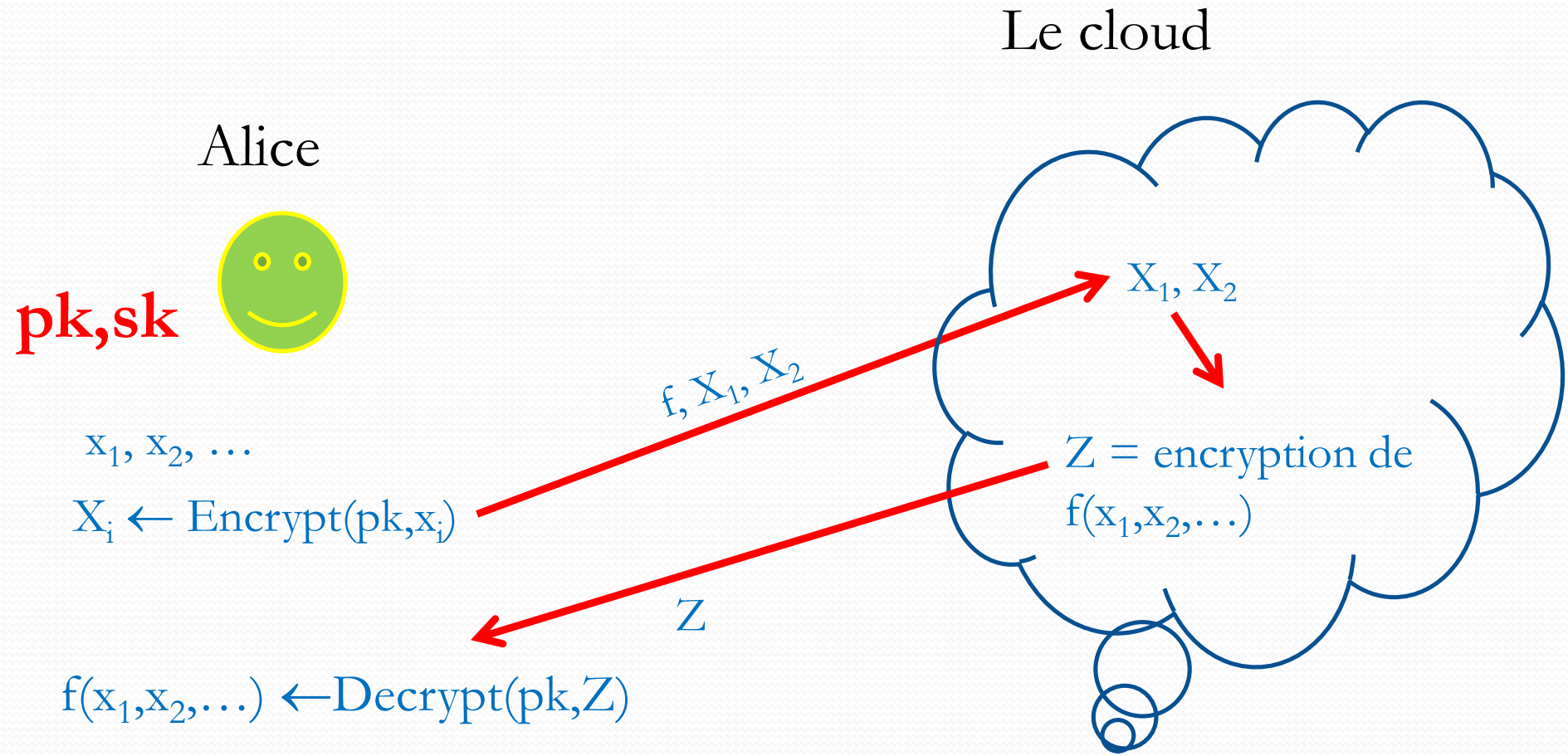
- Modèle précédent permet d'externaliser les données
- Mais pour les traiter, il faut tout d'abord les télécharger, les décrypter
- Serait-il possible d'externaliser aussi les traitements?

Solution : **cryptosystèmes homomorphes**

Rêvons un peu…

- Serait-il possible d'effectuer des opérations sur des données encryptées ?
- Autrement dit, soit deux encryptions X,Y de x,y….serait-il possible de construire (sans connaitre la clé privée) un encryption Z de
  - **z=x+y** ou/et **z=xy**
  - On parlera de cryptosystèmes homomorphes
- Si oui, on a une solution pour la sécurisation du cloud….car avec l'addition et la multiplication, on peut tout calculer

Cryptographie et cloud



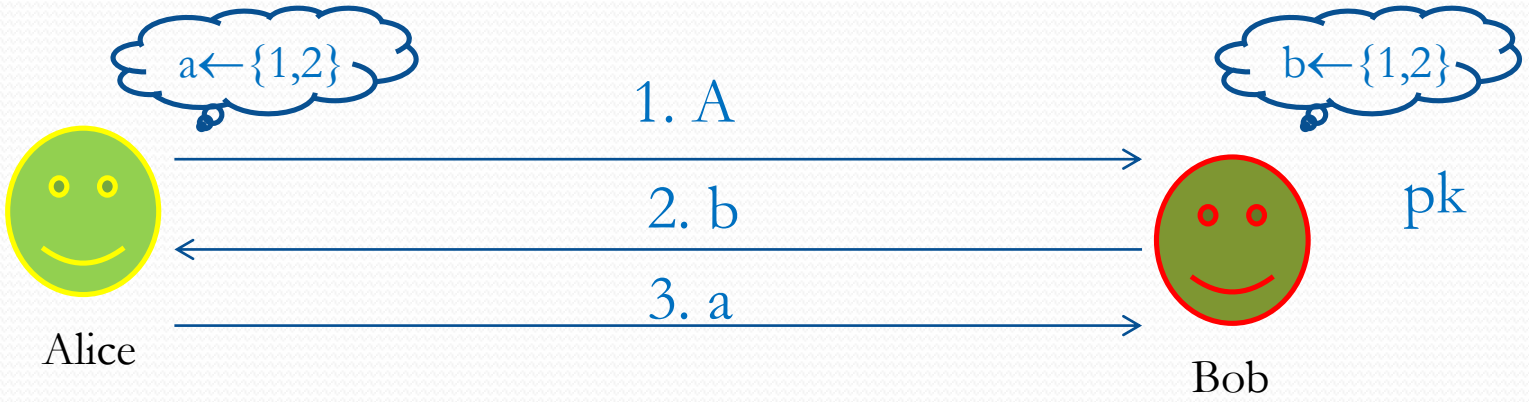
Jouer à pile ou face au téléphone?

Proposer une solution pour jouer à pile ou face au téléphone

Jouer à pile ou face au téléphone?

Proposer une solution pour jouer à pile ou face au téléphone

Une mauvaise solution.

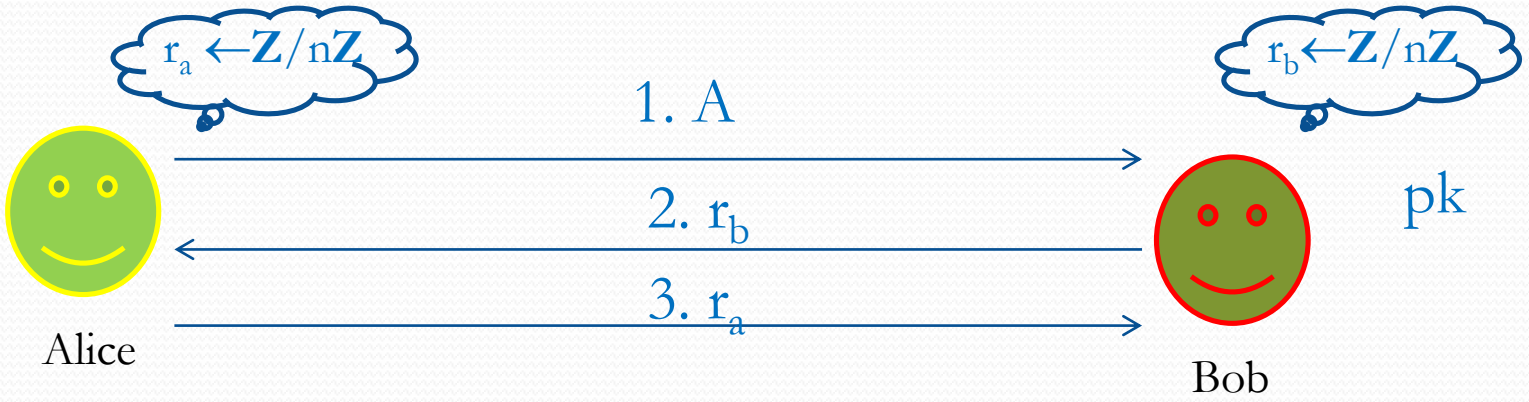


Si a=b alors Alice gagne sinon c'est Bob

Jouer à pile ou face au téléphone?

Proposer une solution pour jouer à pile ou face au téléphone

Une bonne solution.



$(pk,sk) \leftarrow RSA.KeyGen(\lambda)$

$A \leftarrow RSA.Encrypt(pk, r_a)$

Vérifier que :

$A = RSA.Encrypt(pk, r_a)$

Si r_a et r_b ont même parité alors Alice gagne sinon c'est Bob

Mise en gage (to commit)

- A l'étape 1, Alice envoie une encryption A de r_a
  - A est une mise en gage (commitment en anglais) de r_a
  - Alice s'engage sur la valeur r_a (qui ne pourra plus être modifiée) sans la dévoiler
  - A peut être vu comme une boite fermée à clé dont Alice garde la clé
- A l'étape 3, Alice révèle r_a
  - On dit qu'elle ouvre la mise en gage
  - Elle envoie r_a et Bob pourra vérifier que c'est la bonne valeur

⇒sous réserve que RSA est one-way sûr, r_a et r_b sont choisis indépendamment

Cryptosystèmes homomorphes

El Gamal (1985)

Basé sur la difficulté de calculer le logarithme discret, concrètement:

Soit $G=g^r \bmod p$ où r est choisi aléatoirement dans Z_p

Connaissant seulement (G,g) il est difficile de retrouver r

El Gamal (2)

- **KeyGen(λ).** Soit p un grand nombre premier et g un nombre aleatoire de Z_p
  - Choisir aléatoirement r dans Z_p
  - Calculer $G=g^r \bmod p$
  - pk={G}; sk={g}

- **Encrypt(pk,x).** Pour encrypter $x \in Z_p$, choisir un nombre aleatoire u

$$\textbf{Encrypt(pk,x)}=(g^u, G^u x)$$

- **Decrypt(sk,(b,c)).** Retourner $\mathbf{b^{-r}c \bmod p}$

Paillier : point de départ

- Soit n=pq
- On fait l'hypothèse que le problème suivant appelé **« n-residuosity » est difficile**

n-residuosity :

1. Choisir aléatoirement $x_0 \leftarrow \mathbf{Z}_n^2$.
2. Choisir aléatoirement $r \leftarrow \mathbf{Z}_n$
3. Calculer $x_1 = r^n \bmod n^2$.
4. Choisir un bit b aléatoirement
5. Publier $y = x_b$.

Le problème consiste à retrouver b ne connaissant que y (et n).

Et si l'on connait la factorisation de n??

- Le problème **n-residuosity d**evient facile…pourquoi?

Et si l'on connait la factorisation de n??

- Le problème **n-residuosity** devient facile…pourquoi?

Dans ce cas, on peut calculer $\phi(n)$
Comme $\phi(n^2) = n \times \phi(n)$,

$$(r^n)^{\phi(n)}=1$$ (théorème de fermat-Euler)

Donc $b=(y^{\phi(n)}==1)$

- On écrira **n-residuosity ≤ factorisation**

Paillier propose le cryptosysteme suivant:

- KeyGen(λ)

  $pk=\{n=pq\}$; $sk = \{\rho=n^{-1} \bmod \phi(n)\}$

- Encrypt(pk,m)
  - Choisir r aléatoirement dans {0,…,n-1}
  - **retourne**

    $c=(1+m\times n)r^n \bmod n^2$

Decrypt(sk,c)

- **On commence par retrouver r …**

$c=(1+mn)r^n+kn^2 = r^n+n(\dots)$

$\Rightarrow c \bmod n = r^n \bmod n$

$\Rightarrow c^\rho \bmod n = r^{n\times\rho} = r^{1+k\phi(n)} \bmod n = r \bmod n = \mathbf{r}$ car $0 \leq r \leq n-1$.

- **puis m**

$c\times r^{-n} \bmod n^2$

$=(1+nm)r^n r^{-n} \bmod n^2$

$= 1+nm$

$\Rightarrow (c\times r^{-n} \bmod n^2-1)/n = (1+nm-1)/n = m$

Ce chiffrement est-il sémantiquement sur?

Oui si le probleme n-residuosity est difficile….

Propriétés homomorphes

Soit :

- $x,y \in \mathbf{Z}_n$
- $X \leftarrow$ Paillier.Encrypt (pk,x)
- $Y \leftarrow$ Paillier.Encrypt (pk,y)

Montrer que :

Paillier.Decrypt(sk,$X \times Y \bmod n^2$)=$x+y \bmod n$

Exercice. Que vaut Paillier.Decrypt(sk,$X^y \bmod n^2$)?

Propriétés homomorphes

Soit :

- $x,y \in \mathbf{Z}_n$
- $X \leftarrow$ Paillier.Encrypt (pk,x)
- $Y \leftarrow$ Paillier.Encrypt (pk,y)

Montrer que :

Paillier.Decrypt(sk,$X \times Y \bmod n^2$)=$x+y \bmod n$

Exercice. Que vaut Paillier.Decrypt(sk,$X^y \bmod n^2$)?

Réponse : xy

Opérateurs génériques $\oplus$, $\bullet$, $\otimes$

- **Paillier est un cryptosystème homomorphe additif.**
- D'autres cryptosystèmes homomorphes additifs existent.
- Pour ne pas avoir à préciser le cryptosystème, on notera par $\oplus$, $\bullet$ les operateurs génériques définis par:
  - Decrypt (sk,X $\oplus$ Y)=x+y mod n
  - Decrypt (sk,X $\bullet$ y)=xy mod n
- Si le cryptosystème est aussi homomorphe multiplicatif on utilisera $\otimes$

Cryptosystèmes complétement homomorphes (FHE)

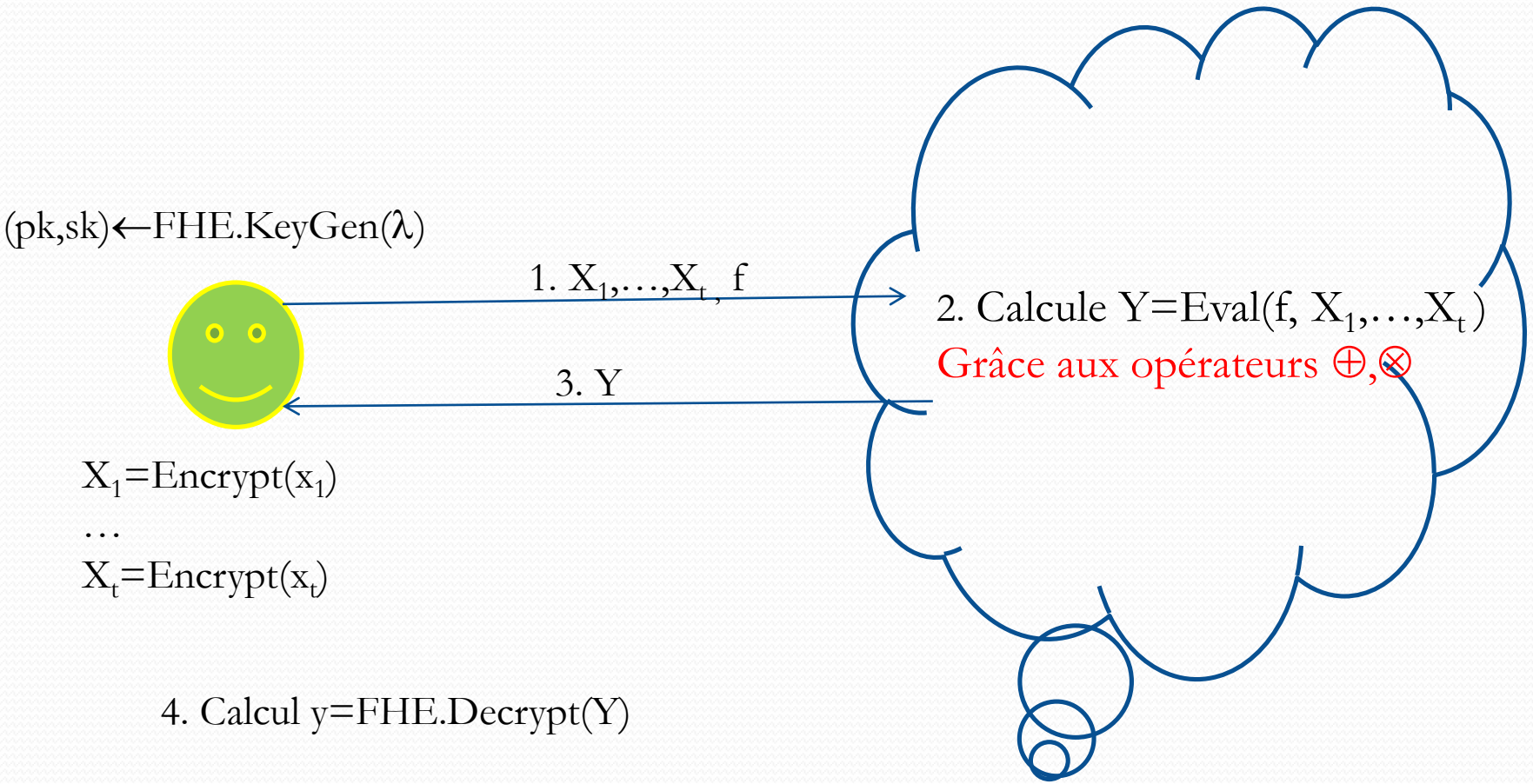
Cryptosystèmes (récents) qui offrent à la fois l'addition et la multiplication homomorphes.

- Ils permettent potentiellement de calculer n'importe quelle fonction
- Outils qui solutionneraient (partiellement) la sécurité dans le cloud computing
- Malheureusement, pas encore opérationnels et de nombreuses recherches sont en œuvre pour améliorés les performances.

Application au cloud computing

- Les FHE permettent d'externaliser les données ainsi que les calculs.
- L'utilisateur encrypte ses données $x_1,\ldots,x_t$ avec FHE.Encrypt et envoie les encryptions $X_1,\ldots,X_t$ dans le cloud (il est le seul possesseur de la clé secrète)
- Imaginons que l'utilisateur souhaite obtenir $y=f(x_1,\ldots,x_t)$
  1. En utilisant les propriétés d'homomorphie, le cloud obtient une encryption Y de y et l'envoie à l'utilisateur
  2. L'utilisateur obtient y=FHE.Decrypt(sk,y)

Application au cloud computing



FHE simple…ou presque

- **KeyGen(λ)**
  - sk= {s} un entier secret (grand)
  - Pour tout i=1,…,m
    - $c_i=q_i s+2e_i+x_i$ avec $x_i \in \{0,1\}$; $e_i << s$; q_i grand entier choisi aléatoirement // ***Intuition.*** *c_i est une encryption de x_i*
  - pk=$\{c_1,\ldots,c_m\}$

- **Encrypt(pk,x)**
  - Choisir un sous-ensemble $S \subseteq \{1,\ldots,m\}$ tel que $x=\sum_{i\in S} x_i \bmod 2$
  - Retourner $c= \sum_{i\in S} c_i$
- **Decrypt(sk,c)**

  Retourner $x=(c \bmod s) \bmod 2$

Propriétés homomorphes

- $c_1 \leftarrow \text{FHE.Encrypt}(pk, x_1)$
- $c_2 \leftarrow \text{FHE.Encrypt}(pk, x_2)$

- $\text{FHE.Decrypt}(sk, c_1 + c_2) = x_1 + x_2 \bmod p$
- $\text{FHE.Decrypt}(sk, c_1 c_2) = x_1 x_2 \bmod p$

Le nombre d'opérations est cependant limité car le bruit augmente :

- Il faut donc une gestion du bruit
- Idée (Gentry 2009) : **décrypter homomorphiquement**

Sécurité

La sécurité repose sur la difficulté du problème suivant :

- Soit s un entier grand secret choisi aléatoirement
- Étant donnés des multiples de s bruitées, i.e.
  - $n_1 = p_1 s + e_1$
  - …
  - $n_t = p_t s + e_t$
- Retrouver s en temps polynomial

Applications

- Les applications sont innombrables
- Toute fonction booléenne(donc tout algorithme) peut être évalué homomorphiquement
- Malheureusement performances trop lentes en pratique

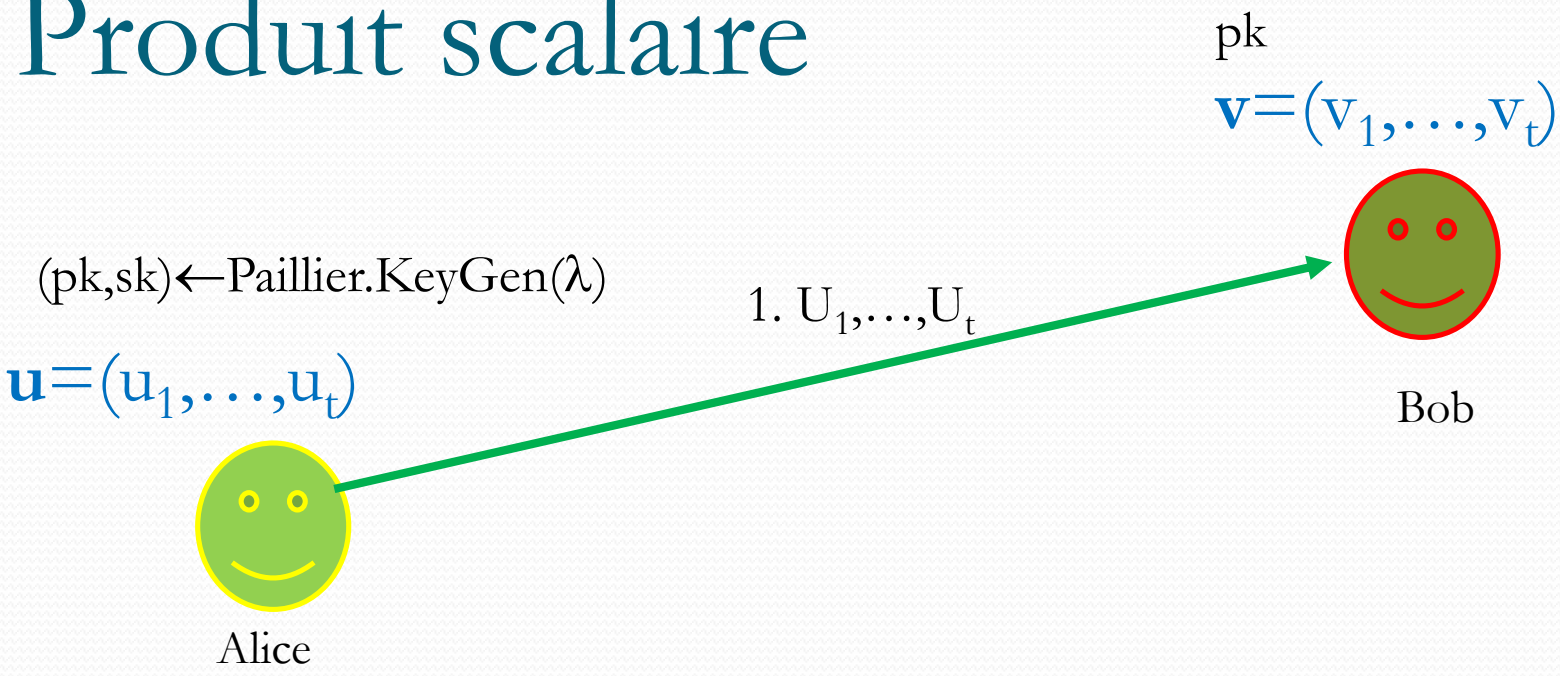
Mais la révolution est en marche…

Produit scalaire

Elaborer un protocole tel que :

• Bob connait **u.v** = $u_1v_1+\ldots+u_tv_t$ à la fin du protocole et **rien de plus**

Produit scalaire



$U_1 \leftarrow Paillier.Encrypt(pk;u_1)$

…

$U_t \leftarrow Paillier.Encrypt(pk;u_t)$

Produit scalaire

pk

$\mathbf{v}=(v_1,\ldots,v_t)$

Bob

$(pk,sk)\leftarrow Paillier.KeyGen(\lambda)$

$\mathbf{u}=(u_1,\ldots,u_t)$

1. $U_1,\ldots,U_t$

Alice

2. $P = U_1 \bullet v_1 \oplus \ldots \oplus U_t \bullet v_t$

$U_1 \leftarrow Paillier.Encrypt(pk;u_1)$

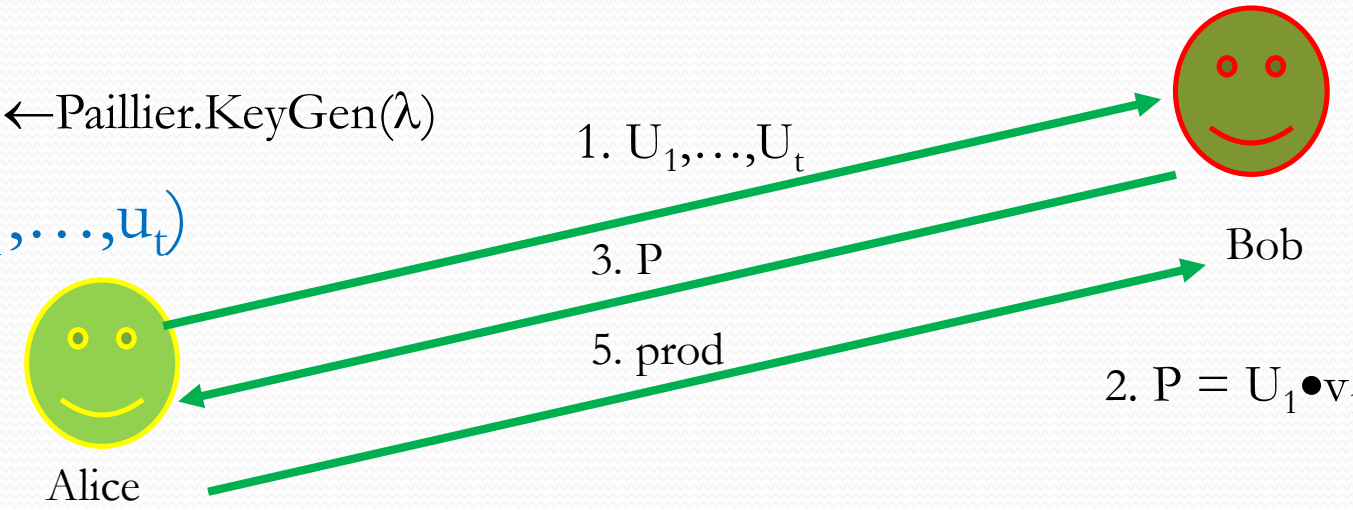
…

$U_t \leftarrow Paillier.Encrypt(pk;u_t)$

Produit scalaire

pk

$\mathbf{v}=(v_1,\ldots,v_t)$



$U_1 \leftarrow$ Paillier.Encrypt(pk;u_1)

…

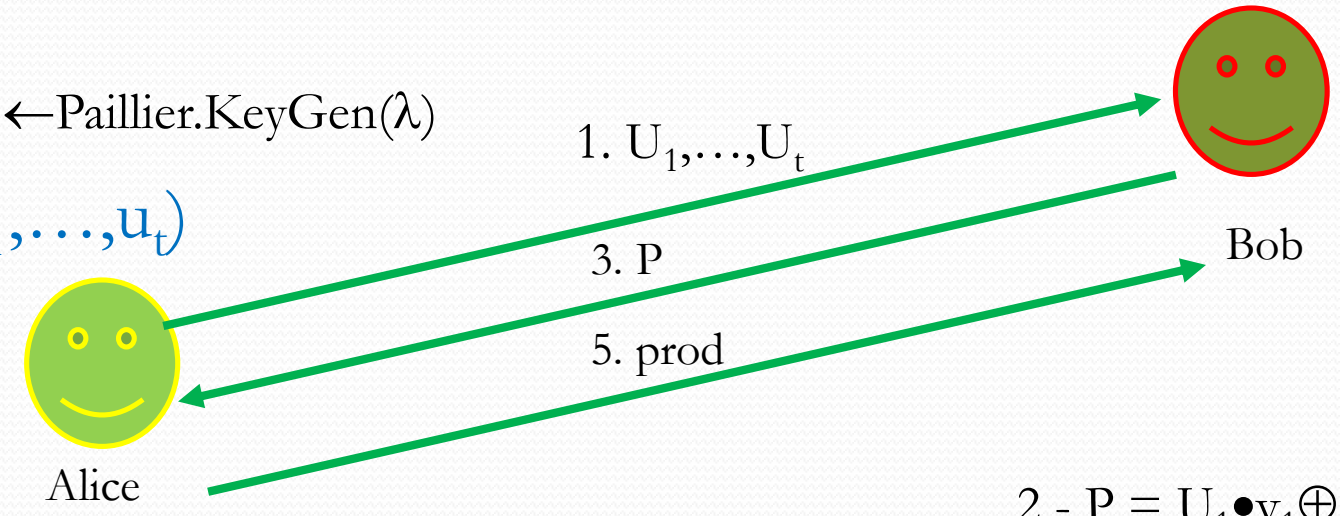
$U_t \leftarrow$ Paillier.Encrypt(pk;u_t)

4. prod ← Paillier.Decrypt(sk,P)

Produit scalaire

pk

$\mathbf{v}=(v_1,\ldots,v_t)$



$2 - P = U_1 \bullet v_1 \oplus \ldots \oplus U_t \bullet v_t$

$U_1 \leftarrow$ Paillier.Encrypt(pk;u_1)

…

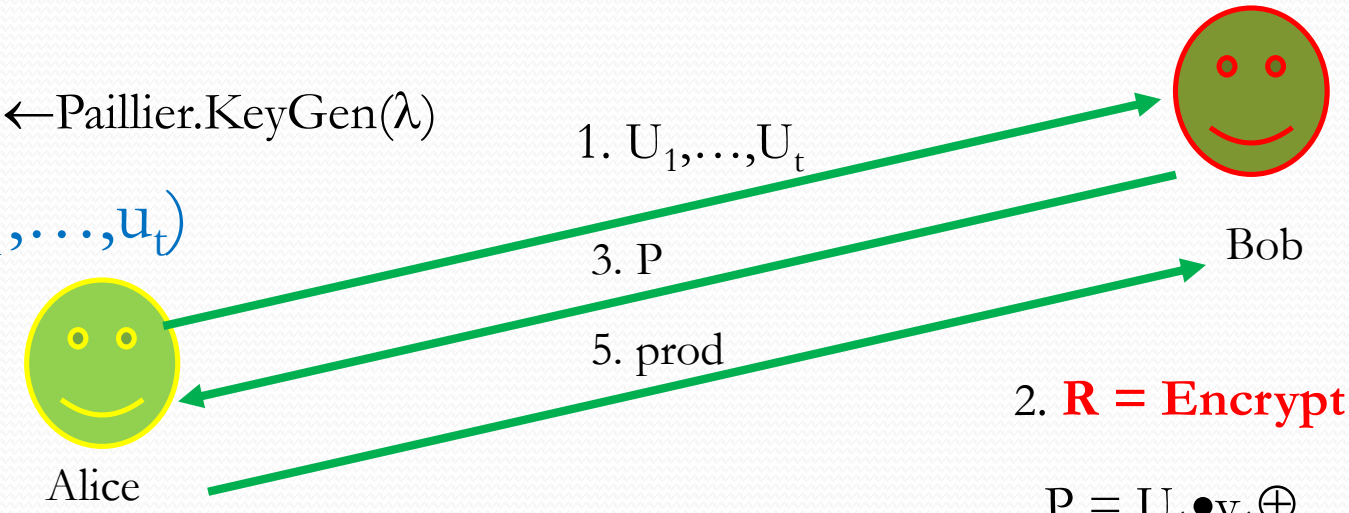
$U_t \leftarrow$ Paillier.Encrypt(pk;u_t)

4. prod ← Paillier.Decrypt(sk,P)

Produit scalaire

pk

$\mathbf{v}=(v_1,\ldots,v_t)$



$(pk,sk)\leftarrow$Paillier.KeyGen(λ)

$\mathbf{u}=(u_1,\ldots,u_t)$

2. **R = Encrypt(pk,r)**

$P = U_1 \bullet v_1 \oplus \ldots \oplus U_t \bullet v_t \oplus R$

$U_1 \leftarrow$Paillier.Encrypt(pk;u_1)

…

$U_t \leftarrow$Paillier.Encrypt(pk;u_t)

5 – **retourne prod-r**

4. prod ← Paillier.Decrypt(sk,P)

Ce protocole est-il sur ?

Plusieurs niveaux d'analyse :

- Bob et Alice respectent le protocole (mais sont curieux) :
  - *honest but curious en anglais*
- …
- Bob ou Alice est malhonnête et dévie arbitrairement du protocole.

Cas honnête mais curieux

- Ce protocole est-il sur?
- Si t est petit et **v** est binaire alors Alice peut retrouver **v** par force brute
  - En générant P elle-même en essayant tous les vecteurs **v** possibles
- **Solution.** randomiser P et ajoutant à P une encryption de 0, i.e.

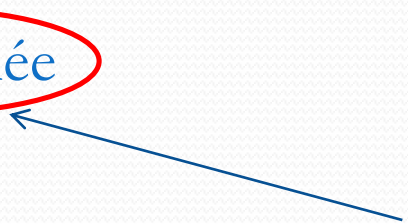
$$P \leftarrow P \oplus \text{Paillier.Encrypt}(pk,0)$$

- Ceci permet d'effacer tout l'historique qui a permis la construction de P

Cas malhonnête?

Identifions les problèmes potentiel :

1. Alice et Bob choisissent leur vecteur **u,v** comme ils veulent…
2. A l'étape 3, Bob peut envoyer l'encryption P de son choix
3. A l'étape 5, Alice peut :
   - Ne rien envoyer
   - Envoyer une valeur *prod* erronée

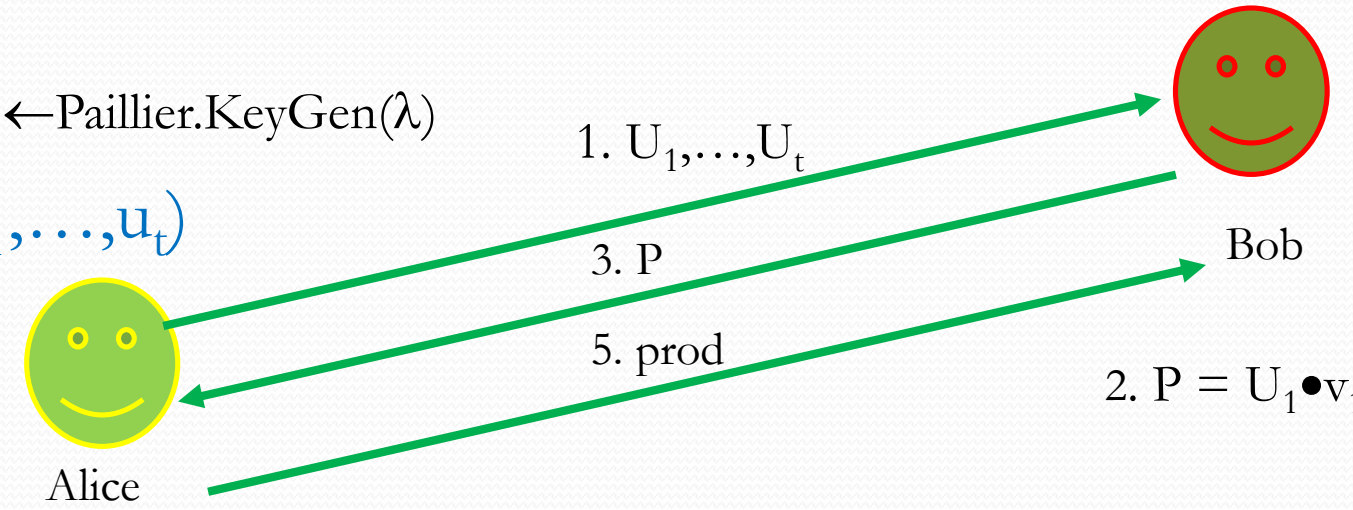


Exercice. Proposer une solution à ce dernier point

Produit scalaire

pk

$\mathbf{v}=(v_1,\ldots,v_t)$



$U_1 \leftarrow$ Paillier.Encrypt(pk;u_1)

…

$U_t \leftarrow$ Paillier.Encrypt(pk;u_t)

4. prod ← Paillier.Decrypt(sk,P)

Calcul Multi-parties

Calcul Multi-parties

Hypothèses

- Soient T parties $p_1,\ldots,p_T$
- Chaque partie p_i a une (ou plusieurs) valeur secrète x_i
- Soient $f_1,\ldots,f_T$ des fonctions définies sur $(x_1,\ldots,x_T)$

But

- Etablir un protocole tel que chaque partie p_i obtienne $f_i(x_1,\ldots,x_T)$ sans rien divulguer sur les x_i

Exemple : Le problème du millionnaire

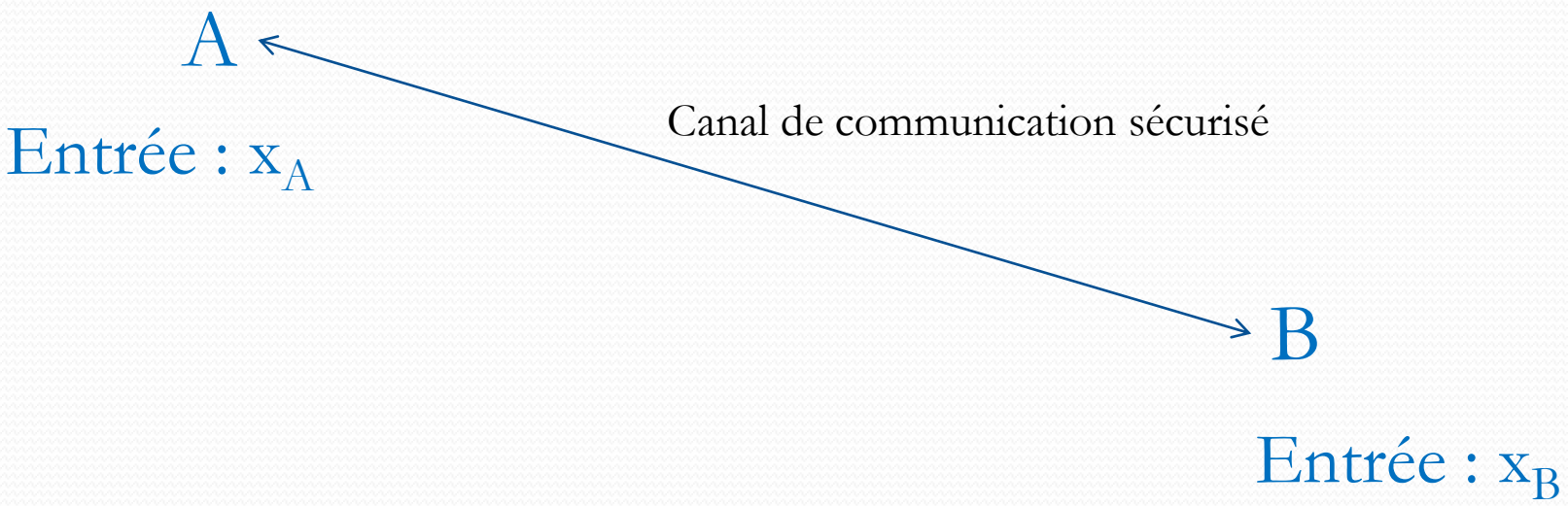
A

Entrée : x_1

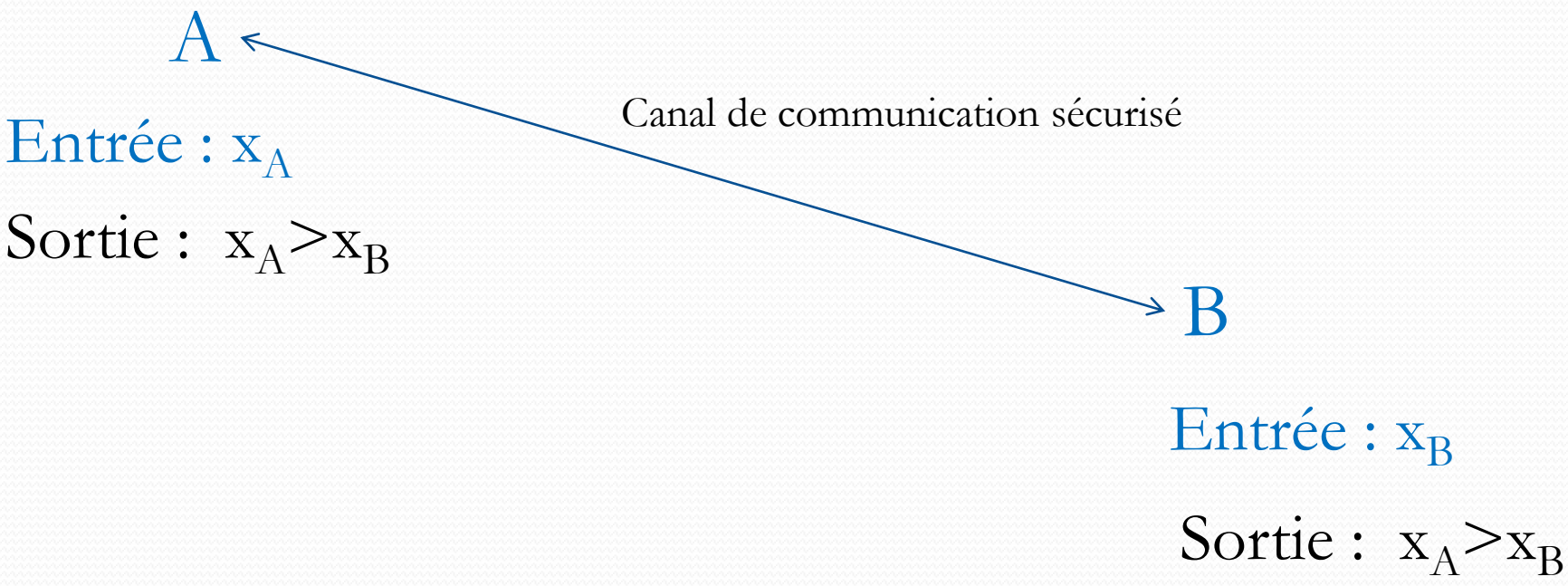
B

Entrée : x_2

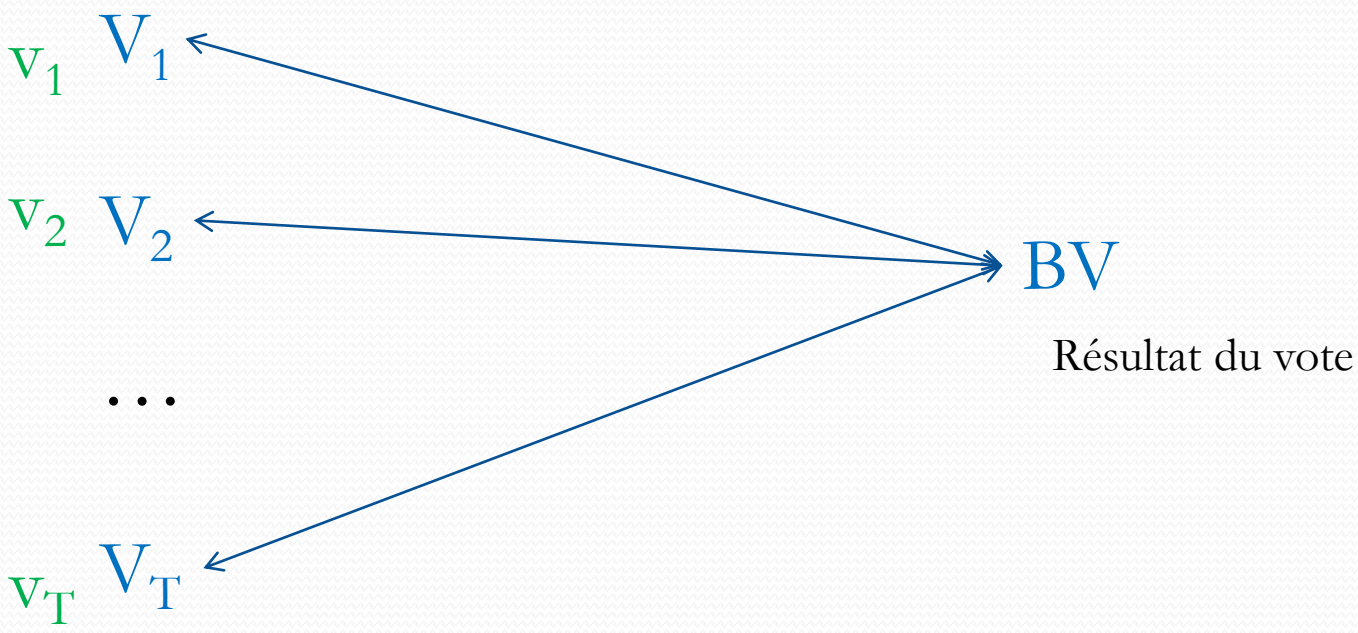
Exemple : Le problème du millionnaire



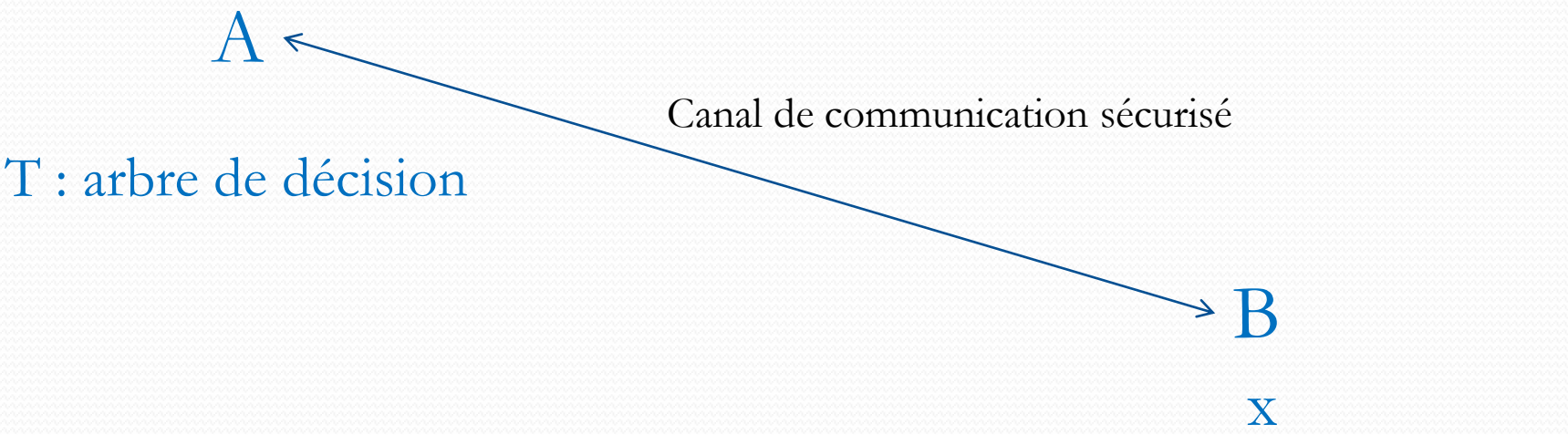
Exemple : Le problème du millionnaire



Exemple : le vote electronique

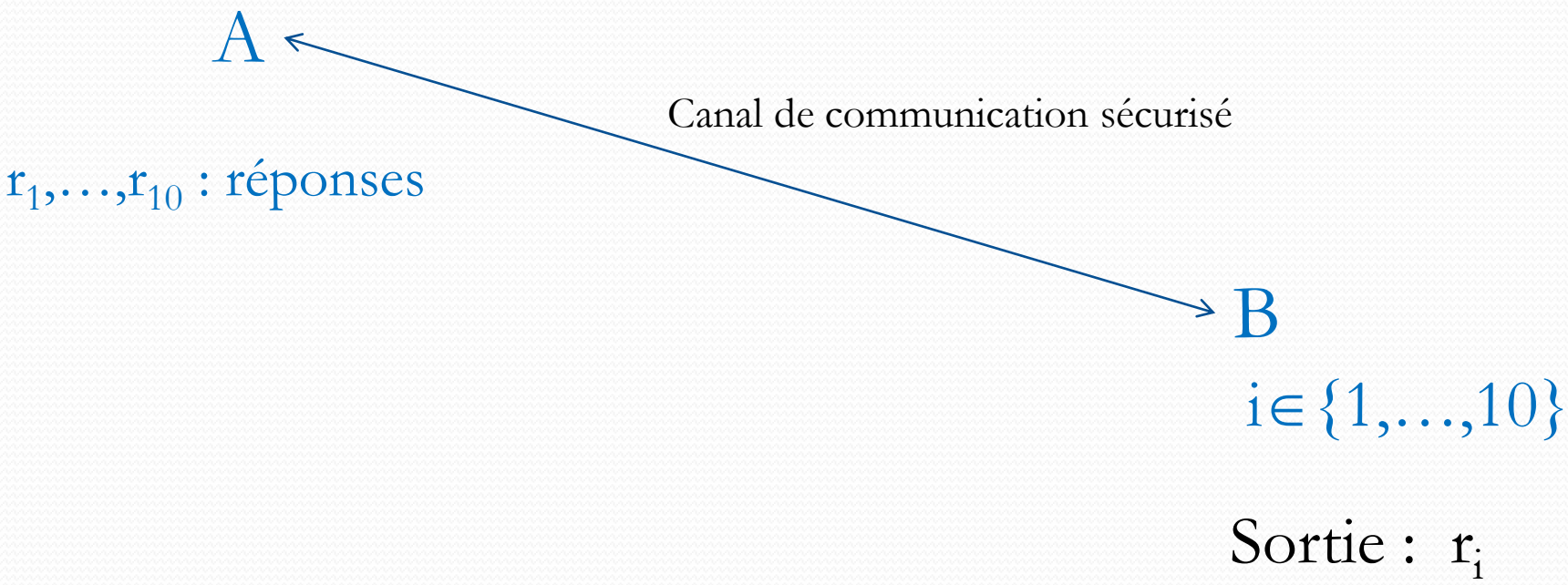


Exemple : machine learning



Sortie : T(x)

Exemple : 10 questions publiques



Qu'est ce qu'un protocole sur?

- On supposera que :
  - les canaux de communications sont sécurisés

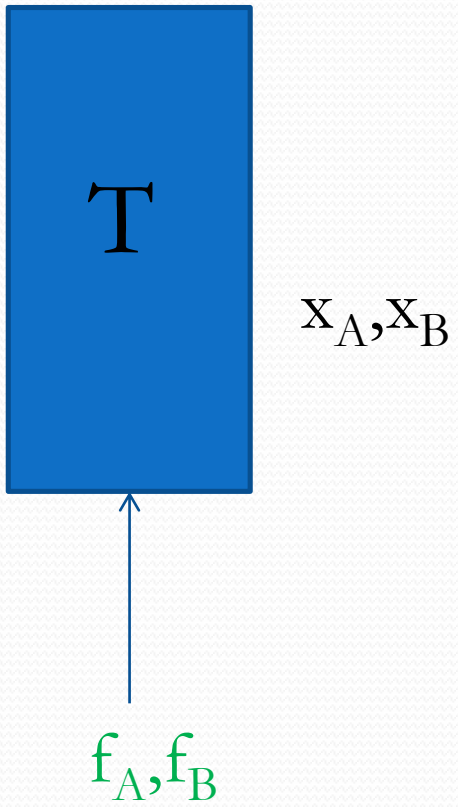
    Messages encryptés, signés…
  - pas de risque d'intrusion.

- Informellement, un protocole est dit sur si aucune partie ne peut obtenir plus que ce qui est prévu par le protocole
  - moyens d'actions : calculer, dévier du protocole, sortir du protocole…

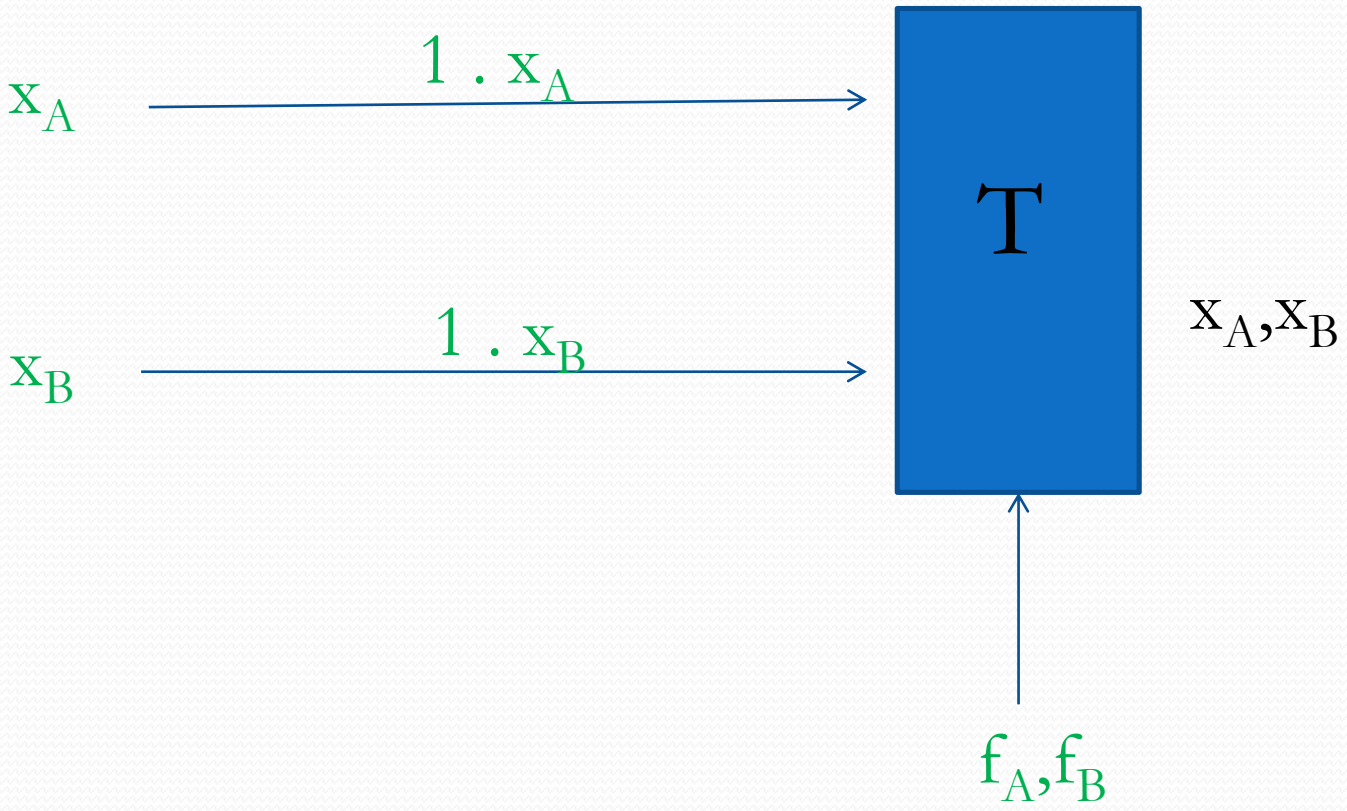
Cas ideal (2 parties)

x_A

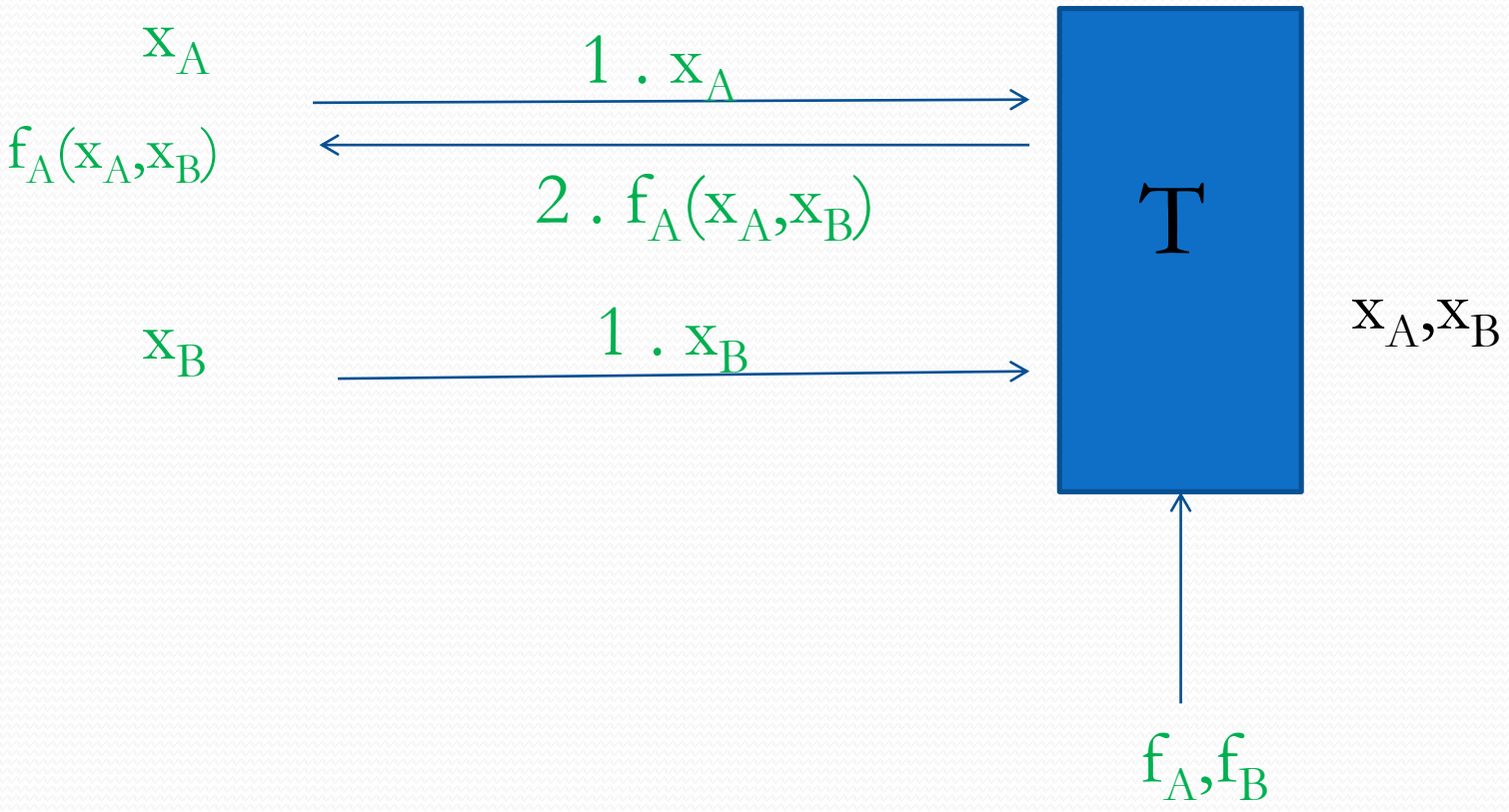
x_B



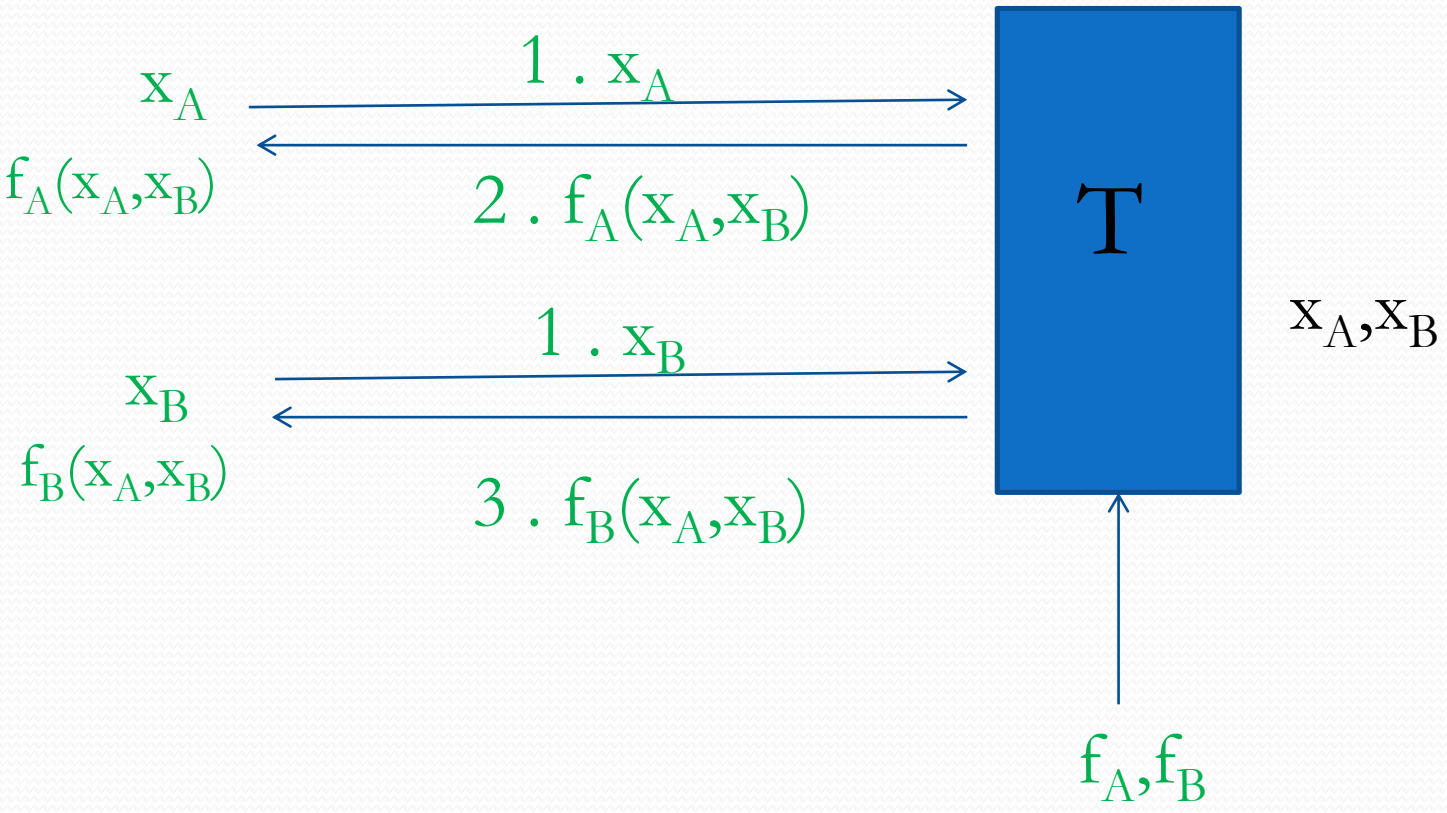
Cas ideal (2 parties)



Cas ideal (2 parties)



Cas ideal (2 parties)



Analyse du cas idéal

- Les parties font confiance à T
- Le résultat reçu (si reçu) est correct
- Alice reçoit le résultat avant Bob

  $\Rightarrow$ si Alice se retire du protocole après avoir reçu son résultat alors Bob ne reçoit rien.
- Les parties peuvent rentrées n'importe quelles valeurs dans le protocole.
  - La cryptographie ne peut rien contre ca!

Question. Est-il possible de faire la même chose sans T?

Protocole cryptographique (2 parties)

- On essaye de se passer de la partie de confiance T et de garder le « même niveau de sécurité » que dans le cas idéal :
  - confidentialité des données, résultat correct
  - comme dans le cas idéal, on ne peut pas garantir que les deux parties reçoivent un résultat
- On suppose qu'il y a une partie *honnête* et une partie *malhonnête*

Un protocole (réel) est dit sûr si la partie malhonnête ne peut pas obtenir un avantage par rapport au cas idéal.

Niveaux de malhonnêteté

- Partie honnête mais curieuse (**malhonnêteté passive**)
  - Cette partie respecte le protocole
  - Cependant elle est curieuse (elle peut faire des calculs pour essayer d'obtenir des informations supplémentaires par exemple)
- **Cas général** (**malhonnêteté active**)
  - **La partie malhonnête agit arbitrairement (elle peut dévier du protocole comme elle l'entend)**
- Plusieurs cas intermédiaires
  - Partie malhonnête qui ne veulent pas se faire repérer (sinon sanctions pénales par exemple)

Sécurité

- Supposons que Bob soit honnête

- Un protocole est dit **sûr** contre une partie malhonnête si :
  - **Correction.** Le résultat reçu par Bob est correct (où il ne reçoit rien)
  - **Confidentialité.** Alice n'apprend rien de plus sur x_B que $f_A(x_A,x_B)$.
  - **Indépendance des entrées.** Alice ne peut pas choisir ses données en fonction de celles de Bob

- Un protocole est dit sur s'il est sur contre toute partie malhonnête

Résultat général

Théorème (cas 2 parties).

Toute fonctionnalité (f_A,f_B) peut être calculée de manière sure (sous réserve de l'existence d'une fonction sens unique et donc que P≠NP)

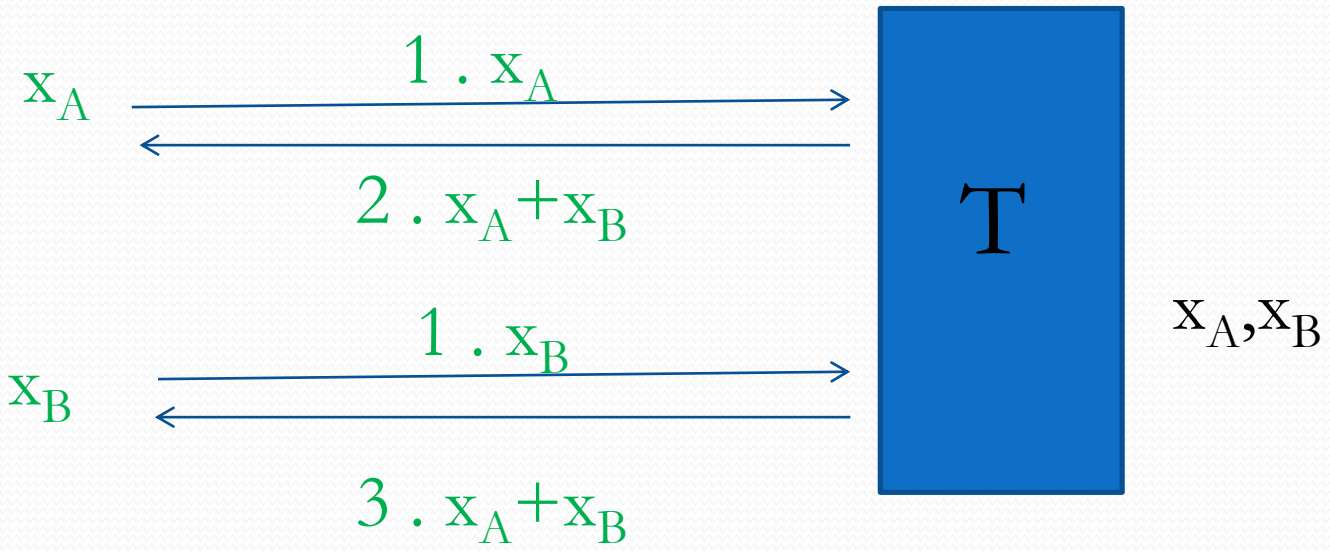
Théorème (cas général)

Toute fonctionnalité $(f_1,\ldots,f_T)$, il existe un protocole calculant f de manière sécurisée s'il y a une majorité de parties honnêtes.

Performances d'un protocole

- La construction générique précédente est très couteuse en général ⇒ besoin de protocoles plus performants en pratique
- Critères de performances d'un protocole
  - Temps de calculs
  - Volume de communications
  - Nombre d'interactions (échanges)

Protocole somme : cas ideal



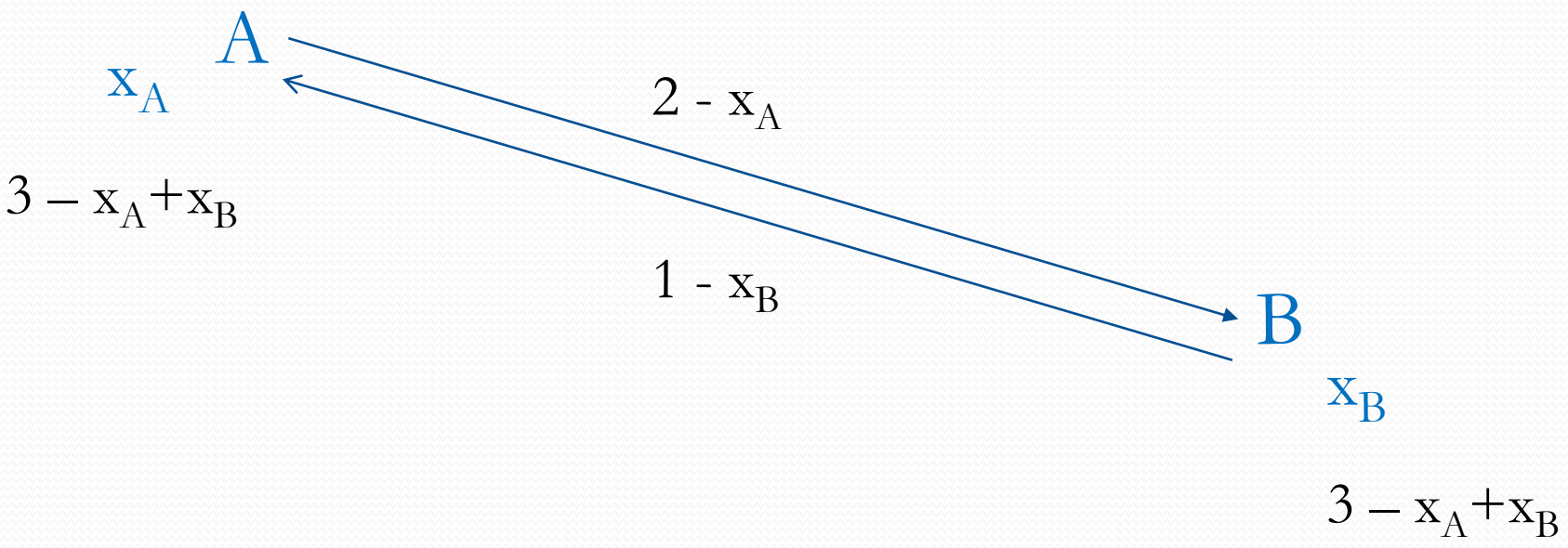
$$f_A(x_A,x_B) = f_B(x_A,x_B) = x_A+x_B$$

Protocole somme : une idée ?

Remarque. Dans le cas idéal, Alice apprend x_B et Bob apprend x_A

⇒ Pas de problème de confidentialité

Que dire de ce protocole ?



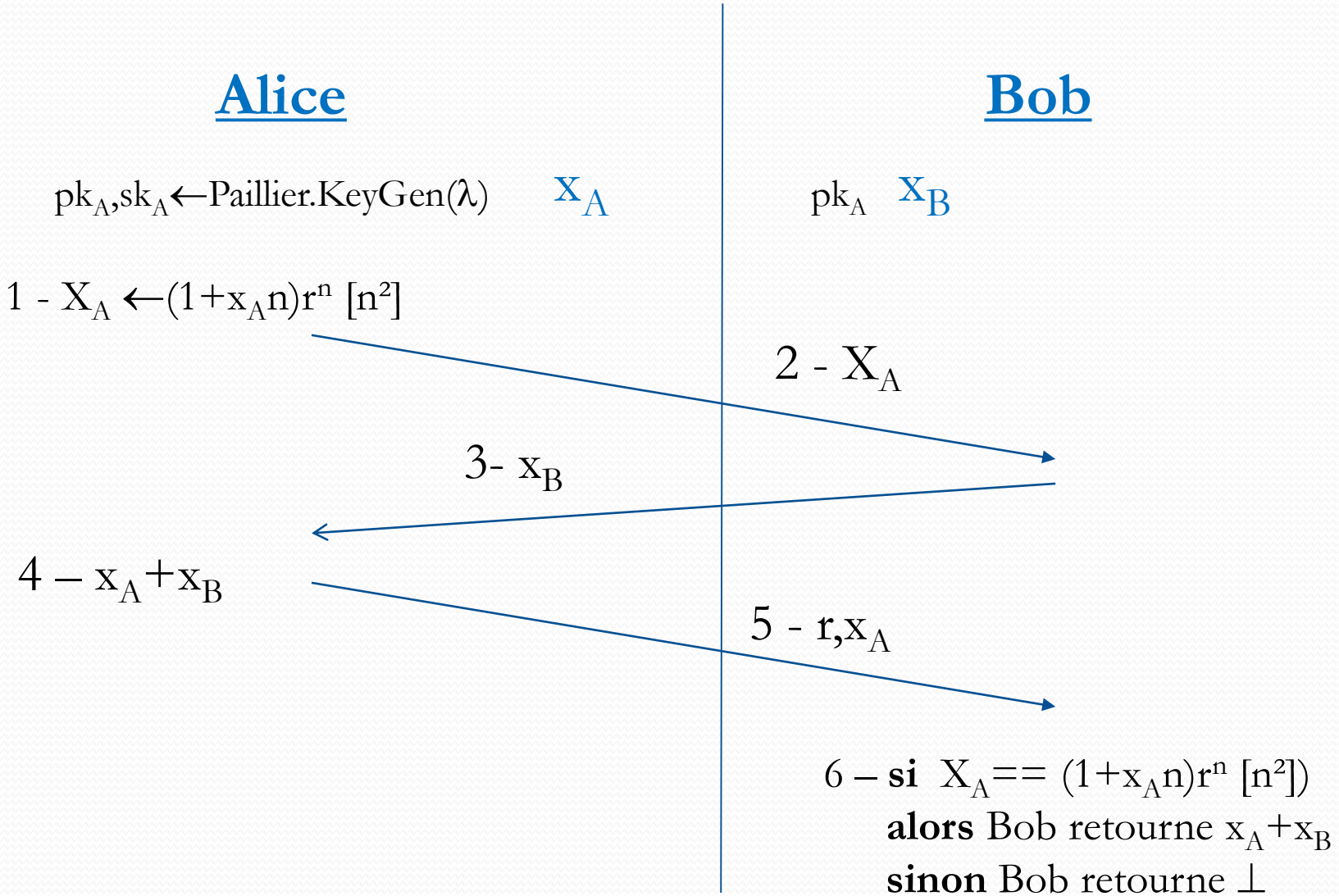
Comparaison avec le cas idéal

- Supposons Bob honnête
- **Correction.** OK
  - Comme dans le cas idéal, les parties Alice entre les valeurs qu'elle veut dans le protocole
- **Confidentialité.** OK
  - Comme dans le cas idéal, Alice apprend x_B…en effet

$$x_B = f_A(x_A, x_B) - x_A$$

- **Indépendance des entrées. NON**
  - **En effet, Alice peut choisir x_B en connaissant x_A**

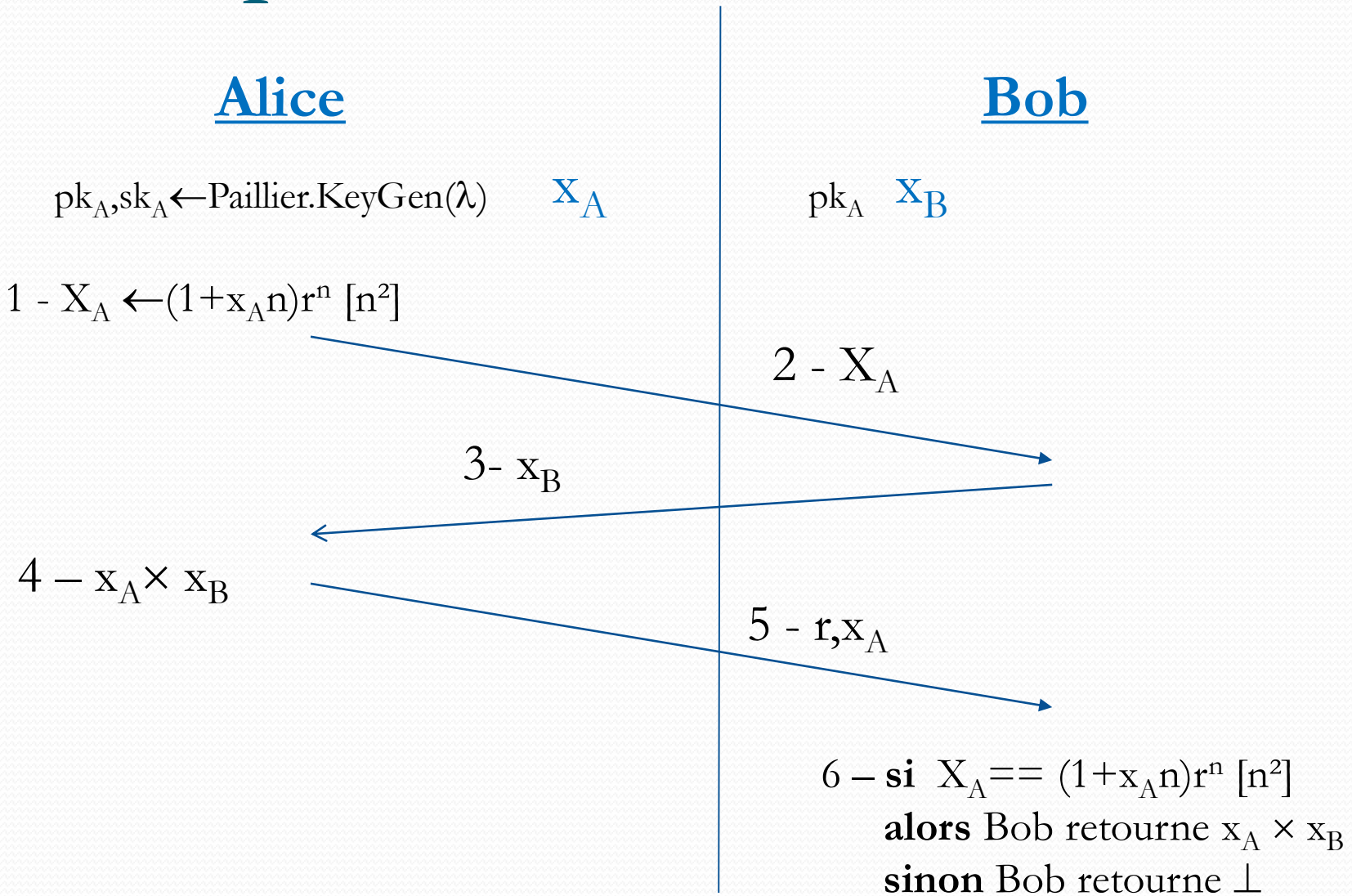
Que faire ?


Alice
Bob
$pk_A, sk_A \leftarrow$ Paillier.KeyGen(λ)
x_A
pk_A
x_B
1 - $X_A \leftarrow (1+x_A n) r^n \ [n^2]$
2 - X_A
3- x_B
4 – $x_A + x_B$
5 - r, x_A
6 – **si** $X_A == (1+x_A n) r^n \ [n^2]$
alors Bob retourne $x_A + x_B$
sinon Bob retourne ⊥

Analyse

- On a résolu le problème de **l'indépendance des entrées.**

Et le produit ?



Analyse

- Cas idéal.

Si $x_A=0$ dans le protocole, Alice reçoit 0 de la partie de confiance et n'apprend donc rien sur x_B

$\neq$

- Cas réel.

Si $x_A=0$ dans le protocole, Alice reçoit quand même x_B

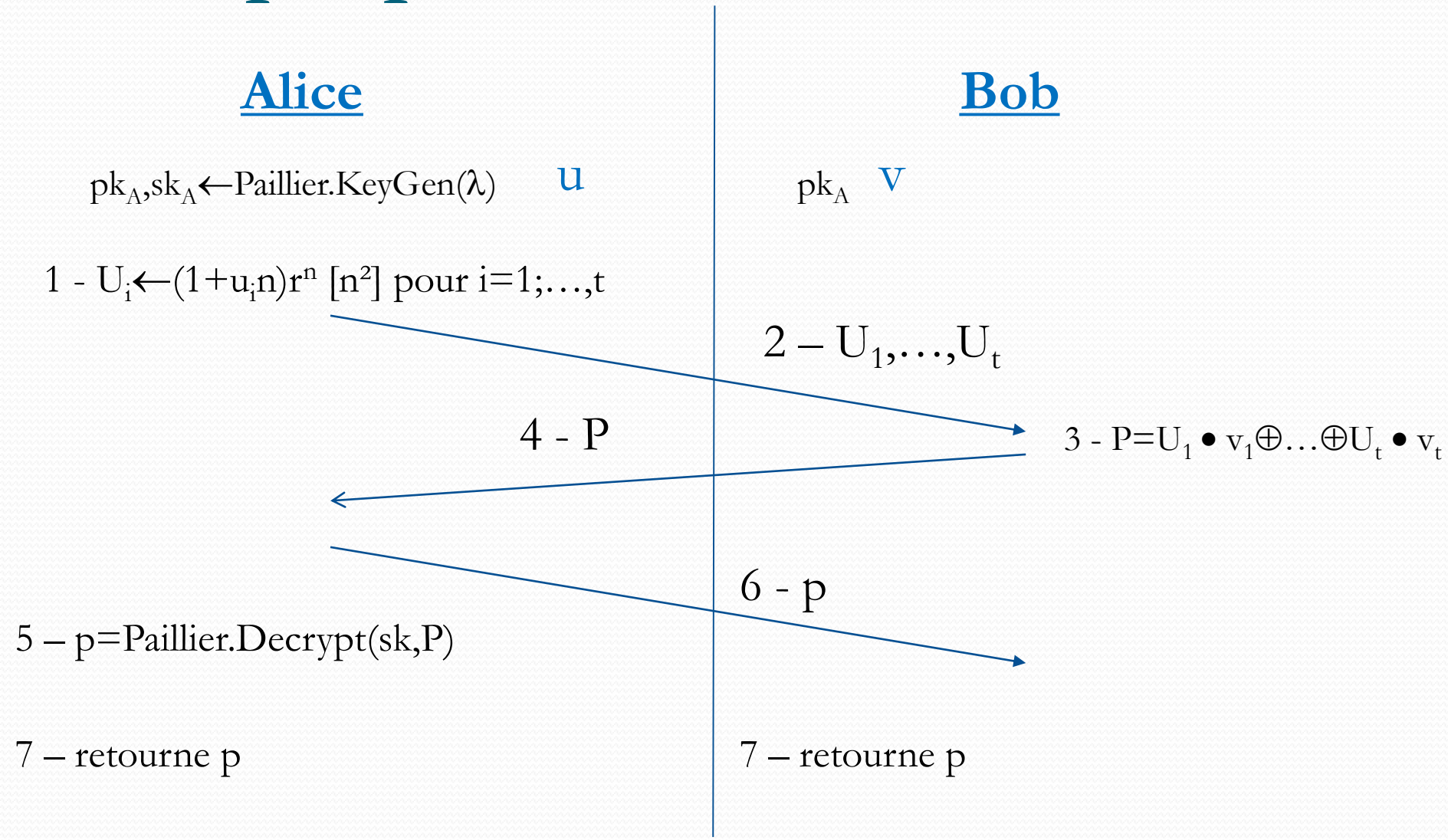
$\Rightarrow$ Pas sûr

Le produit scalaire

- Alice dispose d'un vecteur secret $u=(u_1,\ldots,u_t)$
- Bob dispose d'un vecteur secret $v=(v_1,\ldots,v_t)$
- Alice et Bob souhaite calculer le produit scalaire uv

$$\mathbf{uv=u_1v_1+\ldots+u_tv_t}$$

Une proposition



Failles de ce protocole…

Supposons Alice honnête et Bob malhonnête passif

- Si l'ensemble des vecteurs possibles v de Bob est « petit »
  - Alice peut tous les essayer pour essayer de retrouver C
  - patch…randomiser C avant de l'envoyer…en utilisant les propriétés d'homomorphie.

Failles de ce protocole…

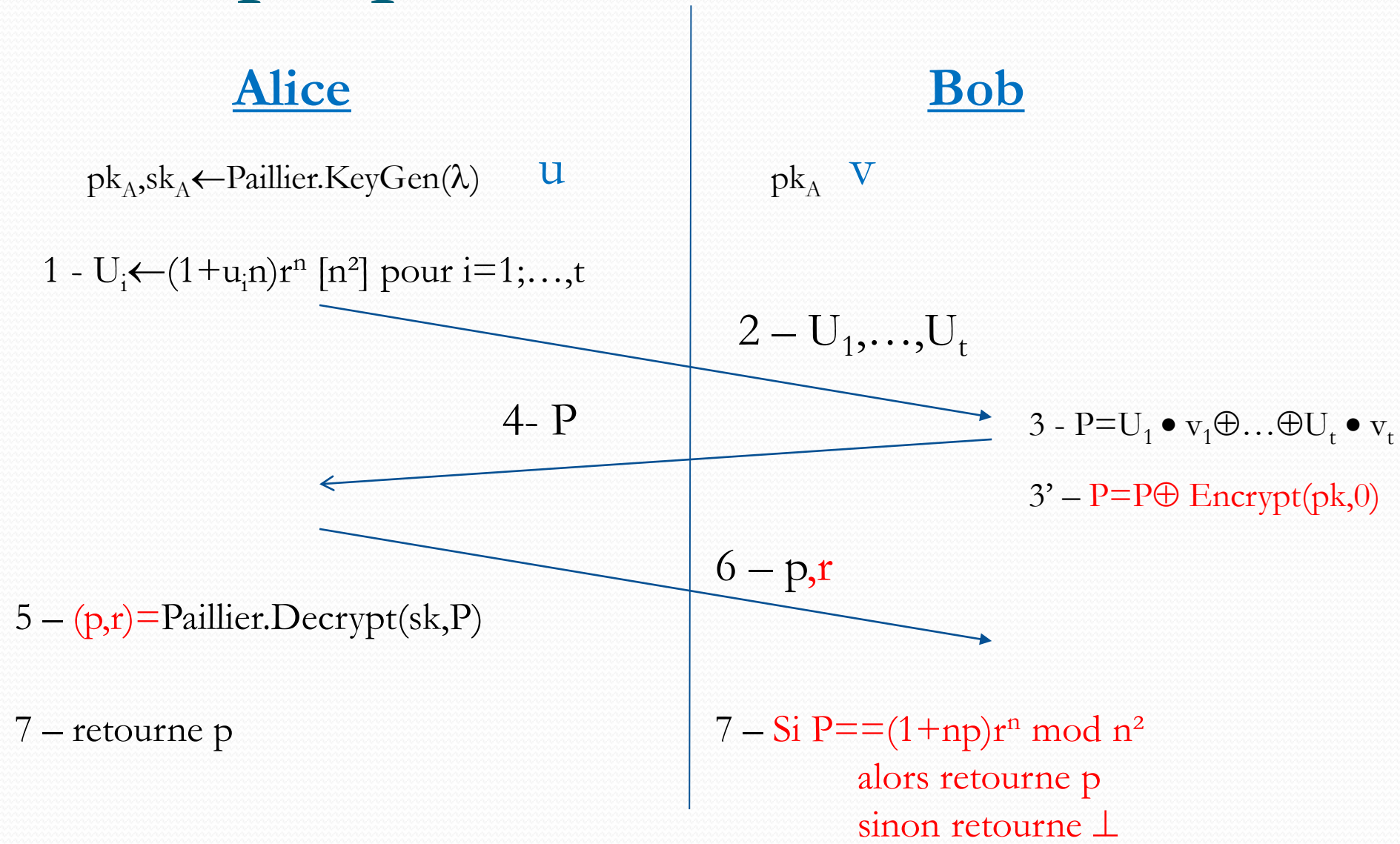
Supposons Alice honnête et Bob malhonnête passif

- Si l'ensemble des vecteurs possibles v de Bob est « petit »
  - Alice peut tous les essayer pour essayer de retrouver C
  - patch…randomiser C avant de l'envoyer…en utilisant les propriétés d'homomorphie.

Supposons Bob honnête

- Alice peut envoyer n'importe quoi à l'étape 6
  - **Idée.** Prouver la décryption (comme dans le protocole somme)

Une proposition



Ces corrections sont-elles suffisantes ?

- On remarque que Bob peut envoyer n'importe quoi à l'étape 4, mais
  - s'il envoie n'importe quoi, il recevra n'importe quoi à l'étape 6
  - il est libre d'entrer le vecteur **v** qu'il souhaite…

Qu'en pensez-vous ? …peut-il en tirer un avantage vs cas idéal ?

Ces corrections sont-elles suffisantes ?

- On remarque que Bob peut envoyer n'importe quoi à l'étape 4, mais
  - s'il envoie n'importe quoi, il recevra n'importe quoi à l'étape 6
  - il est libre d'entrer le vecteur **v** qu'il souhaite…

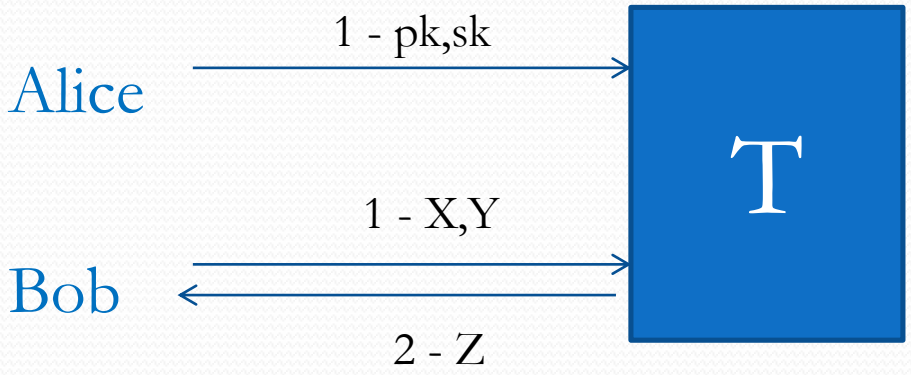
Qu'en pensez-vous ? …peut-il en tirer un avantage vs cas idéal ?

Il faudrait que Bob puisse prouver qu'il agit correctement à l'étape 3 (lorsqu'il construit P) sans rien divulger sur son vecteur v

Le protocole Multiplication – Cas Idéal

Setup.

- Alice possède (pk,sk)←Pailler.KeyGen(λ)
- Bob connait pk et possède deux encryptions X et Y de deux valeurs inconnues x,y



tel que Paillier.Decrypt(sk,Z)=xy

Le protocole Multiplication

Require : Alice has two encryptions E(x),E(y) of unknown values x, y $\in Z_n$

1. Alice sends E(r + x) and E(s + y) where r and s are random numbers chosen in Z_n
2. Bob computes and sends P= E((x+r)(y+s)).
3. Alice computes an encryption of (x+r)(y+s)−sx−ry−rs by using homomorphic properties and outputs it.

Exercice. Ecrire ce protocole de manière plus détaillée.

Le protocole Multiplication

Setup : Bob a généré un couple (pk,sk)←Paillier.KeyGen(λ) et la clé pk a été publiée.

Entrées. Alice a deux encryptions X et Y de valeurs inconnues

1. Alice choisit aléatoirement r,s← **Z**n. Elle calcule et envoie à bob les encryptions suivantes :
   - XR=X ⊕ Paillier.Encrypt(pk,r)
   - YS=Y ⊕ Paillier.Encrypt(pk,s)
2. Bob decrypte XR et YS
   - xr = Paillier.Decrypt(sk,XR) ; ys = Paillier.Decrypt(sk,ys)

   et envoie P=Paillier.Encrypt(pk, xr × ys)
3. …

Analyse

- A l'étape 2, Bob peut envoyer n'importe quoi…
- Il faudrait qu'il puisse prouver qu'il envoie bien une encryption de **xr × ys** sans rien divulguer sur ces valeurs!!!

Comment ?

Analyse

- A l'étape 2, Bob peut envoyer n'importe quoi…
- Il faudrait qu'il puisse prouver qu'il envoie bien une encryption de **xr × ys** sans rien divulguer sur ces valeurs!!!

Comment ?....en utilisant une ZK-proofs

Preuves de savoir sans divulgation de connaissance (ZK – proofs et ZKPOK)

Preuve de savoir sans divulgation de connaissance

- Cette expression désigne un protocole sécurisé dans lequel une entité nommée « fournisseur de preuve », prouve mathématiquement à une autre entité, le « vérificateur », qu'une proposition est vraie sans toutefois révéler une autre information que la véracité de la proposition.
- En pratique, ce schéma se présente souvent sous la forme d'un protocole de type « stimulation/réponse » (*challenge-response*). Le vérificateur et le fournisseur de preuve s'échangent des informations et le vérificateur contrôle si la réponse finale est positive ou négative.

Consistance (*completeness*) :

- si le fournisseur de preuve et le vérificateur suivent le protocole alors le vérificateur doit toujours accepter la preuve

Validité

- Si la proposition est fausse, aucun fournisseur de preuve malicieux ne peut convaincre un vérificateur « honnête » que la proposition est vraie et ceci avec une forte probabilité

aucun apport d'information (*zero knowledge*)

- Le vérificateur n'apprend de la part du fournisseur de preuve rien de plus que la véracité de la proposition, il n'obtient aucune information qu'il ne connaissait déjà sans l'apport du fournisseur de preuve. Si le vérificateur ne suit pas la procédure, cette définition reste valable aussi longtemps que le fournisseur de preuve suit la procédure.

Exemples

1. Soit n=pq. Je souhaite vous prouver que je connais p et q
   - Puis-je vous le prouver sans envoyer p et q ?
2. Soit un graphe G que je sais colorier avec 3 couleurs.
   - Puis-je vous le prouver sans vous apprendre quoi que ce soit sur mon coloriage?

- J'ai réussi à faire un sudoku.
  - Puis-je vous le prouver sans vous montrer la solution ?

Exemples

1. Soit n=pq. Je souhaite vous prouver que je connais p et q
   - Puis-je vous le prouver sans envoyer p et q ?
2. X est une encryption de $b \in \{0,1\}$
   - Puis-je vous le prouver sans révéler b?
- J'ai reussi à faire un sudoku.
   - Puis-je vous le prouver sans vous montrer la solution ?

Réponse. Oui…en utilisant des **preuves de savoir sans divulgation de connaissance** (ZK-proofs)

ZK-proofs

- Ces outils ont été inventés et formalisés en 1989 par Goldwasser, Micali et Rackoff
  - Micali a reçu la médaille Turing pour ses travaux
- Elles ont révolutionné la cryptographie et plus généralement la notion de preuve
- Elle permettent de « compiler » (sécuriser automatiquement) des protocoles prouvés sûrs face à des adversaires passifs en protocoles sûrs face à des adversaires actifs

Sudoku : exemple à la main

3	4		2		9	5	6	1
9		6	5	1	4		3	2
1	2		8	3		7	4	9
	5	3	6	2	1	9		4
	8	2		9	7		5	3
4	9		3	8	5	2	7	
2		4	1			3	9	
8	1	7		6	3	4	2	5
5		9	7	4	2		1	8

Sudoku : exemple à la main

- Comment vous prouver que je sais le faire sans vous donner la solution ?

Sudoku : exemple à la main

- Comment vous prouver que je sais le faire sans vous donner la solution ?

1. On commit la solution avec des papiers retournés

Sudoku : exemple à la main

3	4	8	2	7	9	5	6	1
9	7	6	5	1	4	8	3	2
1	2	5	8	3	6	7	4	9
7	5	3	6	2	1	9	8	4
6	8	2	4	9	7	1	5	3
4	9	1	3	8	5	2	7	6
2	6	4	1	5	8	3	9	7
8	1	7	9	6	3	4	2	5
5	3	9	7	4	2	6	1	8

Sudoku : exemple à la main

- Comment vous prouver que je sais le faire sans vous donner la solution ?

1. Je (Prouveur P) commit la solution avec des papiers retournés
2. Vous (Vérifieur V) choisissez une des 3 contraintes : ligne, colonne ou carré,

   e.g. **ligne**

Sudoku : exemple à la main

Choix étape 2 : Ligne

3	4	8	2	7	9	5	6	1
9	7	6	5	1	4	8	3	2
1	2	5	8	3	6	7	4	9
7	5	3	6	2	1	9	8	4
6	8	2	4	9	7	1	5	3
4	9	1	3	8	5	2	7	6
2	6	4	1	5	8	3	9	7
8	1	7	9	6	3	4	2	5
5	3	9	7	4	2	6	1	8

Sudoku : exemple à la main

- Comment vous prouver que je sais le faire sans vous donner la solution ?

1. Je (Prouveur P) commit la solution avec des papiers retournés
2. Vous (Vérifieur V) choisissez ligne, colonne ou carré
3. J'enlève les papiers de chaque ligne (colonne ou carré), je les ordonne par ordre croissant (ou les trie aléatoirement) et je les retourne

Sudoku : exemple à la main

Choix étape 2 : Ligne

Papiers retournés

3	4	8	2	7	9	5	6	1
9	7	6	5	1	4	8	3	2
1	2	5	8	3	6	7	4	9
7	5	3	6	2	1	9	8	4
6	8	2	4	9	7	1	5	3
4	9	1	3	8	5	2	7	6
2	6	4	1	5	8	3	9	7
8	1	7	9	6	3	4	2	5
5	3	9	7	4	2	6	1	8

????

Sudoku : exemple à la main

Choix étape 2 : Ligne

3	4	8	2	7	9	5	6	1
9	7	6	5	1	4	8	3	2
1	2	5	8	3	6	7	4	9
7	5	3	6	2	1	9	8	4
6	8	2	4	9	7	1	5	3
4	9	1	3	8	5	2	7	6
2	6	4	1	5	8	3	9	7
8	1	7	9	6	3	4	2	5
5	3	9	7	4	2	6	1	8

Papiers retournés

7	8		
7	8		
5	6		
7	8		
1	4	6	
1	6		
5	6	7	8
9			
3	6		

Sudoku : exemple à la main

- Comment vous prouver que je sais le faire sans vous donner la solution ?

1. Je (Prouveur P) commit la solution avec des papiers retournés
2. Vous (Vérifieur V) choisissez ligne, colonne ou carré
3. J'enlève les papiers de chaque ligne (resp. colonne ou carré), je les ordonne par ordre croissant et je les retourne
4. Vous vérifiez que les chiffres manquants sont tous sur les papiers pour chaque ligne (resp. colonne ou carré) si ligne (resp. colonne ou carré) a été choisie à l'étape 2…si ca n'est pas le cas….le **protocole échoue**

Sudoku : exemple à la main

Analyse.

- Si je connais la solution du Sudoku (et que j'agis honnetement) alors le protocole n'échoue pas
- Imaginons que je ne connaisse pas la solution du sudoku
  - Il y a au moins une des 3 contraintes qui n'est pas vérifiée

  $\Rightarrow$ Le protocole échoue avec une probabilité $\geq$ **?**

Sudoku : exemple à la main

Analyse.

- Si je connais la solution du Sudoku (et que j'agis honnetement) alors le protocole n'échoue pas
- Imaginons que je ne connaisse pas la solution du sudoku
  - Il y a au moins une des 3 contraintes qui n'est pas vérifiée

  $\Rightarrow$ Le protocole échoue pas avec une probabilité $\geq$ **1/3**

Sudoku : exemple à la main

Analyse.

- Si je connais la solution du Sudoku (et que j'agis honnetement) alors le protocole n'échoue pas
- Imaginons que je ne connaisse pas la solution du sudoku
  - Il y a au moins une des 3 contraintes qui n'est pas vérifiée

  $\Rightarrow$Le protocole échoue avec une probabilité $\geq$ **1/3**
- Vous n'apprenez rien sur la solution
  - **Vous saviez à l'avance ce que le prouveur allait retourner**

Sudoku : exemple à la main

On répète maintenant le protocole **t** fois

Analyse.

- Si je connais la solution du Sudoku (et que j'agis honnetement) alors le protocole n'échoue pas
- Imaginons que je ne connaisse pas la solution du sudoku
  - Le protocole n'échoue pas avec une probabilité $\leq (1-1/3)^t = (2/3)^t$
  - $(2/3)^t$ décroit exponentiellement vers 0
- Vous n'apprenez toujours rien sur la solution : vous pouvez simuler ce qui vous est dévoilé

En choisissant **t=35,** si **le prouveur ne connait pas la solution alors le protocole échoue avec une proba supérieure à 10^{-6}**

Plus généralement : langage NP

- Soit L un langage NP e.g.
  - l'ensemble des graphes coloriables avec 3 couleurs
  - l'ensemble des entiers n=pq
- Par définition des langages NP,

$$x \in L \Leftrightarrow \exists w \text{ tel que } M(x,w)=1$$

où M est un algorithme efficace (polynomial)

  - e.g. w est un 3-coloriage et M la vérification de ce coloriage

ZK-proof

- Soit L un langage NP et $x \in L$ connu par le prouveur P et le vérifieur V
- Supposons que P connaisse un témoin w, i.e. M(x,w)=1
- Une ZK-proof est un protocole (P,V) entre P et V prouvant à V que **$x \in L$ sans rien divulguer sur x**…notamment sans divulguer w

Remarque. Il est à noter qu'il n'est pas spécifié que P doit prouver qu'il connait w. Si l'on veut que ça soit le cas, on parle alors de **ZKPOK** (Zero-knowledge proof of knowledge)

En formalisant un peu…

Une ZK-proof doit satisfaire les 3 propriétés suivantes :

- **Consistance.** Le protocole réussit pour tout instance $x \in L$ si P et V agissent honnêtement.
- **Solidité.** Si $x \notin L$ alors le protocole échoue avec une probabilité 1 (ou très proche de 1)
- **Zero-savoir.** Un vérifieur, même malhonnête n'apprend rien sur x, en l'occurrence il n'apprend pas w
  - Tout ce qu'il peut déduire (en temps polynomial) de x après le protocole il aurait aussi pu le déduire avant d'y participer.

ZK-proof vs NP

- Soit L_{Ham} l'ensemble des paires (G,C) ou G est un graphe et C est un circuit hamiltonien
  - Un circuit hamiltonien passe par tous les sommets une et une seule fois
- L_{Ham} est un langage NP - complet
- On peut construire une ZK-proof pour L_{Ham}

$\Rightarrow$ **Tout langage NP possède une ZK proof**

Exercice. Construire une ZK-proof pour L_{Ham}

ZK-proof

- Tout langage NP ont une ZK-proof
- **Problème.** La ZK proof précédente est très interactive (beaucoup d'échange entre Alice Bob)
  - Ceci rend son utilisation très couteuse en pratique
- Un des enjeux fondamentaux est de réduire construire des ZK-proofs les moins interactives possibles
- Sous certaines hypothèses, on peut construire des ZK-proofs non-intéractives
  - beaucoup de recherches sur ce sujet

ZK-proof et calcul multi-parties

- Les ZK-proofs sont des outils fondamentaux pour le calcul multi-parties
- Elles permettent de sécuriser (automatiquement) des protocoles
  - On construit un protocole sûrs contres des adversaires passifs (souvent facile à faire)
  - On utilise les ZK-proofs pour forcer l'adversaire à respecter le protocole

Revenons au protocole multiplication…

Setup : Bob a généré un couple (pk,sk)←Paillier.KeyGen(λ), pk a été publiée.

Entrées. Alice a deux encryptions X et Y de valeurs inconnues

1. Alice choisit aléatoirement r,s← **Z**n. Elle calcule et envoie à bob les encryptions suivantes :
   - XR=X ⊕ Paillier.Encrypt(pk,r)
   - YS=Y ⊕ Paillier.Encrypt(pk,s)
2. Bob decrypte XR et YS
   - xr = Paillier.Decrypt(sk,XR) ; ys = Paillier.Decrypt(sk,ys)

   et envoie P=Paillier.Encrypt(pk, xr × ys)
3. …

Revenons au protocole multiplication…

Setup : Bob a généré un couple (pk,sk)←Paillier.KeyGen(λ), pk a été publiée.

Entrées. Alice a deux encryptions X et Y de valeurs inconnues

1. Alice choisit aléatoirement r,s← **Z**n. Elle calcule et envoie à bob les encryptions suivantes :
   - XR=X ⊕ Paillier.Encrypt(pk,r)
   - YS=Y ⊕ Paillier.Encrypt(pk,s)
2. Bob decrypte XR et YS
   - xr = Paillier.Decrypt(sk,XR) ; ys = Paillier.Decrypt(sk,ys)

   et envoie **P=Paillier.Encrypt(pk, xr × ys)**
3. …

Comment garantir que Bob agit correctement !!

Analyse

- Ce protocole est sûr contre des adversaires passifs
- Cependant, il ne l'est pas contre des adversaires actifs
  - A l'étape 2, Bob peut envoyer n'importe quoi
- On peut utiliser une ZK-proof (que l'on appelera proofMultiplication) pour forcer Bob à agir honnêtement

ZK-proofMultiplication pour sécuriser étape 2

- Bob (le prouveur) a généré (pk,sk) avec le cryptosystème de Paillier
- Bob **publie** 3 encryptions X,Y,Z de x,y,z tel que z=xy
- Il veut prouver à Alice (le vérifieur) que Z est bien une encryption de z=xy sans divulguer aucune information sur x,y

ZK-proofMultiplication

- **Bob** choisit *a* aléatoirement dans $\mathbf{Z}_n$ et publie les encryptions A,B de *a* et *a*y
- **Alice** choisit e $\leftarrow Z_n^*$ et publie e
- **Bob** calcule et publie
  1. (c,r)=Decrypt(sk, $X^e A \bmod n^2$)
  2. (d,r')=Decrypt(sk, $Y^c B^{-1} Z^{-e} \bmod n^2$)
- **Alice** accepte la preuve si
  1. $(1+cn)r^n \bmod n^2 = X^e A \bmod n^2$
  2. $r'^n = Y^c B^{-1} Z^{-e} \bmod n^2$ // *d=0*

Analyse

- Cette ZK-proof **ne requiert qu'un échange** entre Bob et Alice
  - C'est un Σ - protocole
- Elle permet de rendre le protocole multiplication sûr contre des adversaires actifs
- **Son analyse fait l'objet du TP/TD 5**