Cours de Cryptographie

Yves Legrandgérard

email : ylg@irif.fr

Laboratoire IRIF - Université Paris Diderot

Septembre 2019

Table des Matières I

Table des Matières II

Table des Matières III

Table des Matières IV

Qu'est-ce que la cryptographie ?

- Une réponse un peu formelle : c'est la discipline qui traite de la **transmission confidentielle** de données.
- C'est une discipline très ancienne qui remonte à l'antiquité et qui a connu un bouleversement profond à la fin du $20^{ème}$ sciècle.
- Il y a essentiellement deux types de cryptographie :
  - La cryptographie à **clé secrète** ou cryptographie symétrique. C'est la plus ancienne.
  - La cryptographie à **clé publique** ou cryptographie asymétrique. C'est la plus récente : on considère généralement qu'elle est née en 1976 avec l'article de *Diffie* et *Hellman* : "New directions in cryptography".
- Le chiffrement symétrique est **beaucoup plus rapide** que le chiffrement asymétrique mais a l'inconvénient de nécessiter le partage au préalable d'une clé secrète.
- En pratique, on utilise d'abord un **chiffrement asymétrique** pour échanger la clé secrète et ensuite un **chiffrement symétrique** pour l'échange des données.

Qu'est-ce que la cryptographie ?

- Une réponse un peu formelle : c'est la discipline qui traite de la **transmission confidentielle** de données.
- C'est une discipline très ancienne qui remonte à l'antiquité et qui a connu un bouleversement profond à la fin du $20^{ème}$ sciècle.
- Il y a essentiellement deux types de cryptographie :
  - La cryptographie à **clé secrète** ou cryptographie symétrique. C'est la plus ancienne.
  - La cryptographie à **clé publique** ou cryptographie asymétrique. C'est la plus récente : on considère généralement qu'elle est née en 1976 avec l'article de *Diffie* et *Hellman* : "New directions in cryptography".
- Le chiffrement symétrique est **beaucoup plus rapide** que le chiffrement asymétrique mais a l'inconvénient de nécessiter le partage au préalable d'une clé secrète.
- En pratique, on utilise d'abord un **chiffrement asymétrique** pour échanger la clé secrète et ensuite un **chiffrement symétrique** pour l'échange des données.

Qu'est-ce que la cryptographie ?

- Une réponse un peu formelle : c'est la discipline qui traite de la **transmission confidentielle** de données.
- C'est une discipline très ancienne qui remonte à l'antiquité et qui a connu un bouleversement profond à la fin du $20^{ème}$ sciècle.
- Il y a essentiellement deux types de cryptographie :
  - La cryptographie à **clé secrète** ou cryptographie symétrique. C'est la plus ancienne.
  - La cryptographie à **clé publique** ou cryptographie asymétrique. C'est la plus récente : on considère généralement qu'elle est née en 1976 avec l'article de *Diffie* et *Hellman* : "New directions in cryptography".
- Le chiffrement symétrique est **beaucoup plus rapide** que le chiffrement asymétrique mais a l'inconvénient de nécessiter le partage au préalable d'une clé secrète.
- En pratique, on utilise d'abord un **chiffrement asymétrique** pour échanger la clé secrète et ensuite un **chiffrement symétrique** pour l'échange des données.

Qu'est-ce que la cryptographie ?

- Une réponse un peu formelle : c'est la discipline qui traite de la **transmission confidentielle** de données.
- C'est une discipline très ancienne qui remonte à l'antiquité et qui a connu un bouleversement profond à la fin du $20^{ème}$ sciècle.
- Il y a essentiellement deux types de cryptographie :
  - La cryptographie à **clé secrète** ou cryptographie symétrique. C'est la plus ancienne.
  - La cryptographie à **clé publique** ou cryptographie asymétrique. C'est la plus récente : on considère généralement qu'elle est née en 1976 avec l'article de *Diffie* et *Hellman* : "New directions in cryptography".
- Le chiffrement symétrique est **beaucoup plus rapide** que le chiffrement asymétrique mais a l'inconvénient de nécessiter le partage au préalable d'une clé secrète.
- En pratique, on utilise d'abord un **chiffrement asymétrique** pour échanger la clé secrète et ensuite un **chiffrement symétrique** pour l'échange des données.

Qu'est-ce que la cryptographie ?

- Une réponse un peu formelle : c'est la discipline qui traite de la **transmission confidentielle** de données.
- C'est une discipline très ancienne qui remonte à l'antiquité et qui a connu un bouleversement profond à la fin du $20^{ème}$ sciècle.
- Il y a essentiellement deux types de cryptographie :
  - La cryptographie à **clé secrète** ou cryptographie symétrique. C'est la plus ancienne.
  - La cryptographie à **clé publique** ou cryptographie asymétrique. C'est la plus récente : on considère généralement qu'elle est née en 1976 avec l'article de *Diffie* et *Hellman* : "New directions in cryptography".
- Le chiffrement symétrique est **beaucoup plus rapide** que le chiffrement asymétrique mais a l'inconvénient de nécessiter le partage au préalable d'une clé secrète.
- En pratique, on utilise d'abord un **chiffrement asymétrique** pour échanger la clé secrète et ensuite un **chiffrement symétrique** pour l'échange des données.

Qu'est-ce que la cryptographie ?

- Une réponse un peu formelle : c'est la discipline qui traite de la **transmission confidentielle** de données.
- C'est une discipline très ancienne qui remonte à l'antiquité et qui a connu un bouleversement profond à la fin du $20^{ème}$ sciècle.
- Il y a essentiellement deux types de cryptographie :
  - La cryptographie à **clé secrète** ou cryptographie symétrique. C'est la plus ancienne.
  - La cryptographie à **clé publique** ou cryptographie asymétrique. C'est la plus récente : on considère généralement qu'elle est née en 1976 avec l'article de *Diffie* et *Hellman* : "New directions in cryptography".
- Le chiffrement symétrique est **beaucoup plus rapide** que le chiffrement asymétrique mais a l'inconvénient de nécessiter le partage au préalable d'une clé secrète.
- En pratique, on utilise d'abord un **chiffrement asymétrique** pour échanger la clé secrète et ensuite un **chiffrement symétrique** pour l'échange des données.

Qu'est-ce que la cryptographie ?

- Une réponse un peu formelle : c'est la discipline qui traite de la **transmission confidentielle** de données.
- C'est une discipline très ancienne qui remonte à l'antiquité et qui a connu un bouleversement profond à la fin du $20^{ème}$ sciècle.
- Il y a essentiellement deux types de cryptographie :
  - La cryptographie à **clé secrète** ou cryptographie symétrique. C'est la plus ancienne.
  - La cryptographie à **clé publique** ou cryptographie asymétrique. C'est la plus récente : on considère généralement qu'elle est née en 1976 avec l'article de *Diffie* et *Hellman* : "New directions in cryptography".
- Le chiffrement symétrique est **beaucoup plus rapide** que le chiffrement asymétrique mais a l'inconvénient de nécessiter le partage au préalable d'une clé secrète.
- En pratique, on utilise d'abord un **chiffrement asymétrique** pour échanger la clé secrète et ensuite un **chiffrement symétrique** pour l'échange des données.

Quelques exemples d'algorithmes symétriques

- Le chiffre de *César* utilisé par *Jules César* dans ses correspondances :

$$\begin{pmatrix} A & B & C & D & E & F & G & H & I & J & K & L & M & N & O & P & Q & R & S & T & U & V & W & X & Y & Z \\ \Downarrow & \Downarrow \\ D & E & F & G & H & I & J & K & L & M & N & O & P & Q & R & S & T & U & V & W & X & Y & Z & A & B & C \end{pmatrix}$$

Suétone rapporte que *Jules César* utilisait systématiquement le décalage de trois lettres ci-dessus, donc la même clé.

- Le chiffre de *Vigenère* (1586) qui utilise le chiffre de *César* sauf que le décalage de chaque lettre du texte en clair est dépendant de la position celle-ci dans le texte. Ce décalage est calculé à l'aide d'une clé secrète. Cet algorithme, contrairement au précédent, résiste à l'*analyse par fréquences* mais a été cassé par *Kasiski* en 1863.

Quelques exemples d'algorithmes symétriques

- Le chiffre de *César* utilisé par *Jules César* dans ses correspondances :

$$\begin{pmatrix} A & B & C & D & E & F & G & H & I & J & K & L & M & N & O & P & Q & R & S & T & U & V & W & X & Y & Z \\ \Downarrow & \Downarrow \\ D & E & F & G & H & I & J & K & L & M & N & O & P & Q & R & S & T & U & V & W & X & Y & Z & A & B & C \end{pmatrix}$$

  Suétone rapporte que *Jules César* utilisait systématiquement le décalage de trois lettres ci-dessus, donc la même clé.

- Le chiffre de *Vigenère* (1586) qui utilise le chiffre de *César* sauf que le décalage de chaque lettre du texte en clair est dépendant de la position celle-ci dans le texte. Ce décalage est calculé à l'aide d'une clé secrète. Cet algorithme, contrairement au précédent, résiste à l'*analyse par fréquences* mais a été cassé par *Kasiski* en 1863.

Quelques exemples d'algorithmes symétriques

- Le chiffre de *César* utilisé par *Jules César* dans ses correspondances :

$$\begin{pmatrix} A & B & C & D & E & F & G & H & I & J & K & L & M & N & O & P & Q & R & S & T & U & V & W & X & Y & Z \\ \Downarrow & \Downarrow \\ D & E & F & G & H & I & J & K & L & M & N & O & P & Q & R & S & T & U & V & W & X & Y & Z & A & B & C \end{pmatrix}$$

  Suétone rapporte que *Jules César* utilisait systématiquement le décalage de trois lettres ci-dessus, donc la même clé.
- Le chiffre de *Vigenère* (1586) qui utilise le chiffre de *César* sauf que le décalage de chaque lettre du texte en clair est dépendant de la position celle-ci dans le texte. Ce décalage est calculé à l'aide d'une clé secrète. Cet algorithme, contrairement au précédent, résiste à l'*analyse par fréquences* mais a été cassé par *Kasiski* en 1863.

Quelques exemples d'algorithmes symétriques

- Le chiffre de *Vernam* (1926) est un chiffre de *Vigenère* où la clé secrète, de même longueur que le message à chiffrer, est choisie aléatoirement et n'est utilisée qu'une fois. Cet algorithme est théoriquement **incassable** mais sa mise en œuvre est difficile en pratique. Néanmoins, la ligne téléphonique entre le Kremlin et la Maison Blanche a été, à un moment, protégée par un tel système.
- Une liste non exhaustive d'algorithme de chiffrements symétriques datant de la période "moderne" : DES (n'est plus utilisé), 3DES (variante du précédent), AES (le **standard actuel** en trois déclinaisons : 128, 192 et 256 bits), RC4, RC5 et d'autres encore.
- Rappelons que tous ces algorithmes symétriques nécessitent que les participants à l'échange de données disposent, au préalable, d'une même **clé secrète partagée**. C'est pour cela qu'ils ne sont presque jamais utilisés seuls.

Quelques exemples d'algorithmes symétriques

- Le chiffre de *Vernam* (1926) est un chiffre de *Vigenère* où la clé secrète, de même longueur que le message à chiffrer, est choisie aléatoirement et n'est utilisée qu'une fois. Cet algorithme est théoriquement **incassable** mais sa mise en œuvre est difficile en pratique. Néanmoins, la ligne téléphonique entre le Kremlin et la Maison Blanche a été, à un moment, protégée par un tel système.
- Une liste non exhaustive d'algorithme de chiffrements symétriques datant de la période "moderne" : DES (n'est plus utilisé), 3DES (variante du précédent), AES (le **standard actuel** en trois déclinaisons : 128, 192 et 256 bits), RC4, RC5 et d'autres encore.
- Rappelons que tous ces algorithmes symétriques nécessitent que les participants à l'échange de données disposent, au préalable, d'une même **clé secrète partagée**. C'est pour cela qu'ils ne sont presque jamais utilisés seuls.

Quelques exemples d'algorithmes symétriques

- Le chiffre de *Vernam* (1926) est un chiffre de *Vigenère* où la clé secrète, de même longueur que le message à chiffrer, est choisie aléatoirement et n'est utilisée qu'une fois. Cet algorithme est théoriquement **incassable** mais sa mise en œuvre est difficile en pratique. Néanmoins, la ligne téléphonique entre le Kremlin et la Maison Blanche a été, à un moment, protégée par un tel système.
- Une liste non exhaustive d'algorithme de chiffrements symétriques datant de la période "moderne" : DES (n'est plus utilisé), 3DES (variante du précédent), AES (le **standard actuel** en trois déclinaisons : 128, 192 et 256 bits), RC4, RC5 et d'autres encore.
- Rappelons que tous ces algorithmes symétriques nécessitent que les participants à l'échange de données disposent, au préalable, d'une même **clé secrète partagée**. C'est pour cela qu'ils ne sont presque jamais utilisés seuls.

Algorithmes asymétriques

- Ces algorithmes ont besoin de deux clés :
  - une clé **publique** qui sert au **chiffrement** ou parfois aussi à la **vérification de signature**,
  - une clé **privée** qui sert au **déchiffrement** ou parfois aussi à la **signature**.
- Le plus connu d'entre eux : RSA (*Rivest*, *Shamir* et *Adleman* en 1977). Il permet le chiffrement et la signature.
- Le protocole d'échanges de clés Diffie-Hellman (en 1976), protocole à l'origine de la cryptographie moderne.
- La cryptographie sur les courbes elliptiques (*Koblitz* et *Miller* en 1985) : ECC (*Elliptic Curve Cryptography*).
- L'algorithme de signature numérique DSA (*Digital Signature Algorithm*) proposé par le NIST (*National Institute of Standards and Technology*) en 1991.
- Et bien d'autres encore...

Algorithmes asymétriques

- Ces algorithmes ont besoin de deux clés :
  - une clé **publique** qui sert au **chiffrement** ou parfois aussi à la **vérification de signature**,
  - une clé **privée** qui sert au **déchiffrement** ou parfois aussi à la **signature**.
- Le plus connu d'entre eux : RSA (*Rivest*, *Shamir* et *Adleman* en 1977). Il permet le chiffrement et la signature.
- Le protocole d'échanges de clés Diffie-Hellman (en 1976), protocole à l'origine de la cryptographie moderne.
- La cryptographie sur les courbes elliptiques (*Koblitz* et *Miller* en 1985) : ECC (*Elliptic Curve Cryptography*).
- L'algorithme de signature numérique DSA (*Digital Signature Algorithm*) proposé par le NIST (*National Institute of Standards and Technology*) en 1991.
- Et bien d'autres encore...

Algorithmes asymétriques

- Ces algorithmes ont besoin de deux clés :
  - une clé **publique** qui sert au **chiffrement** ou parfois aussi à la **vérification de signature**,
  - une clé **privée** qui sert au **déchiffrement** ou parfois aussi à la **signature**.
- Le plus connu d'entre eux : RSA (*Rivest*, *Shamir* et *Adleman* en 1977). Il permet le chiffrement et la signature.
- Le protocole d'échanges de clés Diffie-Hellman (en 1976), protocole à l'origine de la cryptographie moderne.
- La cryptographie sur les courbes elliptiques (*Koblitz* et *Miller* en 1985) : ECC (*Elliptic Curve Cryptography*).
- L'algorithme de signature numérique DSA (*Digital Signature Algorithm*) proposé par le NIST (*National Institute of Standards and Technology*) en 1991.
- Et bien d'autres encore...

Algorithmes asymétriques

- Ces algorithmes ont besoin de deux clés :
  - une clé **publique** qui sert au **chiffrement** ou parfois aussi à la **vérification de signature**,
  - une clé **privée** qui sert au **déchiffrement** ou parfois aussi à la **signature**.
- Le plus connu d'entre eux : RSA (*Rivest*, *Shamir* et *Adleman* en 1977). Il permet le chiffrement et la signature.
- Le protocole d'échanges de clés Diffie-Hellman (en 1976), protocole à l'origine de la cryptographie moderne.
- La cryptographie sur les courbes elliptiques (*Koblitz* et *Miller* en 1985) : ECC (*Elliptic Curve Cryptography*).
- L'algorithme de signature numérique DSA (*Digital Signature Algorithm*) proposé par le NIST (*National Institute of Standards and Technology*) en 1991.
- Et bien d'autres encore...

Algorithmes asymétriques

- Ces algorithmes ont besoin de deux clés :
  - une clé **publique** qui sert au **chiffrement** ou parfois aussi à la **vérification de signature**,
  - une clé **privée** qui sert au **déchiffrement** ou parfois aussi à la **signature**.
- Le plus connu d'entre eux : RSA (*Rivest*, *Shamir* et *Adleman* en 1977). Il permet le chiffrement et la signature.
- Le protocole d'échanges de clés Diffie-Hellman (en 1976), protocole à l'origine de la cryptographie moderne.
- La cryptographie sur les courbes elliptiques (*Koblitz* et *Miller* en 1985) : ECC (*Elliptic Curve Cryptography*).
- L'algorithme de signature numérique DSA (*Digital Signature Algorithm*) proposé par le NIST (*National Institute of Standards and Technology*) en 1991.
- Et bien d'autres encore...

Algorithmes asymétriques

- Ces algorithmes ont besoin de deux clés :
  - une clé **publique** qui sert au **chiffrement** ou parfois aussi à la **vérification de signature**,
  - une clé **privée** qui sert au **déchiffrement** ou parfois aussi à la **signature**.
- Le plus connu d'entre eux : RSA (*Rivest*, *Shamir* et *Adleman* en 1977). Il permet le chiffrement et la signature.
- Le protocole d'échanges de clés Diffie-Hellman (en 1976), protocole à l'origine de la cryptographie moderne.
- La cryptographie sur les courbes elliptiques (*Koblitz* et *Miller* en 1985) : ECC (*Elliptic Curve Cryptography*).
- L'algorithme de signature numérique DSA (*Digital Signature Algorithm*) proposé par le NIST (*National Institute of Standards and Technology*) en 1991.
- Et bien d'autres encore...

Algorithmes asymétriques

- Ces algorithmes ont besoin de deux clés :
  - une clé **publique** qui sert au **chiffrement** ou parfois aussi à la **vérification de signature**,
  - une clé **privée** qui sert au **déchiffrement** ou parfois aussi à la **signature**.
- Le plus connu d'entre eux : RSA (*Rivest*, *Shamir* et *Adleman* en 1977). Il permet le chiffrement et la signature.
- Le protocole d'échanges de clés Diffie-Hellman (en 1976), protocole à l'origine de la cryptographie moderne.
- La cryptographie sur les courbes elliptiques (*Koblitz* et *Miller* en 1985) : ECC (*Elliptic Curve Cryptography*).
- L'algorithme de signature numérique DSA (*Digital Signature Algorithm*) proposé par le NIST (*National Institute of Standards and Technology*) en 1991.
- Et bien d'autres encore...

Algorithmes asymétriques

- Ces algorithmes ont besoin de deux clés :
  - une clé **publique** qui sert au **chiffrement** ou parfois aussi à la **vérification de signature**,
  - une clé **privée** qui sert au **déchiffrement** ou parfois aussi à la **signature**.
- Le plus connu d'entre eux : RSA (*Rivest*, *Shamir* et *Adleman* en 1977). Il permet le chiffrement et la signature.
- Le protocole d'échanges de clés Diffie-Hellman (en 1976), protocole à l'origine de la cryptographie moderne.
- La cryptographie sur les courbes elliptiques (*Koblitz* et *Miller* en 1985) : ECC (*Elliptic Curve Cryptography*).
- L'algorithme de signature numérique DSA (*Digital Signature Algorithm*) proposé par le NIST (*National Institute of Standards and Technology*) en 1991.
- Et bien d'autres encore...

Fonctions de hachage

- Ces fonctions sont un outil fondamental de la cryptographie moderne.
- Une fonction de hachage est une fonction qui prend en entrée des données de longueur quelconque et rend une **valeur de taille fixe** et dont la propriété fondamentale est d'être **difficilement inversible**.
- On peut la voir, par exemple, comme une fonction définie sur les entiers naturels et à valeurs dans l'ensemble des entiers de n bits, $f : \mathbb{N} \to \{0, \ldots, 2^n - 1\}$, avec $f(x)$ facile à calculer et $f^{-1}(x)$ difficile à calculer.
- Un des premiers usages a été le stockage des mots de passe des utilisateurs : au lieu de stocker MDP, on stocke $f(MDP)$. Comme $f^{-1}(MDP)$ est difficile à calculer, on ne peut pas retrouver (en un temps raisonnable) le mot de passe MDP.
- Par exemple, dans le fichier `/etc/passwd` sous UNIX :

Fonctions de hachage

- Ces fonctions sont un outil fondamental de la cryptographie moderne.
- Une fonction de hachage est une fonction qui prend en entrée des données de longueur quelconque et rend une **valeur de taille fixe** et dont la propriété fondamentale est d'être **difficilement inversible**.
- On peut la voir, par exemple, comme une fonction définie sur les entiers naturels et à valeurs dans l'ensemble des entiers de n bits, $f : \mathbb{N} \to \{0, \ldots, 2^n - 1\}$, avec $f(x)$ facile à calculer et $f^{-1}(x)$ difficile à calculer.
- Un des premiers usages a été le stockage des mots de passe des utilisateurs : au lieu de stocker MDP, on stocke $f(MDP)$. Comme $f^{-1}(MDP)$ est difficile à calculer, on ne peut pas retrouver (en un temps raisonnable) le mot de passe MDP.
- Par exemple, dans le fichier `/etc/passwd` sous UNIX :

Fonctions de hachage

- Ces fonctions sont un outil fondamental de la cryptographie moderne.
- Une fonction de hachage est une fonction qui prend en entrée des données de longueur quelconque et rend une **valeur de taille fixe** et dont la propriété fondamentale est d'être **difficilement inversible**.
- On peut la voir, par exemple, comme une fonction définie sur les entiers naturels et à valeurs dans l'ensemble des entiers de n bits, $f : \mathbb{N} \to \{0, \ldots, 2^n - 1\}$, avec $f(x)$ facile à calculer et $f^{-1}(x)$ difficile à calculer.
- Un des premiers usages a été le stockage des mots de passe des utilisateurs : au lieu de stocker MDP, on stocke $f(MDP)$. Comme $f^{-1}(MDP)$ est difficile à calculer, on ne peut pas retrouver (en un temps raisonnable) le mot de passe MDP.
- Par exemple, dans le fichier `/etc/passwd` sous UNIX :

Fonctions de hachage

- Ces fonctions sont un outil fondamental de la cryptographie moderne.
- Une fonction de hachage est une fonction qui prend en entrée des données de longueur quelconque et rend une **valeur de taille fixe** et dont la propriété fondamentale est d'être **difficilement inversible**.
- On peut la voir, par exemple, comme une fonction définie sur les entiers naturels et à valeurs dans l'ensemble des entiers de n bits, $f : \mathbb{N} \to \{0, \ldots, 2^n - 1\}$, avec $f(x)$ facile à calculer et $f^{-1}(x)$ difficile à calculer.
- Un des premiers usages a été le stockage des mots de passe des utilisateurs : au lieu de stocker MDP, on stocke $f(MDP)$. Comme $f^{-1}(MDP)$ est difficile à calculer, on ne peut pas retrouver (en un temps raisonnable) le mot de passe MDP.
- Par exemple, dans le fichier `/etc/passwd` sous UNIX :

Fonctions de hachage

- Ces fonctions sont un outil fondamental de la cryptographie moderne.
- Une fonction de hachage est une fonction qui prend en entrée des données de longueur quelconque et rend une **valeur de taille fixe** et dont la propriété fondamentale est d'être **difficilement inversible**.
- On peut la voir, par exemple, comme une fonction définie sur les entiers naturels et à valeurs dans l'ensemble des entiers de n bits, $f : \mathbb{N} \to \{0, \ldots, 2^n - 1\}$, avec $f(x)$ facile à calculer et $f^{-1}(x)$ difficile à calculer.
- Un des premiers usages a été le stockage des mots de passe des utilisateurs : au lieu de stocker MDP, on stocke $f(MDP)$. Comme $f^{-1}(MDP)$ est difficile à calculer, on ne peut pas retrouver (en un temps raisonnable) le mot de passe MDP.
- Par exemple, dans le fichier `/etc/passwd` sous UNIX :

Schéma de fonctionnement

Schéma de fonctionnement

Alice

Bob

Schéma de fonctionnement

Alice

Bob

Schéma de fonctionnement

Alice

Bob

Schéma de fonctionnement

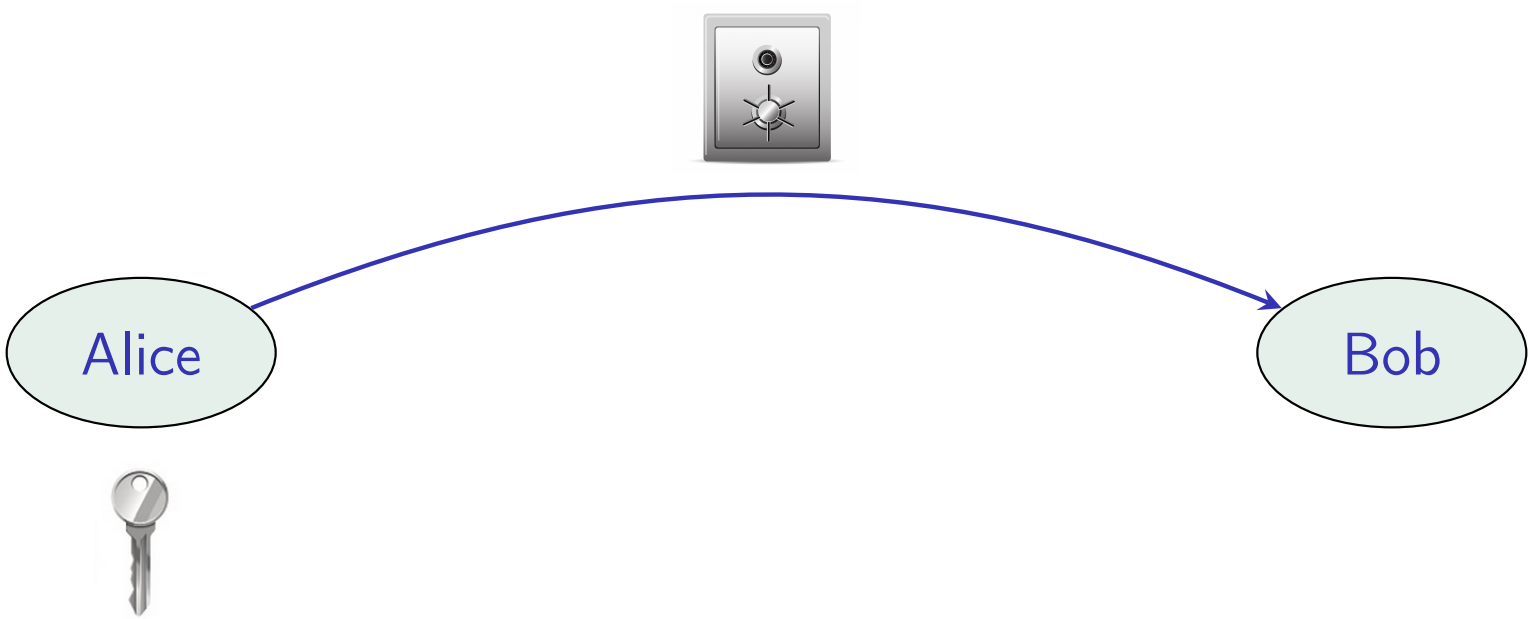


Schéma de fonctionnement

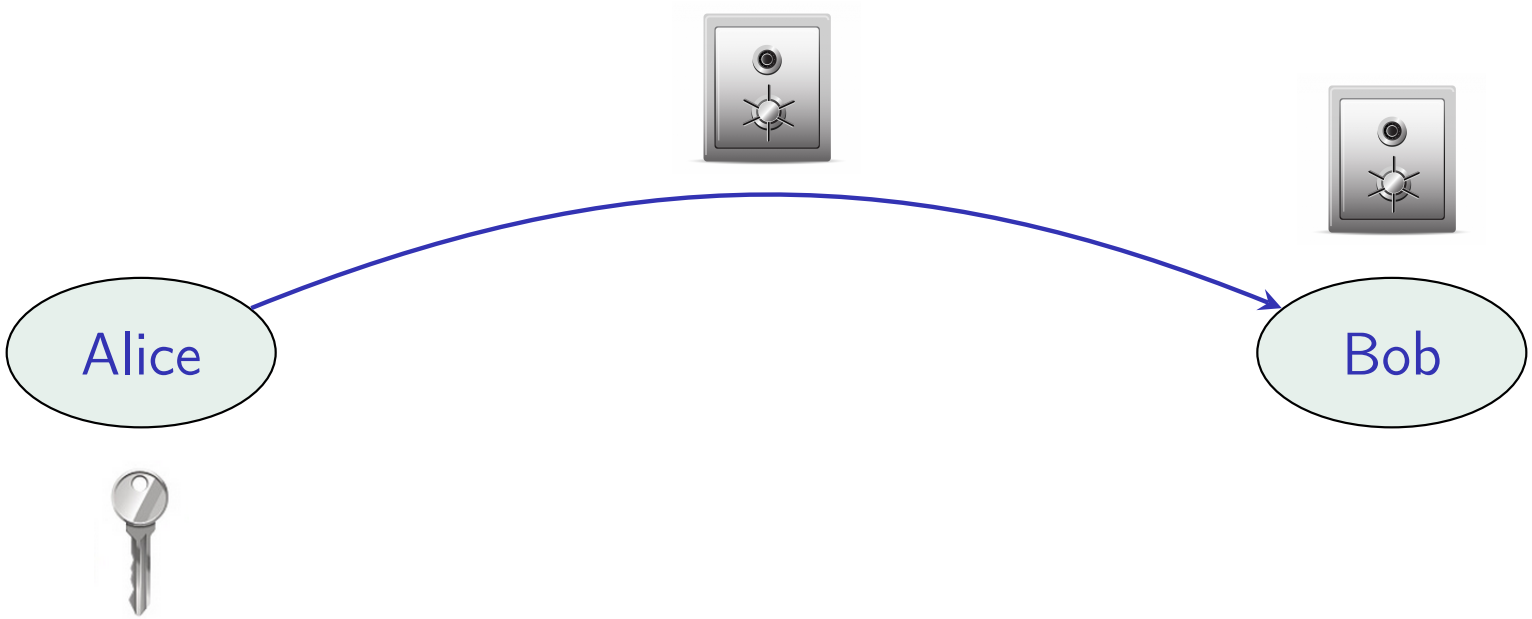


Schéma de fonctionnement

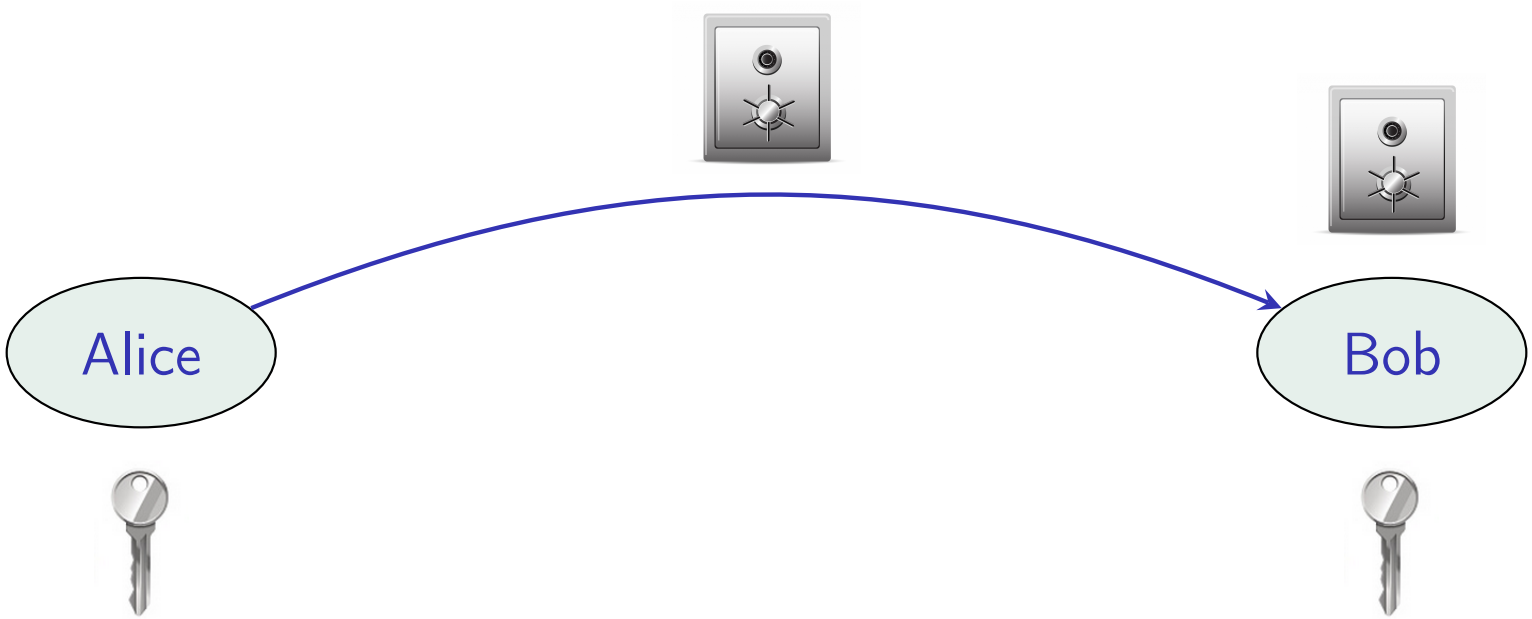


Schéma de fonctionnement

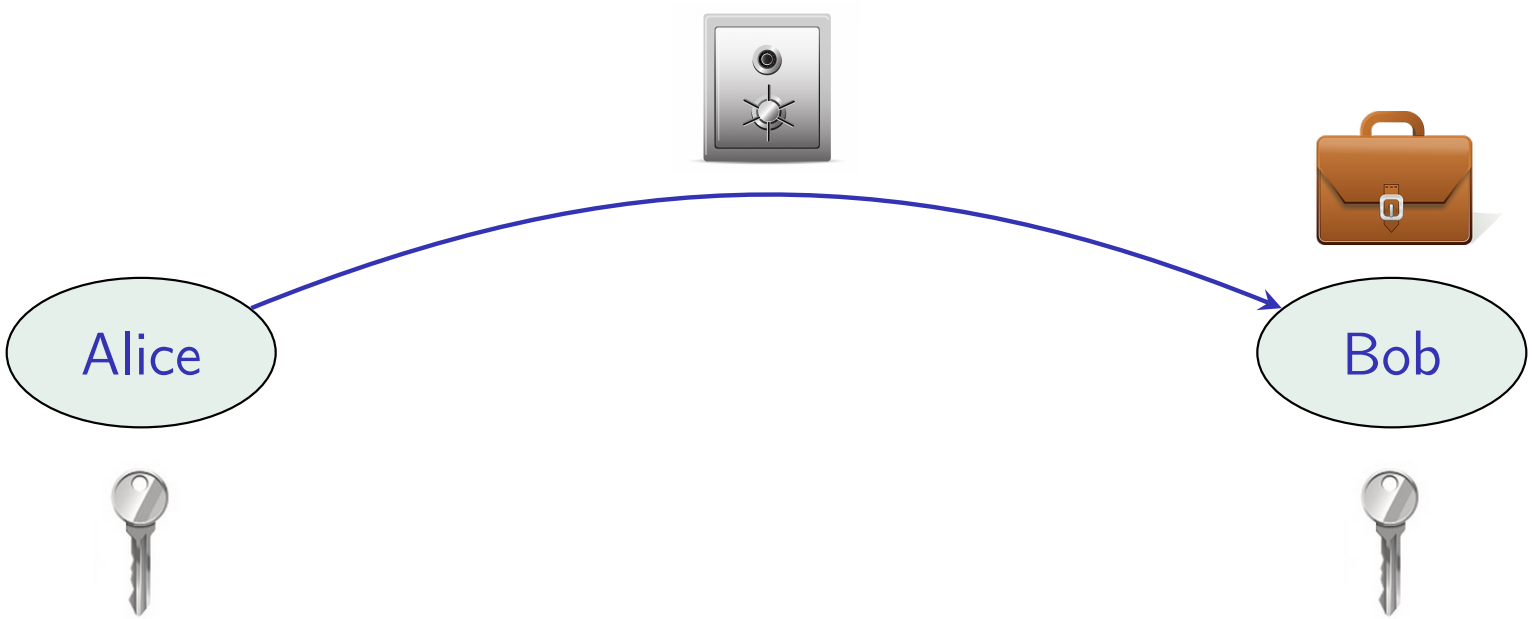


Schéma de fonctionnement

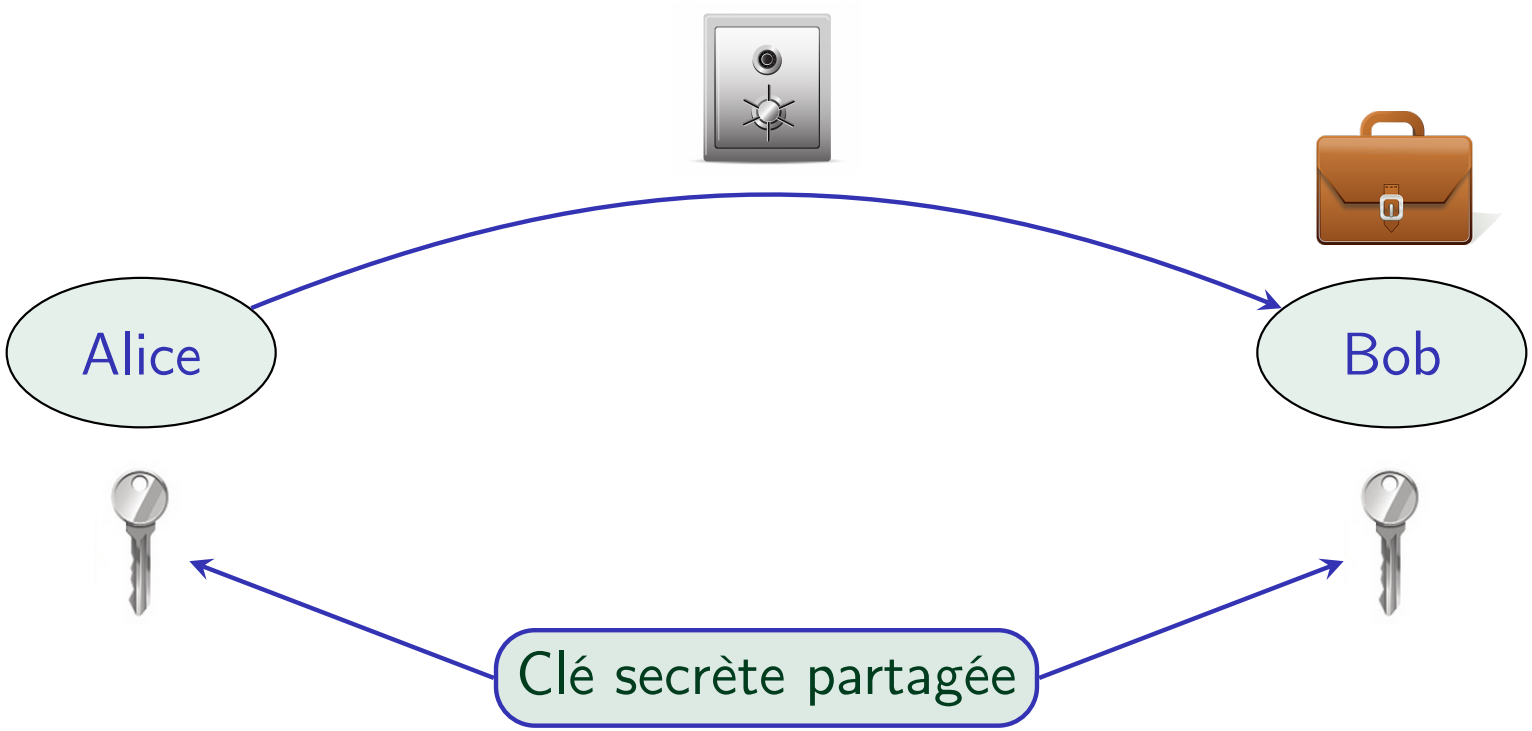


Schéma de fonctionnement

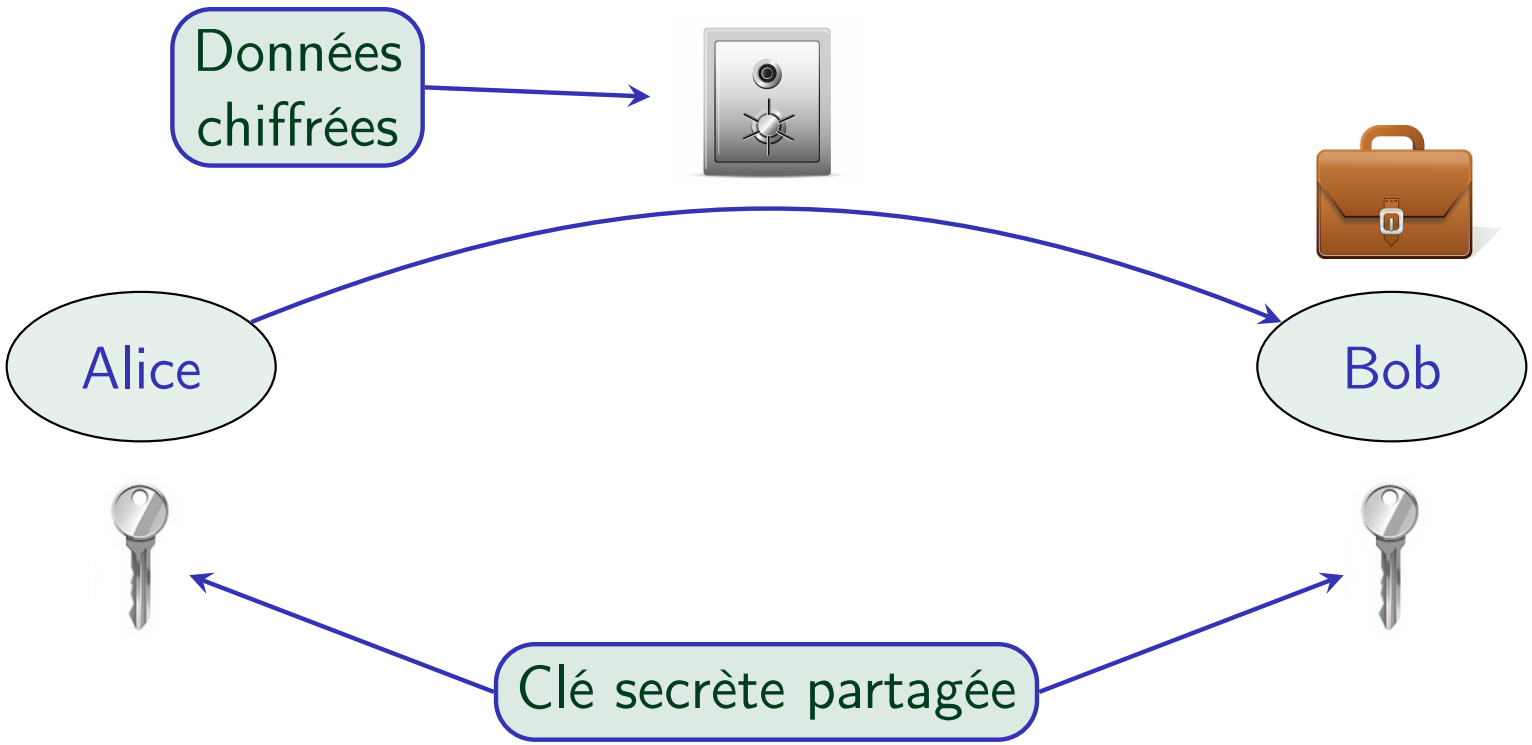


Schéma de fonctionnement

Schéma de fonctionnement

Alice

Bob

Schéma de fonctionnement

Alice

Bob

Schéma de fonctionnement

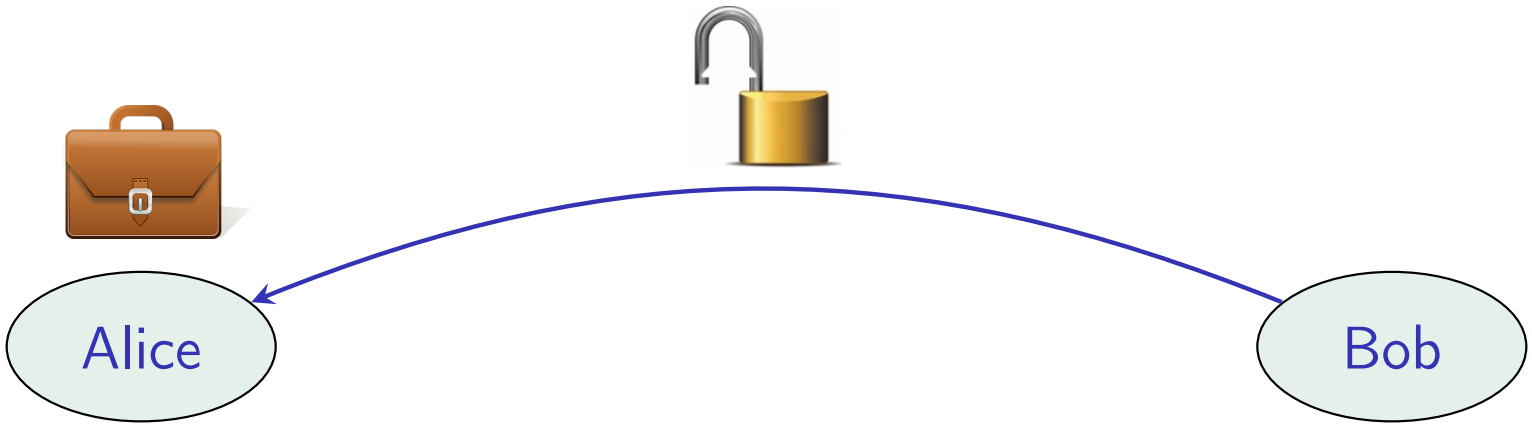


Schéma de fonctionnement

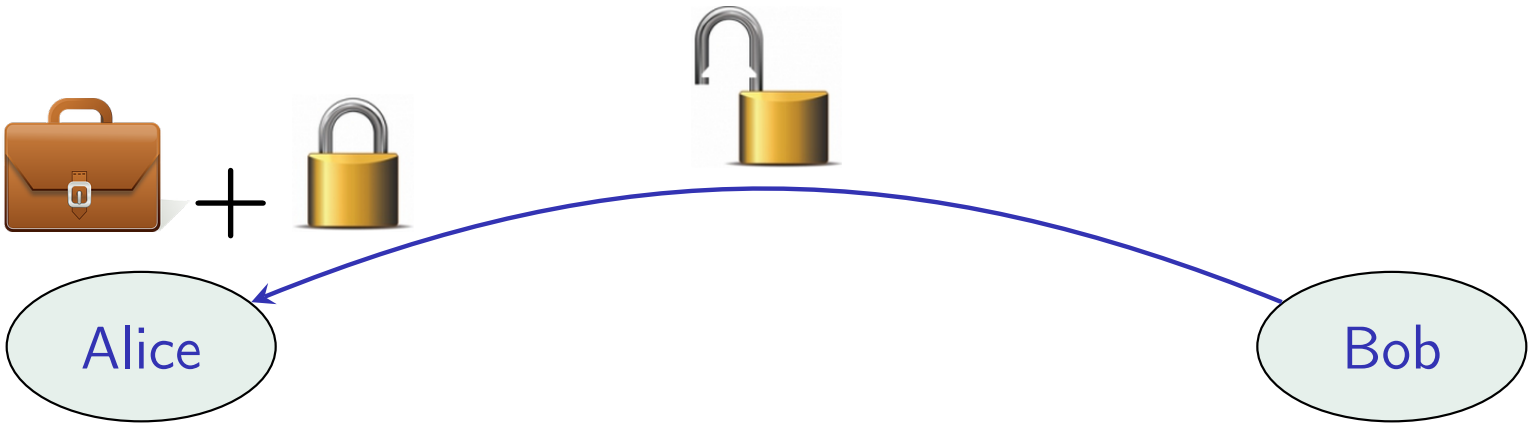


Schéma de fonctionnement

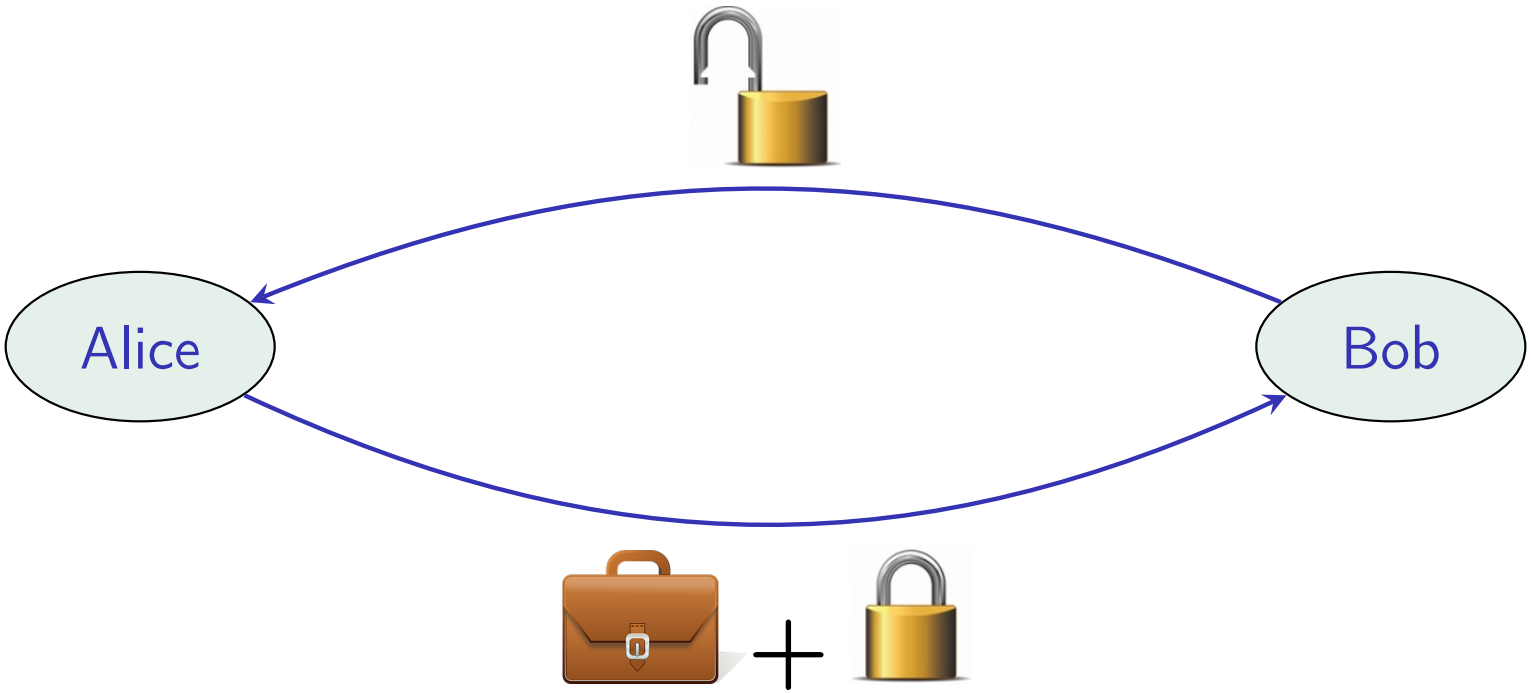


Schéma de fonctionnement

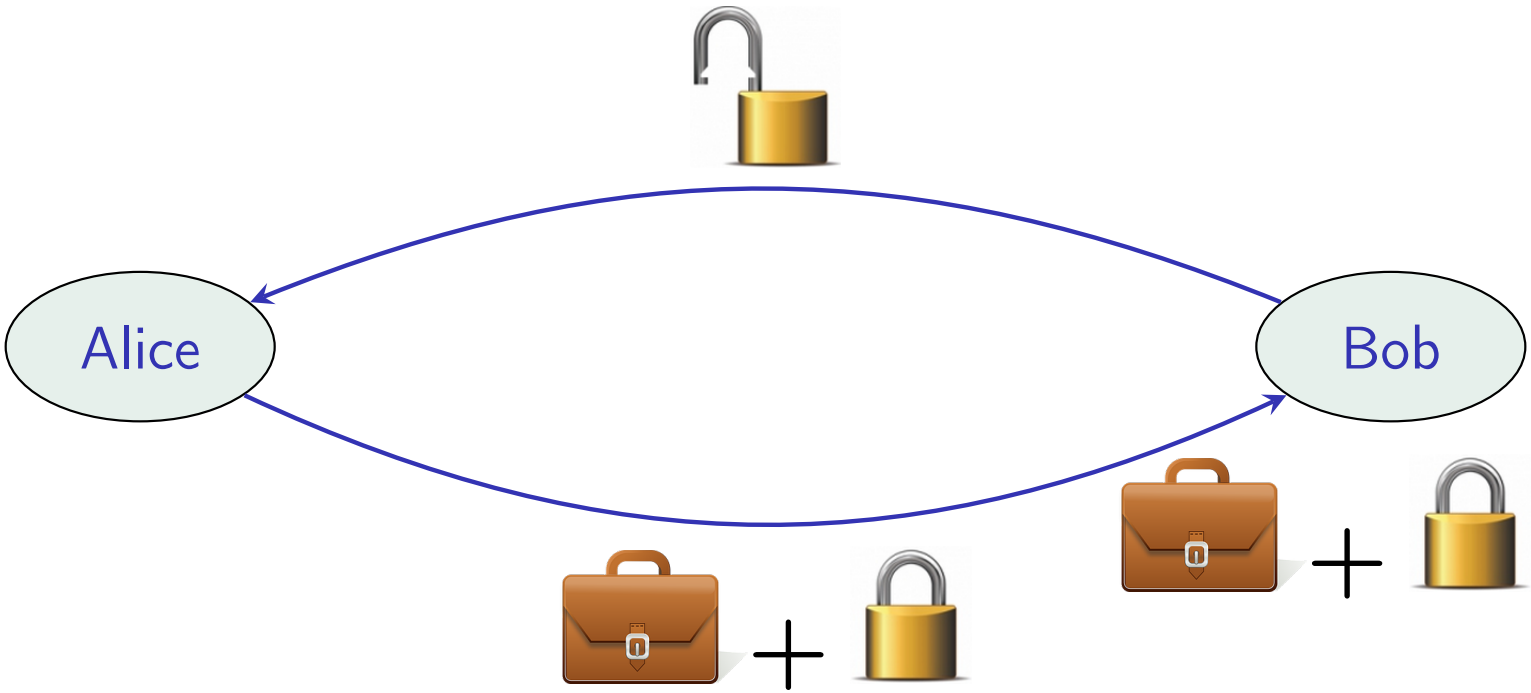


Schéma de fonctionnement

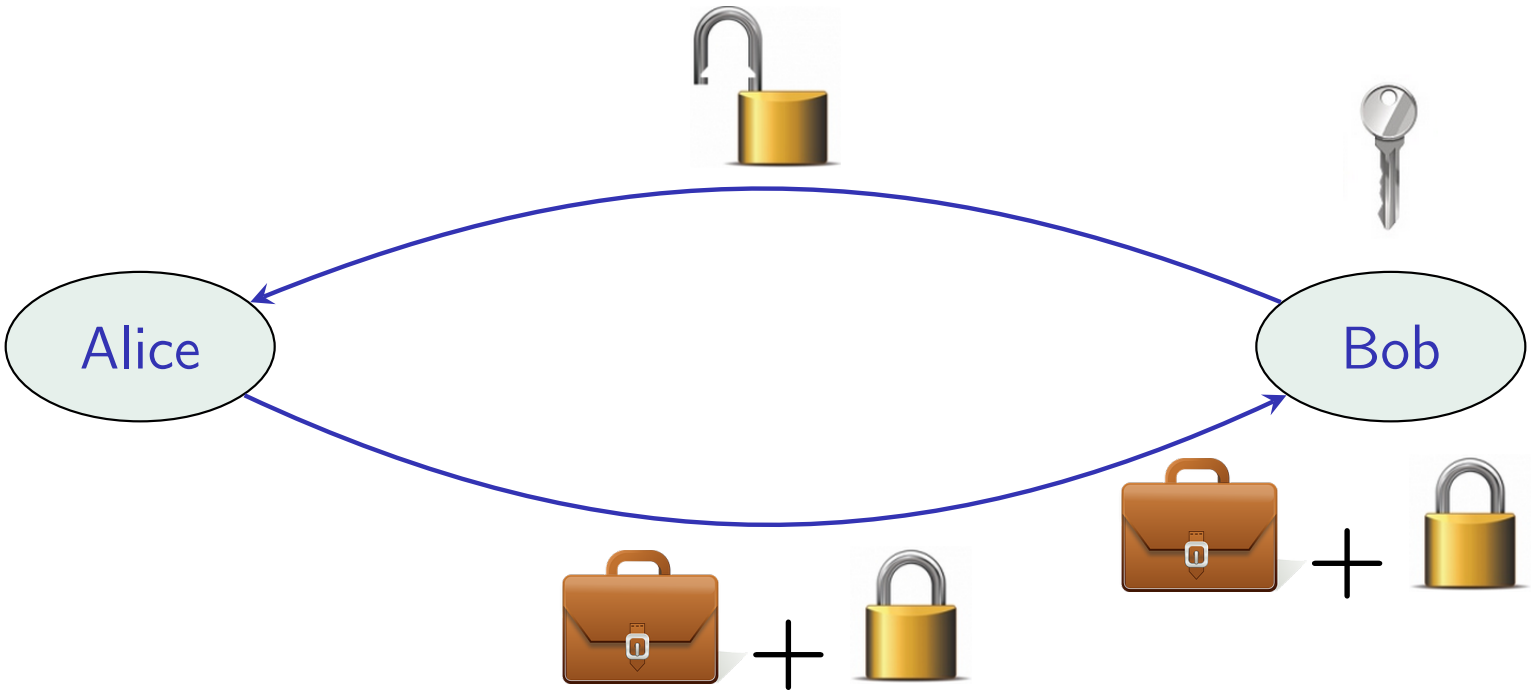


Schéma de fonctionnement

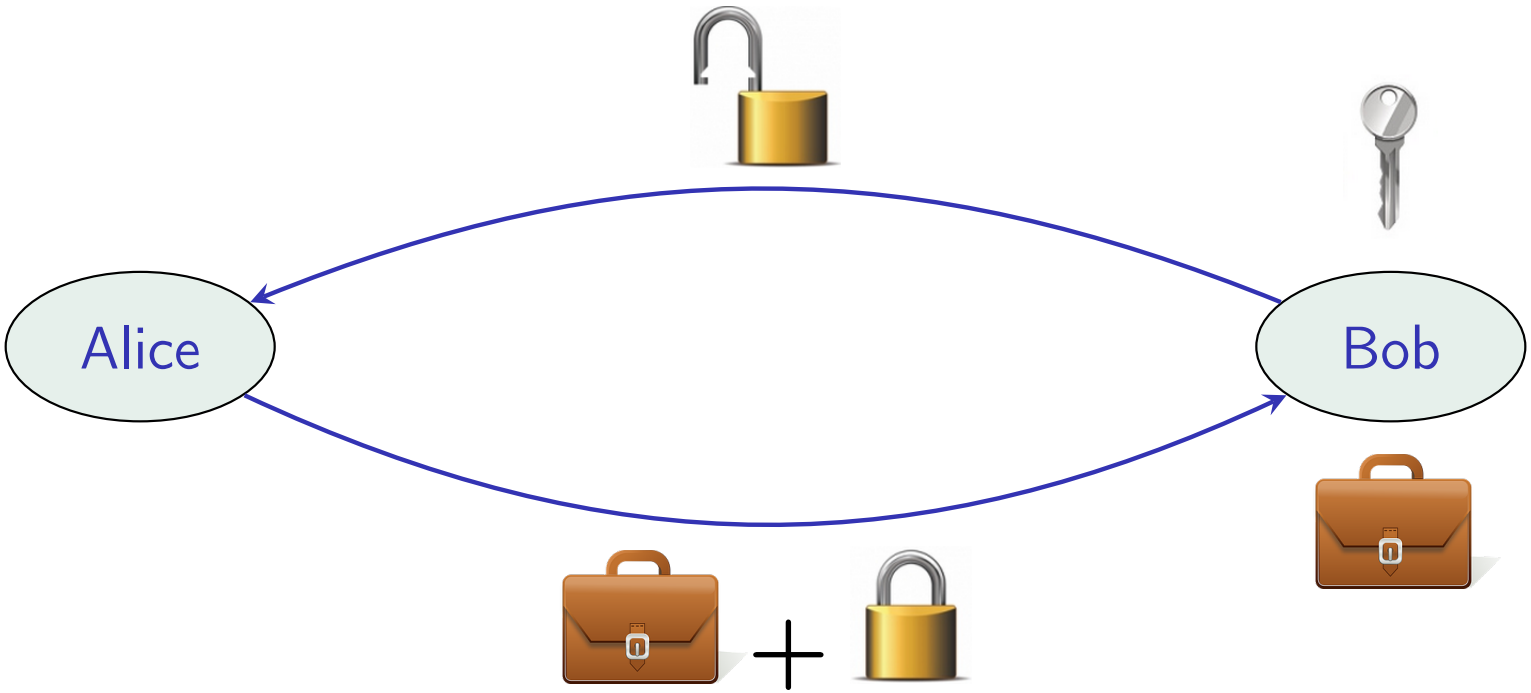


Schéma de fonctionnement

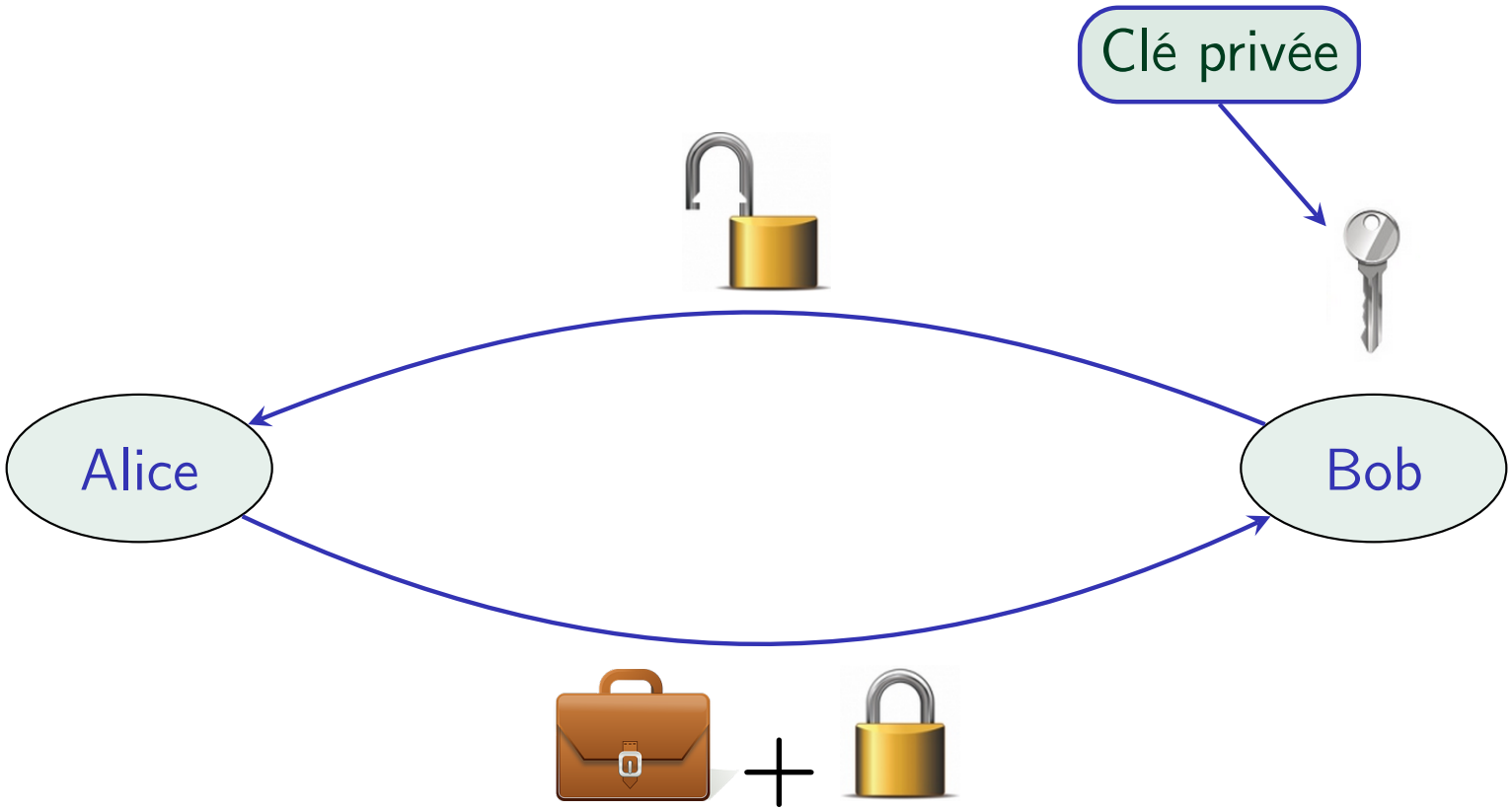


Schéma de fonctionnement

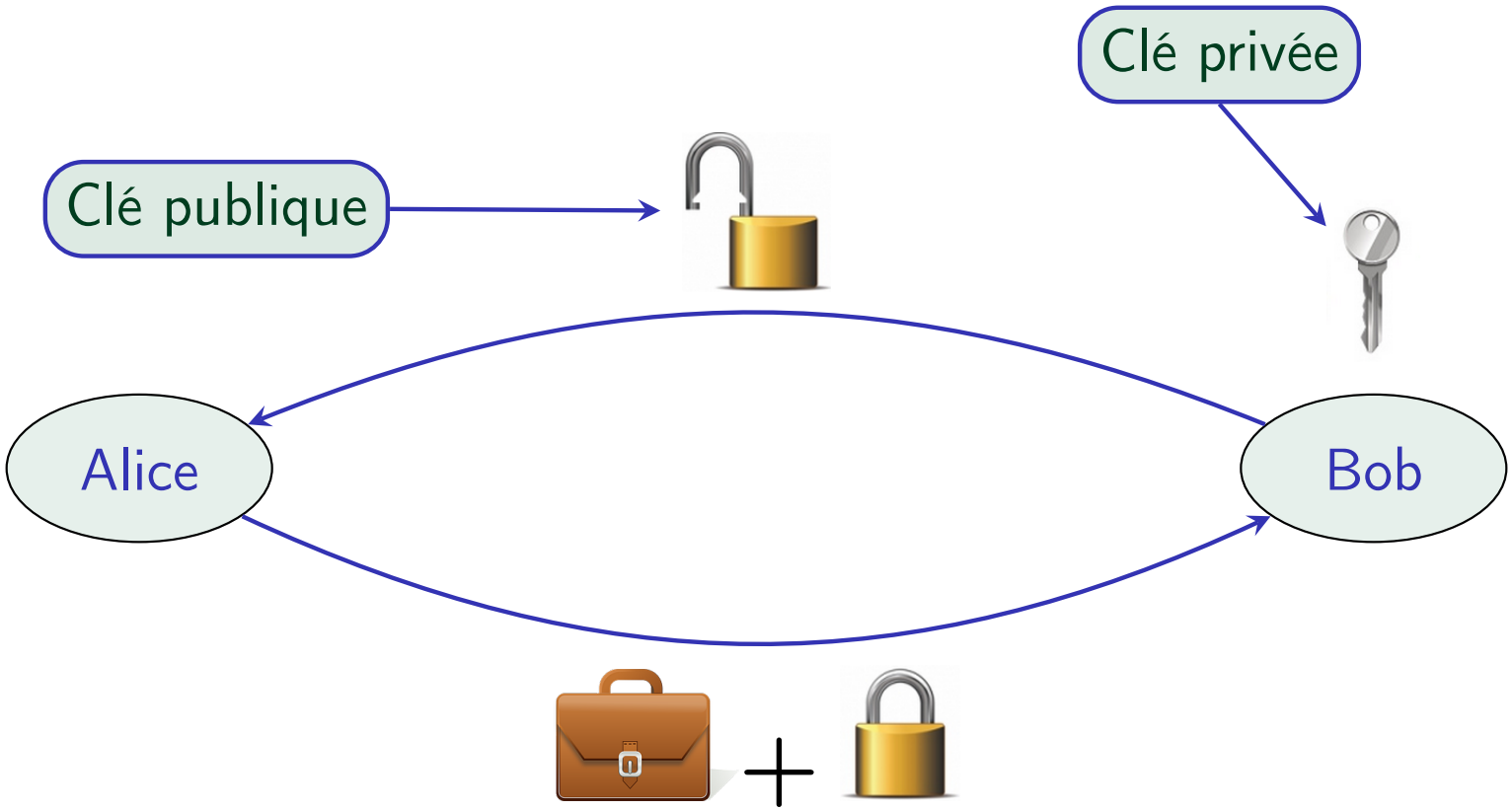
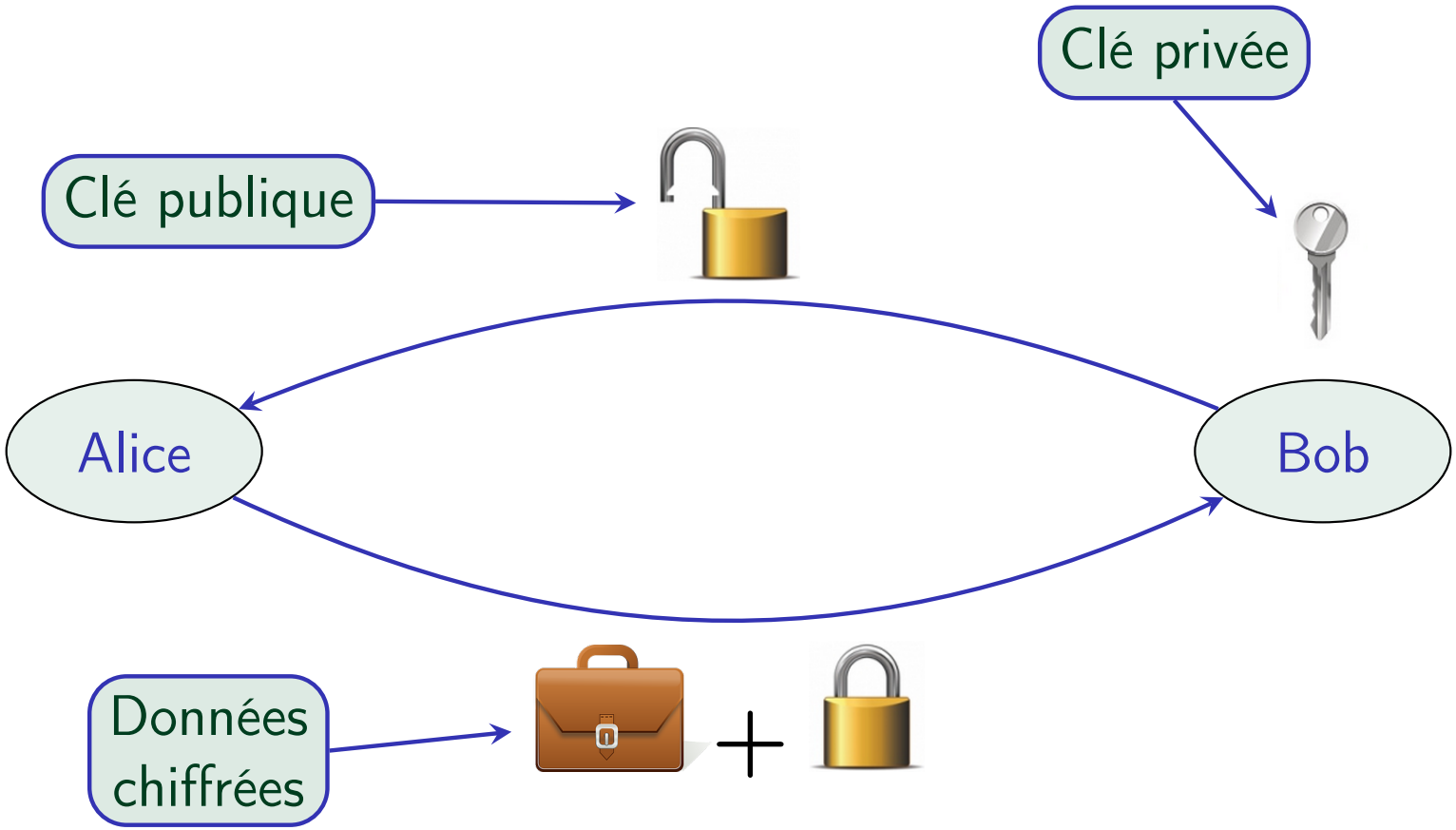
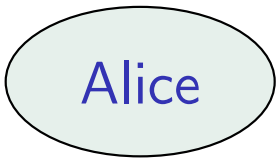
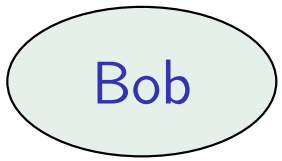


Schéma de fonctionnement



Attaque “man in the middle”





Attaque “man in the middle”

Alice

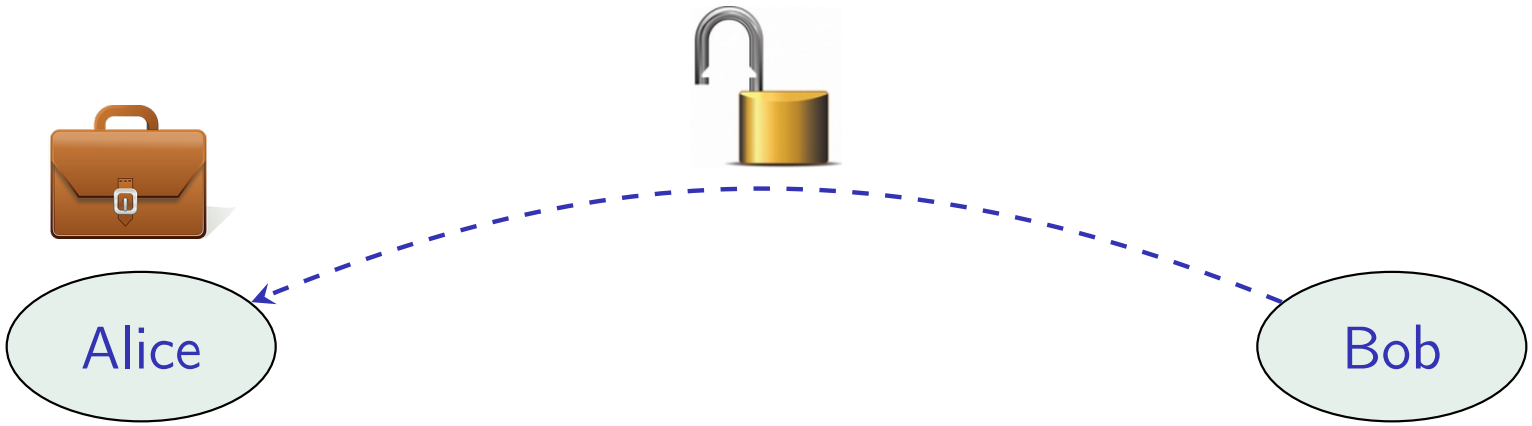
Bob

Attaque “man in the middle”

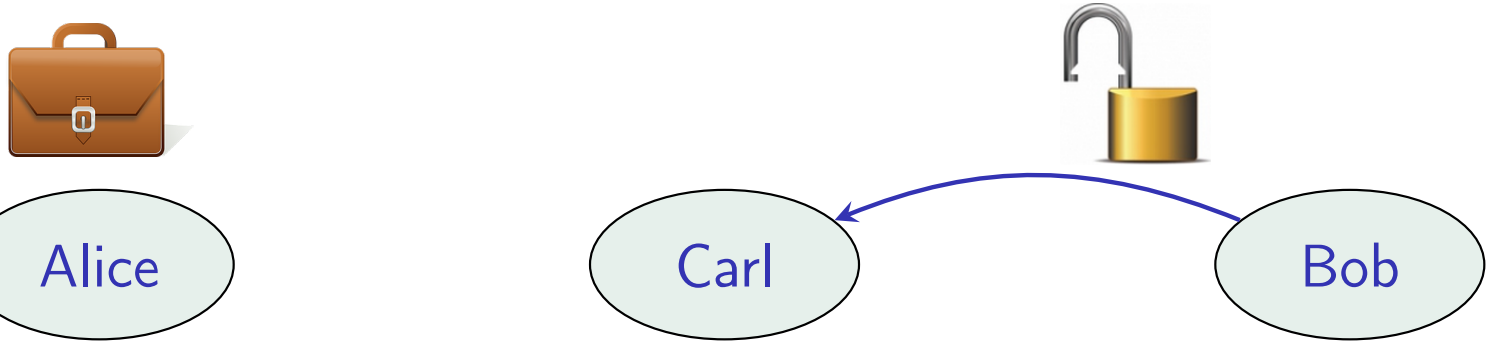
Alice

Bob

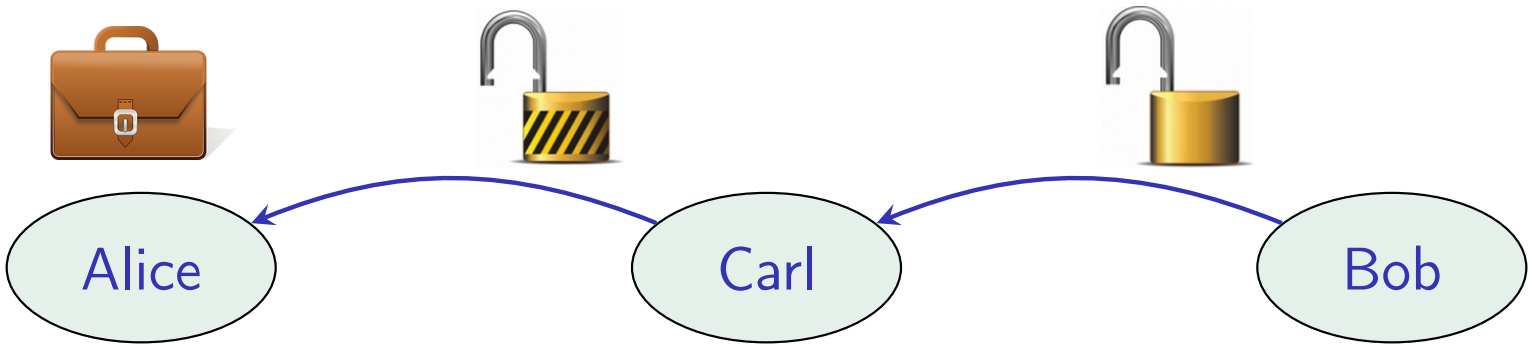
Attaque “man in the middle”



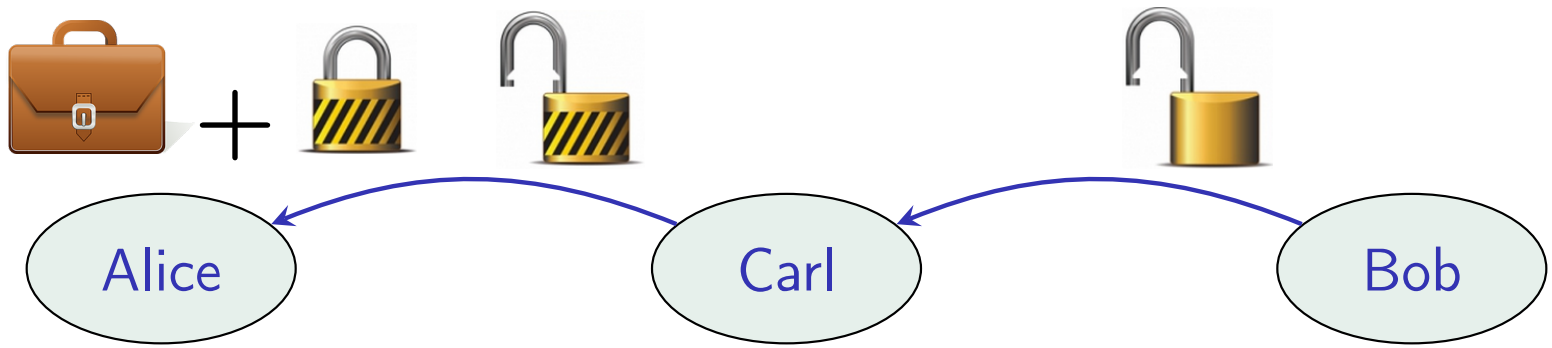
Attaque “man in the middle”



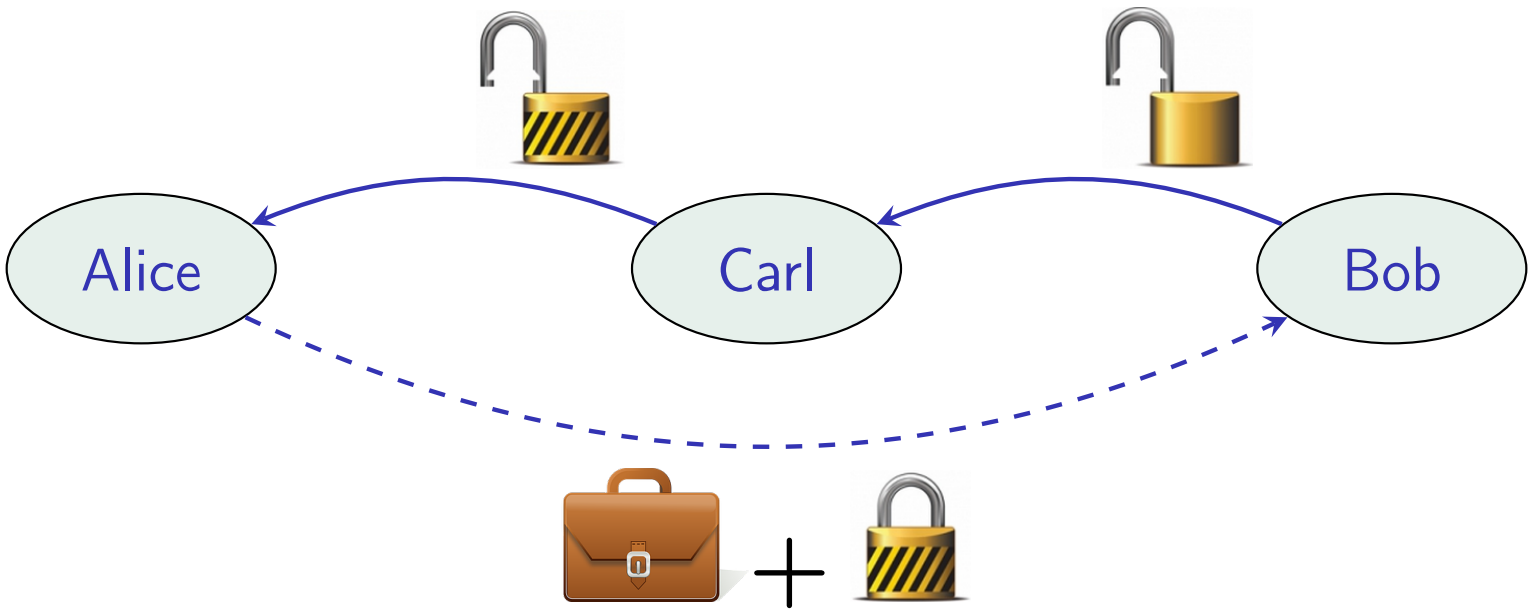
Attaque “man in the middle”



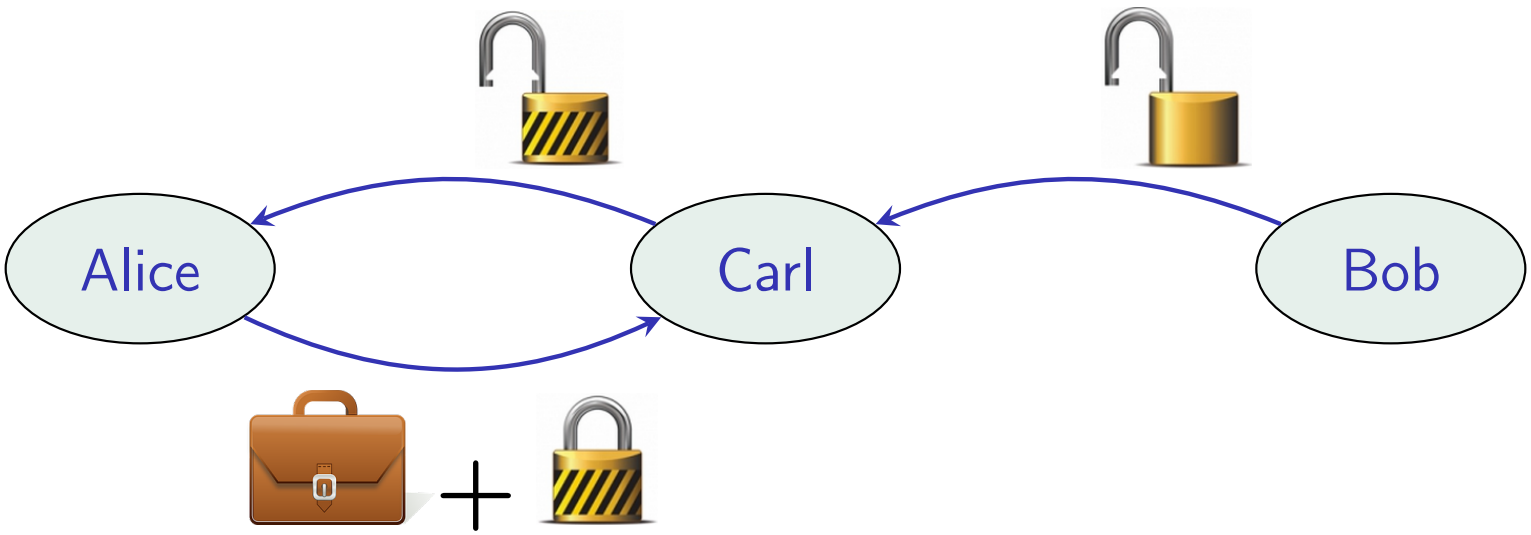
Attaque "man in the middle"



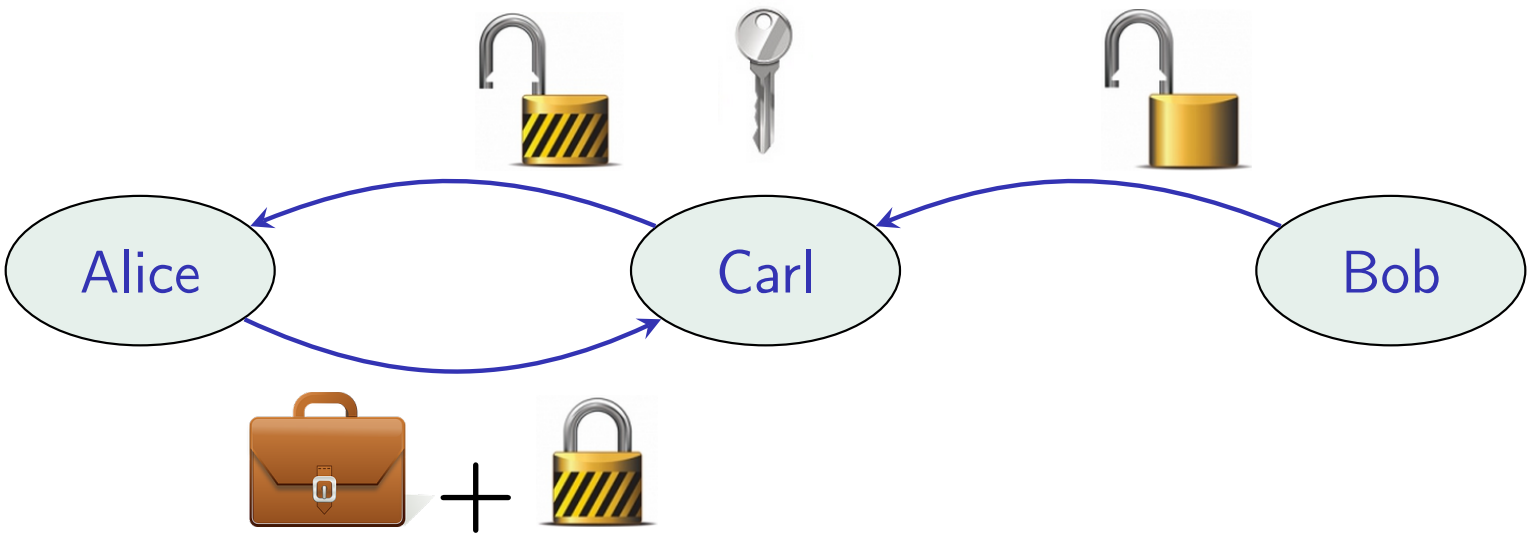
Attaque “man in the middle”



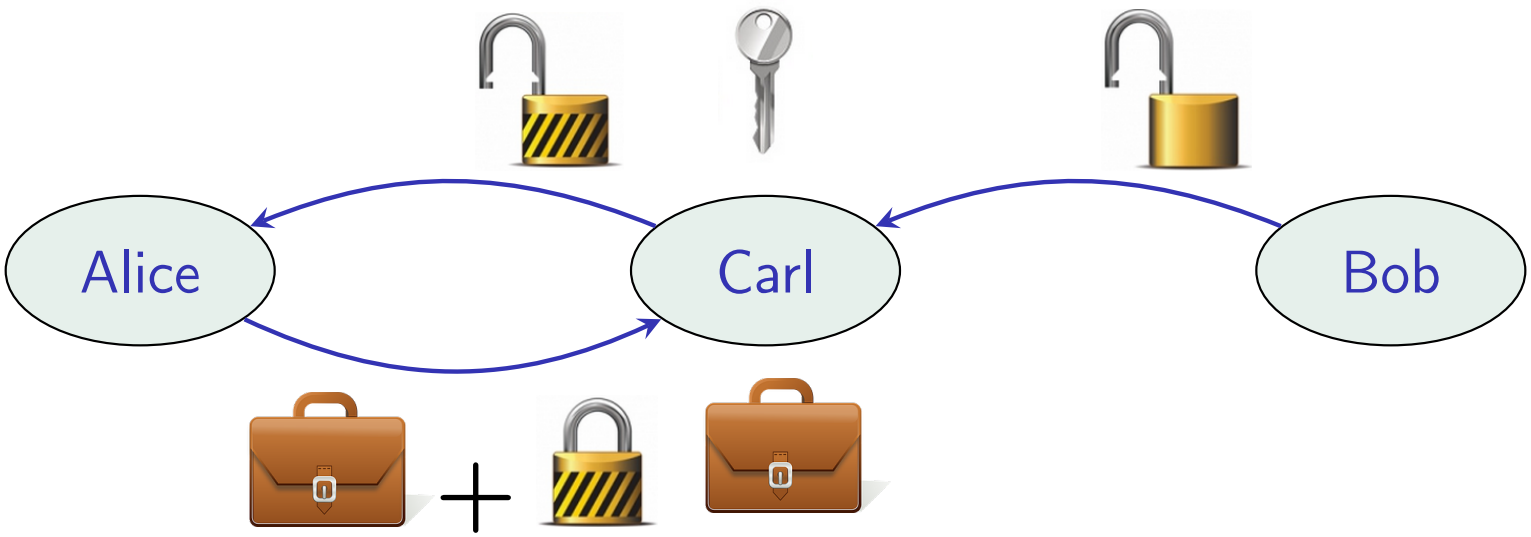
Attaque “man in the middle”



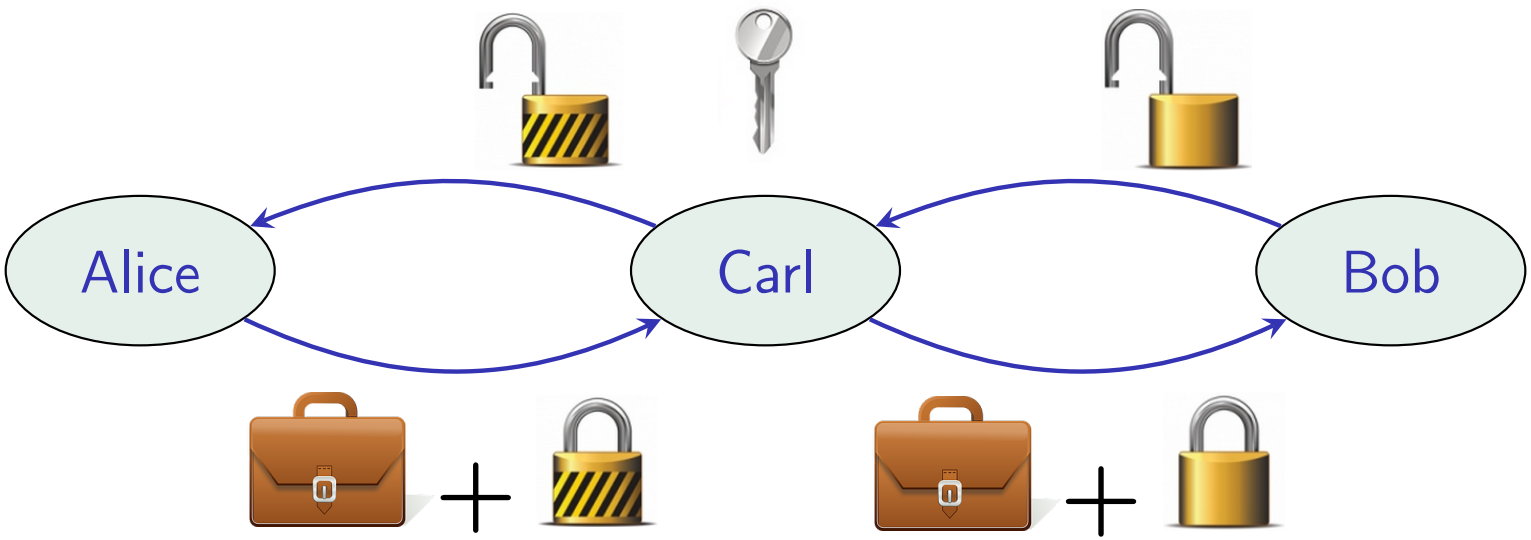
Attaque “man in the middle”



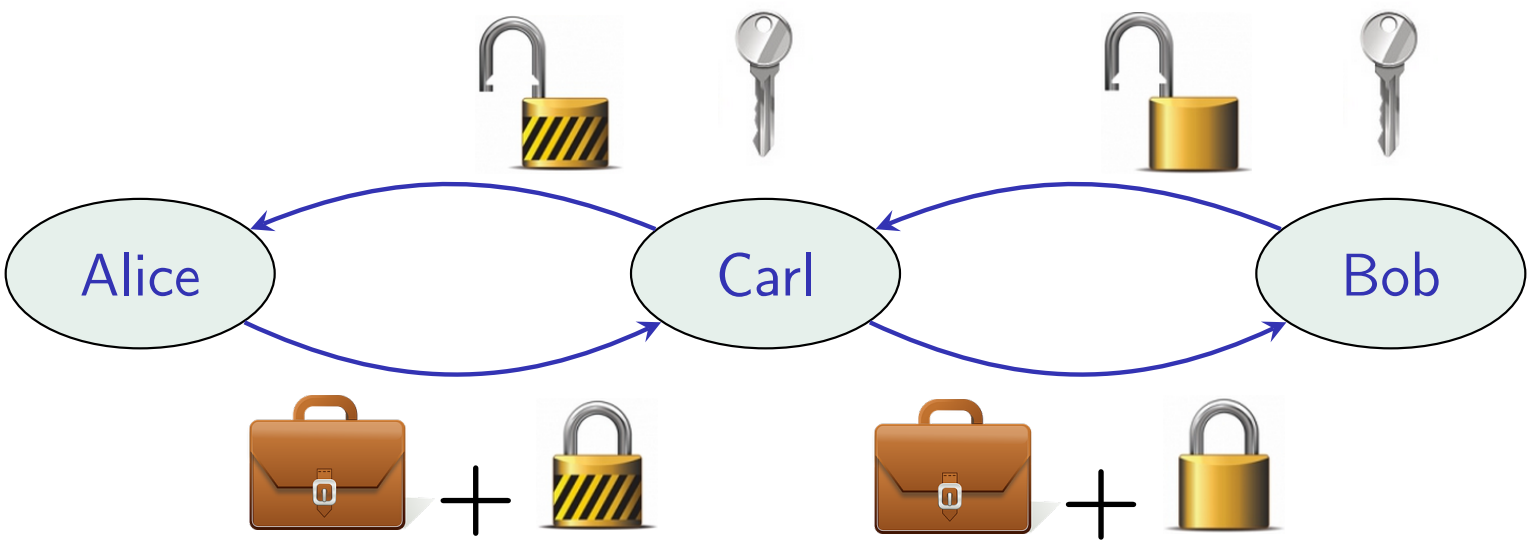
Attaque “man in the middle”



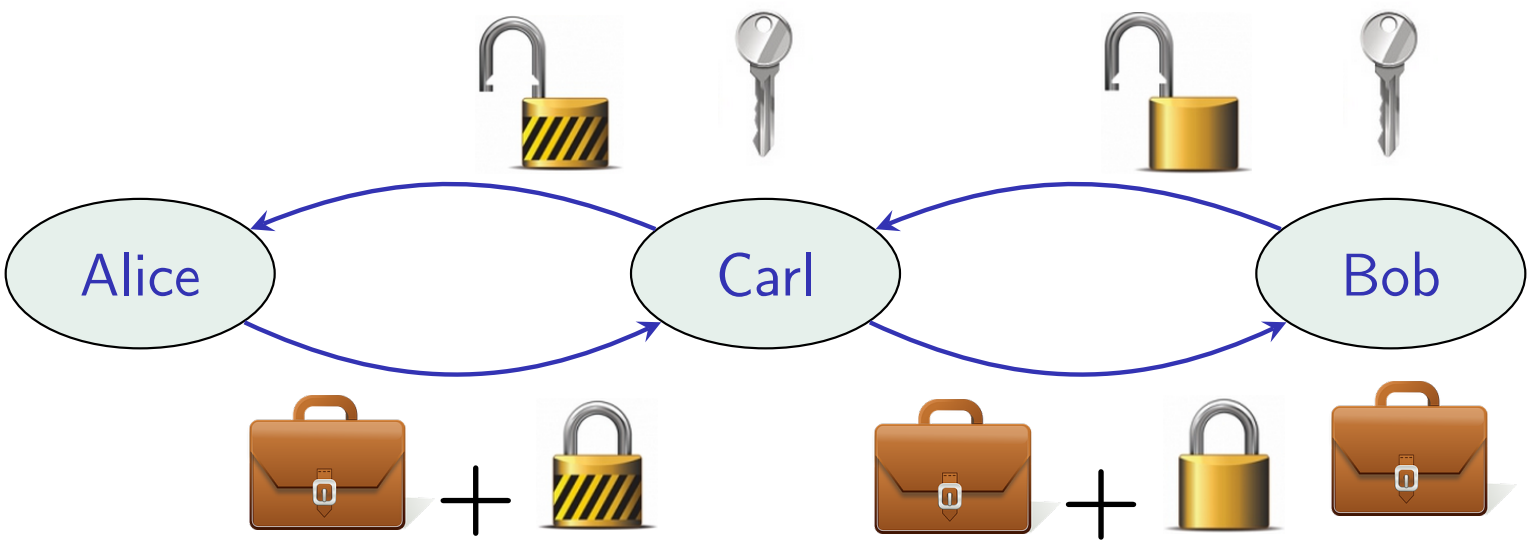
Attaque “man in the middle”



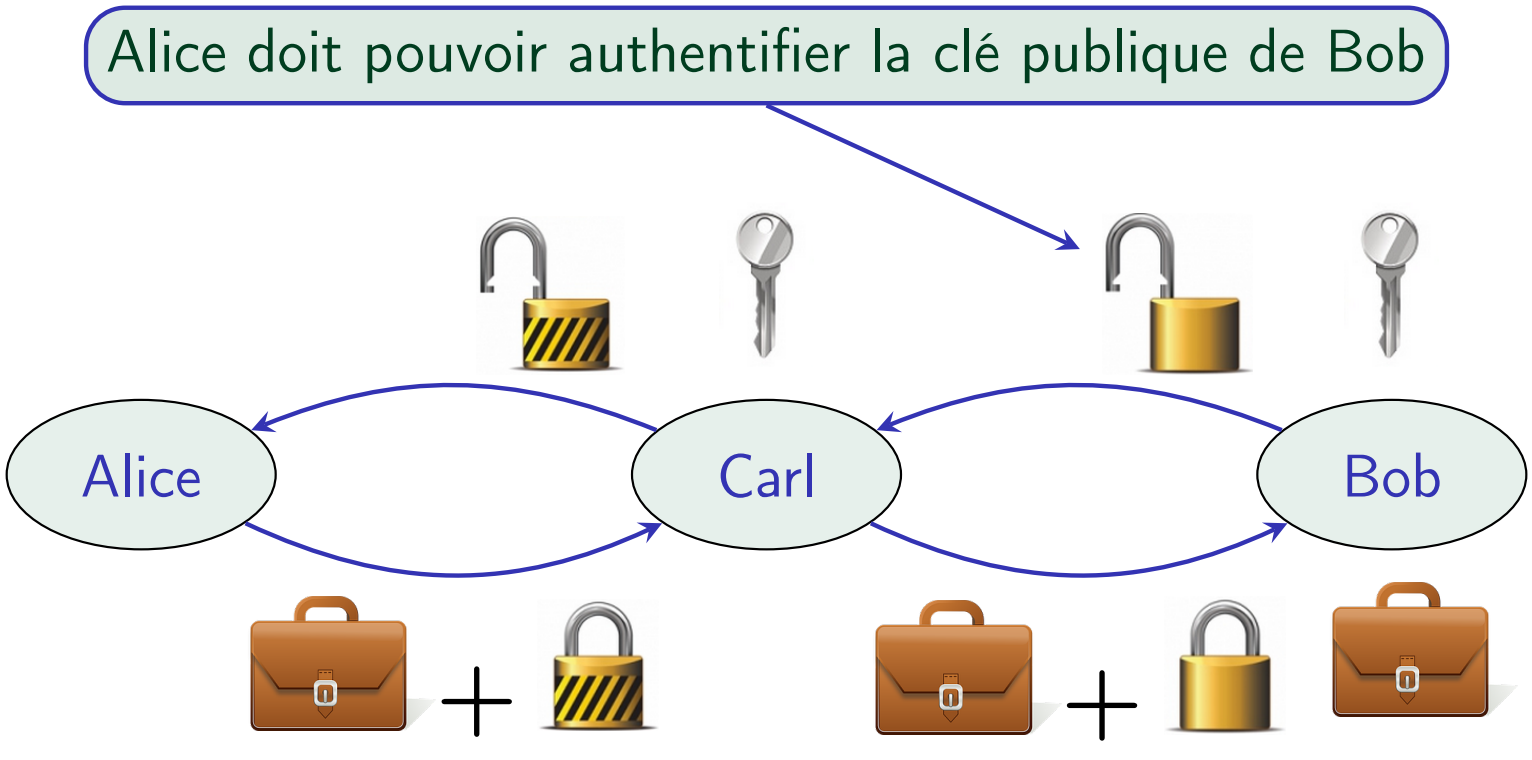
Attaque “man in the middle”



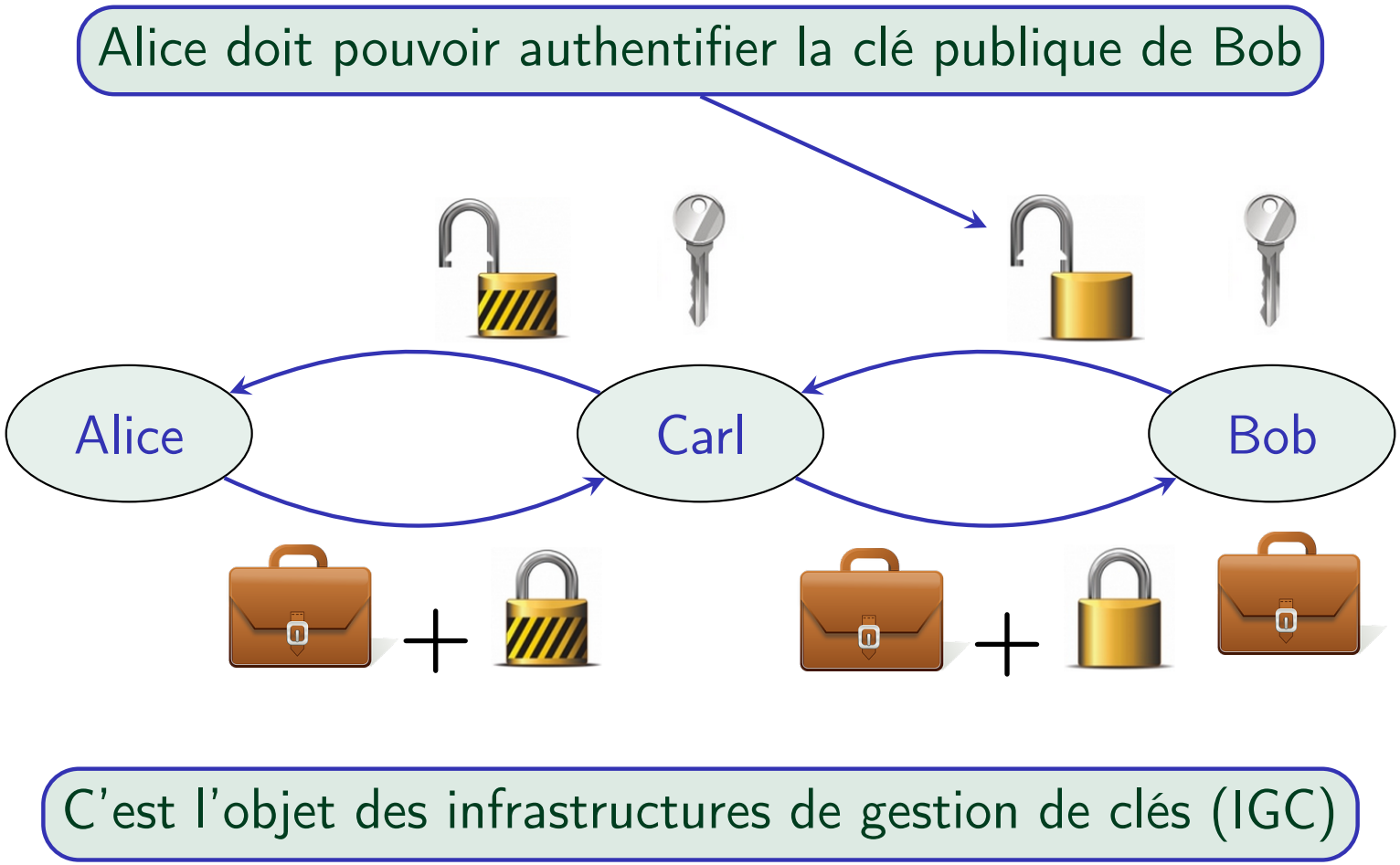
Attaque “man in the middle”



Attaque "man in the middle"



Attaque "man in the middle"



Connexion sur le site : https://fr.wikipedia.org

Cryptographie — Wikipédia - Iceweasel

https://fr.wikipedia.org/wiki/Cryptographie

Non connecté Discussion Contributions Créer un compte Se connecter

Article Discussion | Lire Modifier Modifier le code Historique | Rechercher

WIKIPÉDIA
L'encyclopédie libre

Accueil
Portails thématiques
Article au hasard
Contact

Contribuer
Débuter sur Wikipédia
Aide
Communauté
Modifications récentes
Faire un don

Outils
Pages liées
Suivi des pages liées
Importer un fichier
Pages spéciales
Adresse permanente
Information sur la page
Élément Wikidata
Citer cette page

Imprimer / exporter
Créer un livre
Télécharger comme PDF
Version imprimable

Dans d'autres projets
Wikimédia Commons
Wikiversité

Dans d'autres langues
Afrikaans
العربية
Azərbaycanca
Boarisch
Беларуская
Беларуская (тарашкевіца)
Български
বাংলা
Bosanski
Català

Cryptographie

La **cryptographie** est une des disciplines de la cryptologie s'attachant à protéger des messages (assurant confidentialité, authenticité et intégrité) en s'aidant souvent de *secrets* ou *clés*. Elle se distingue de la stéganographie qui fait passer inaperçu un message dans un autre message alors que la cryptographie rend un message inintelligible à autre que qui-de-droit.

Elle est utilisée depuis l'Antiquité, mais certaines de ses méthodes les plus importantes, comme la cryptographie asymétrique, datent de la fin du xx^{e} siècle.

La machine de Lorenz utilisée par les Allemands durant la Seconde Guerre mondiale pour chiffrer les communications militaires de haut niveau entre le quartier-général du Führer et les quartiers-généraux des groupes d'armées

Sommaire [masquer]

Étymologie et vocabulaire [modifier | modifier le code]

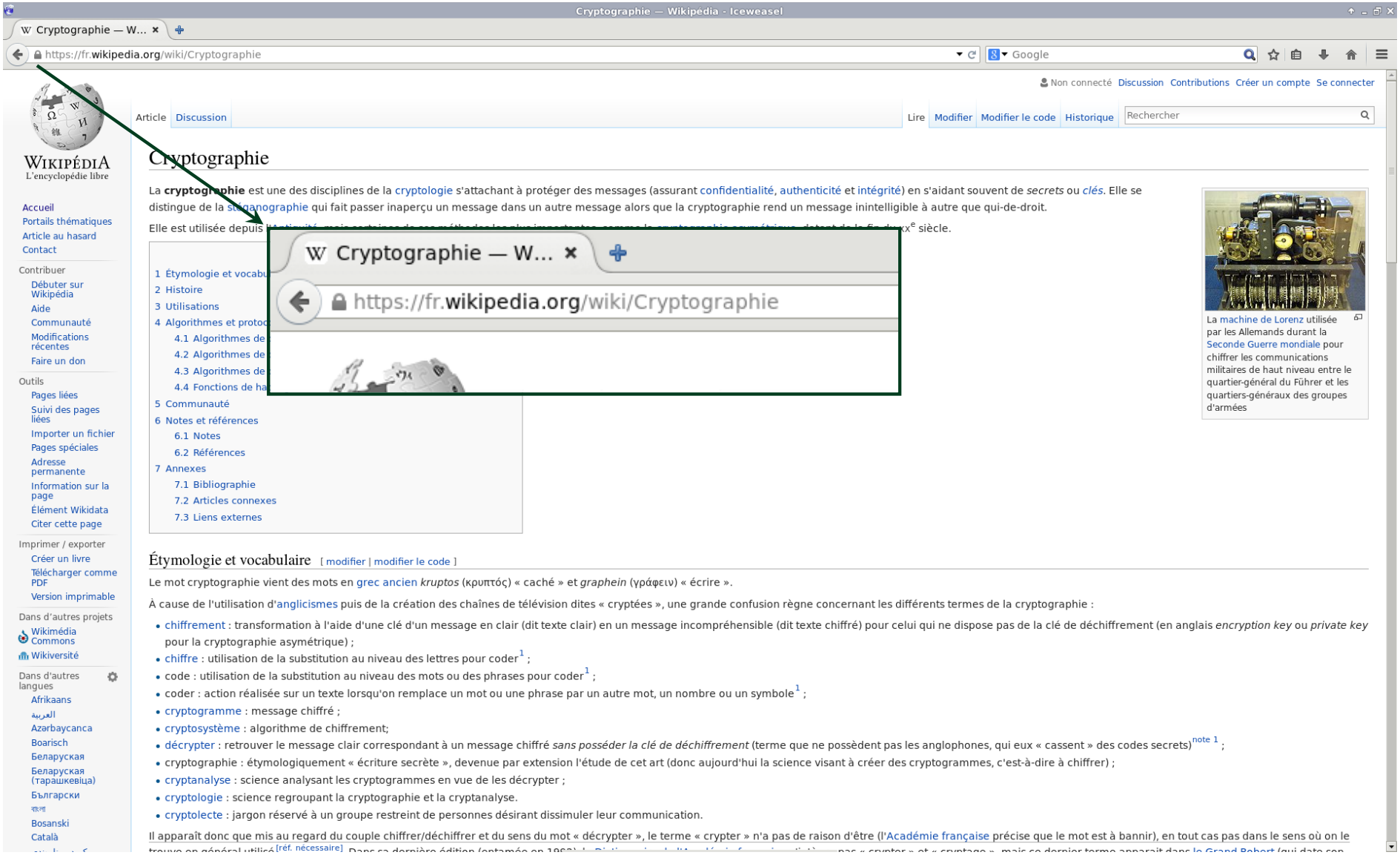
Le mot cryptographie vient des mots en grec ancien *kruptos* (κρυπτός) « caché » et *graphein* (γράφειν) « écrire ».

À cause de l'utilisation d'anglicismes puis de la création des chaînes de télévision dites « cryptées », une grande confusion règne concernant les différents termes de la cryptographie :

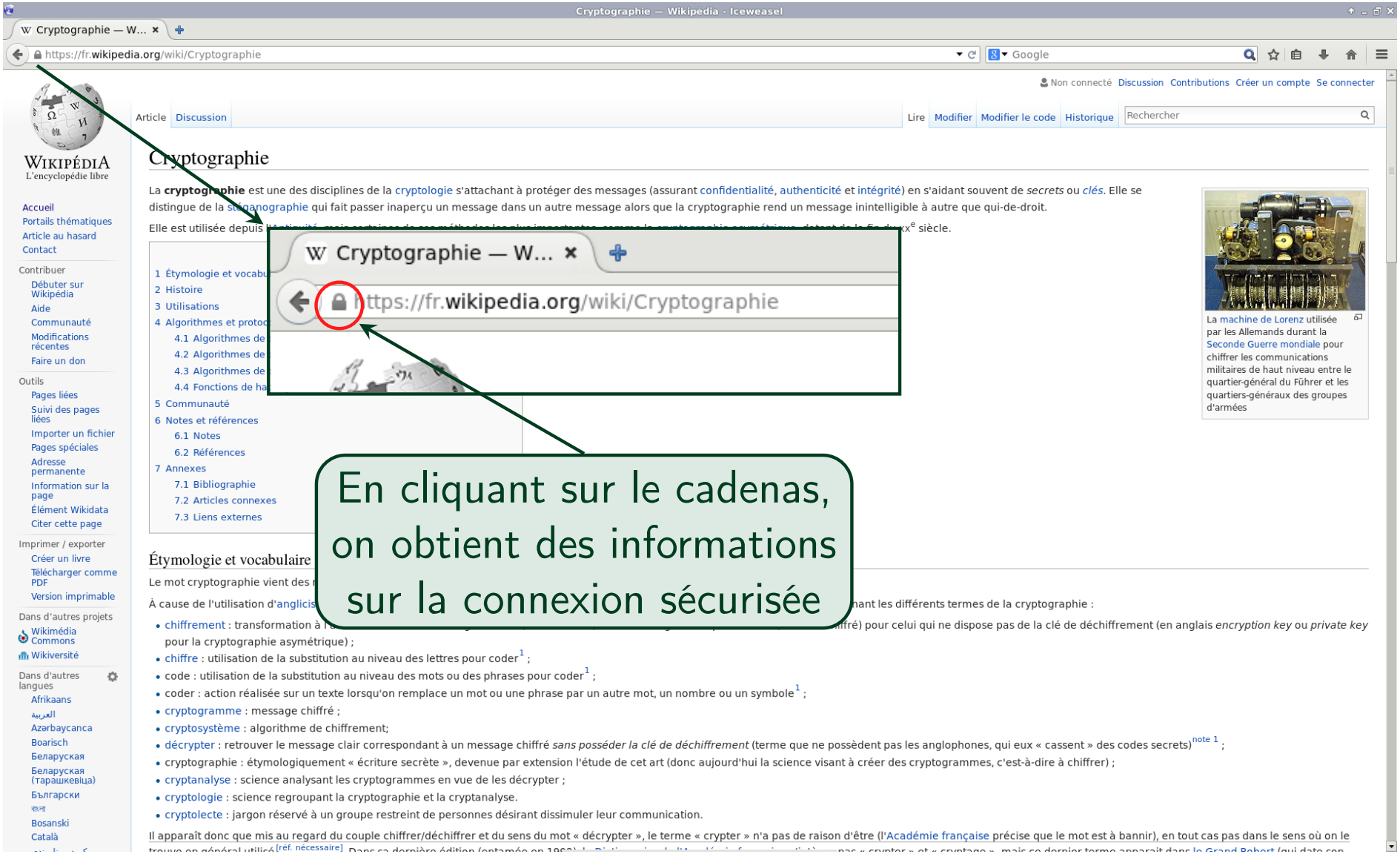
- chiffrement : transformation à l'aide d'une clé d'un message en clair (dit texte clair) en un message incompréhensible (dit texte chiffré) pour celui qui ne dispose pas de la clé de déchiffrement (en anglais *encryption key* ou *private key* pour la cryptographie asymétrique) ;
- chiffre : utilisation de la substitution au niveau des lettres pour coder[1] ;
- code : utilisation de la substitution au niveau des mots ou des phrases pour coder[1] ;
- coder : action réalisée sur un texte lorsqu'on remplace un mot ou une phrase par un autre mot, un nombre ou un symbole[1] ;
- cryptogramme : message chiffré ;
- cryptosystème : algorithme de chiffrement;
- décrypter : retrouver le message clair correspondant à un message chiffré *sans posséder la clé de déchiffrement* (terme que ne possèdent pas les anglophones, qui eux « cassent » des codes secrets)[note 1] ;
- cryptographie : étymologiquement « écriture secrète », devenue par extension l'étude de cet art (donc aujourd'hui la science visant à créer des cryptogrammes, c'est-à-dire à chiffrer) ;
- cryptanalyse : science analysant les cryptogrammes en vue de les décrypter ;
- cryptologie : science regroupant la cryptographie et la cryptanalyse.
- cryptolecte : jargon réservé à un groupe restreint de personnes désirant dissimuler leur communication.

Il apparaît donc que mis au regard du couple chiffrer/déchiffrer et du sens du mot « décrypter », le terme « crypter » n'a pas de raison d'être (l'Académie française précise que le mot est à bannir), en tout cas pas dans le sens où on le trouve en général utilisé.[réf. nécessaire] Dans sa dernière édition (entamée en 1863) le Dictionnaire de l'Académie française n'admet pas « crypter » et « cryptage », mais ce dernier terme apparaît dans le Grand Robert (qui date son

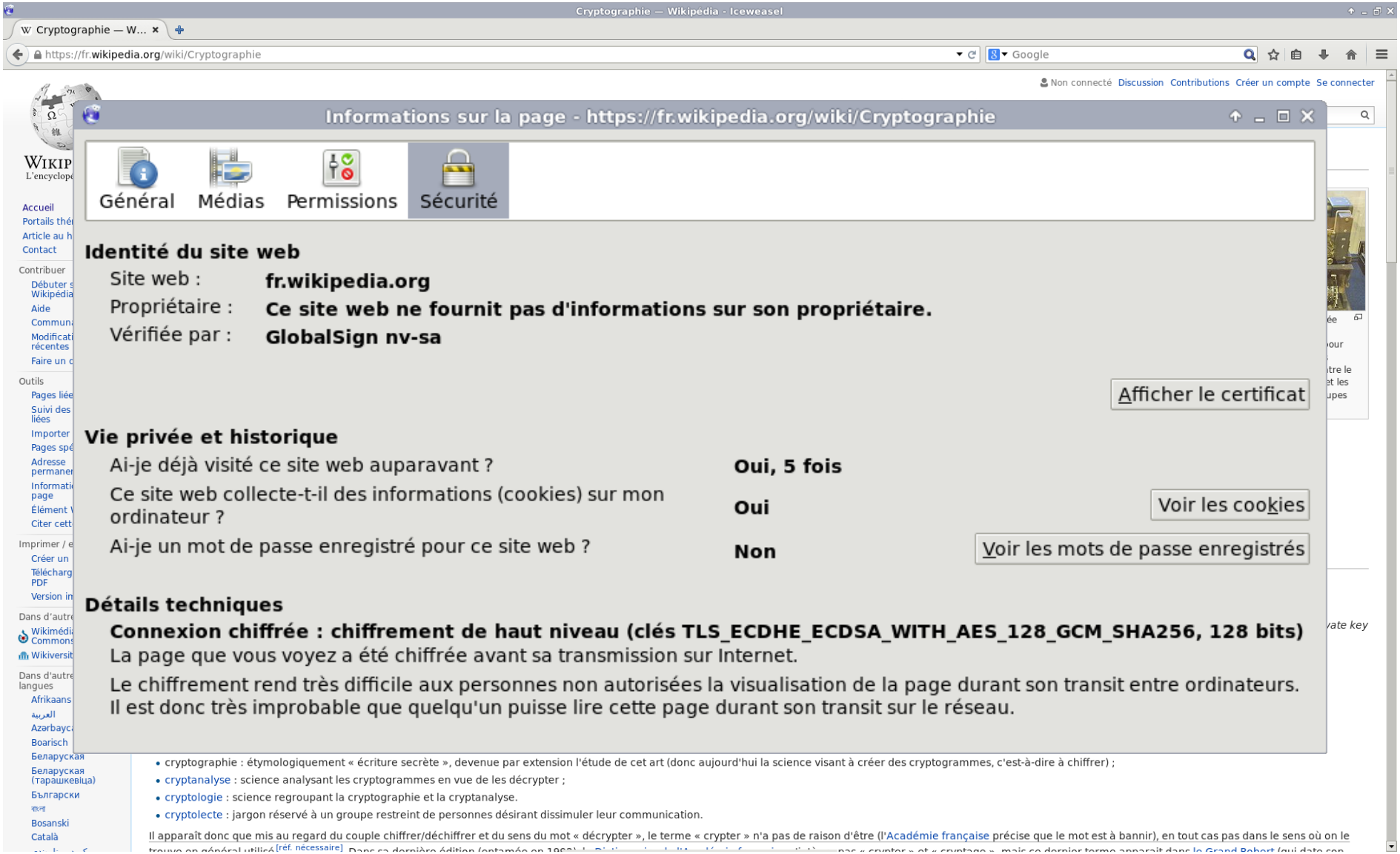
Connexion sur le site : https://fr.wikipedia.org



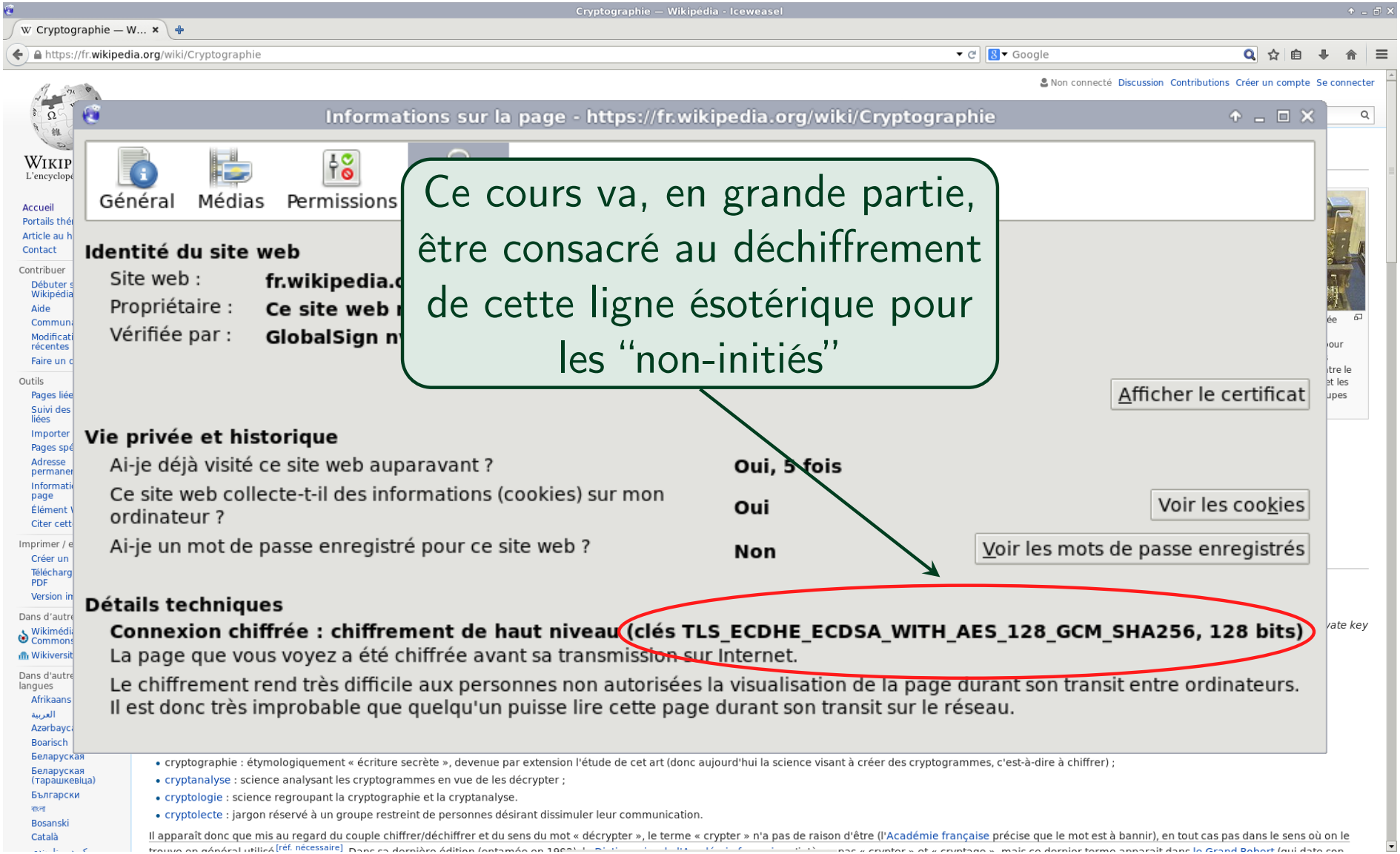
Connexion sur le site : https://fr.wikipedia.org



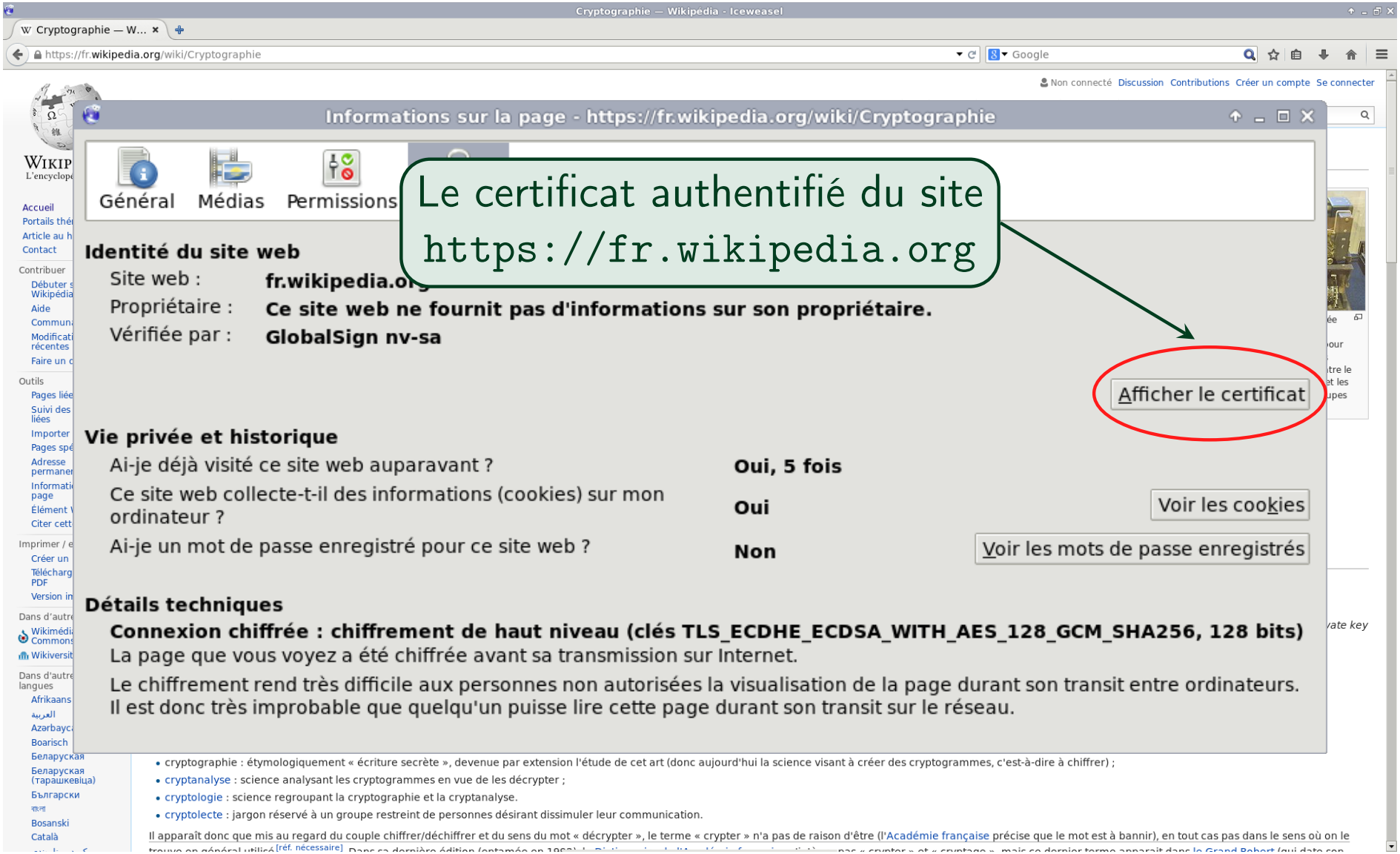
Connexion sur le site : https://fr.wikipedia.org



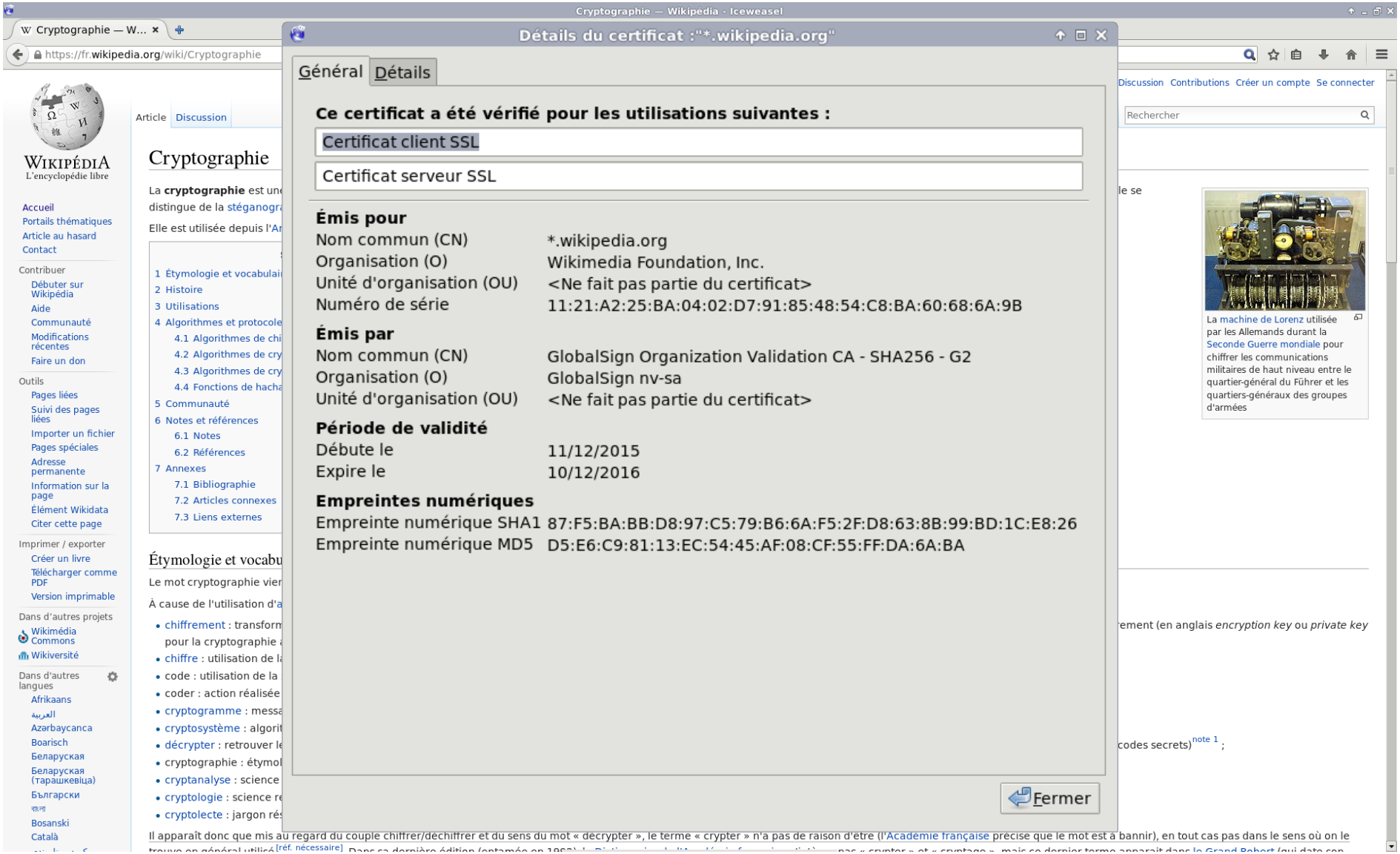
Connexion sur le site : https://fr.wikipedia.org



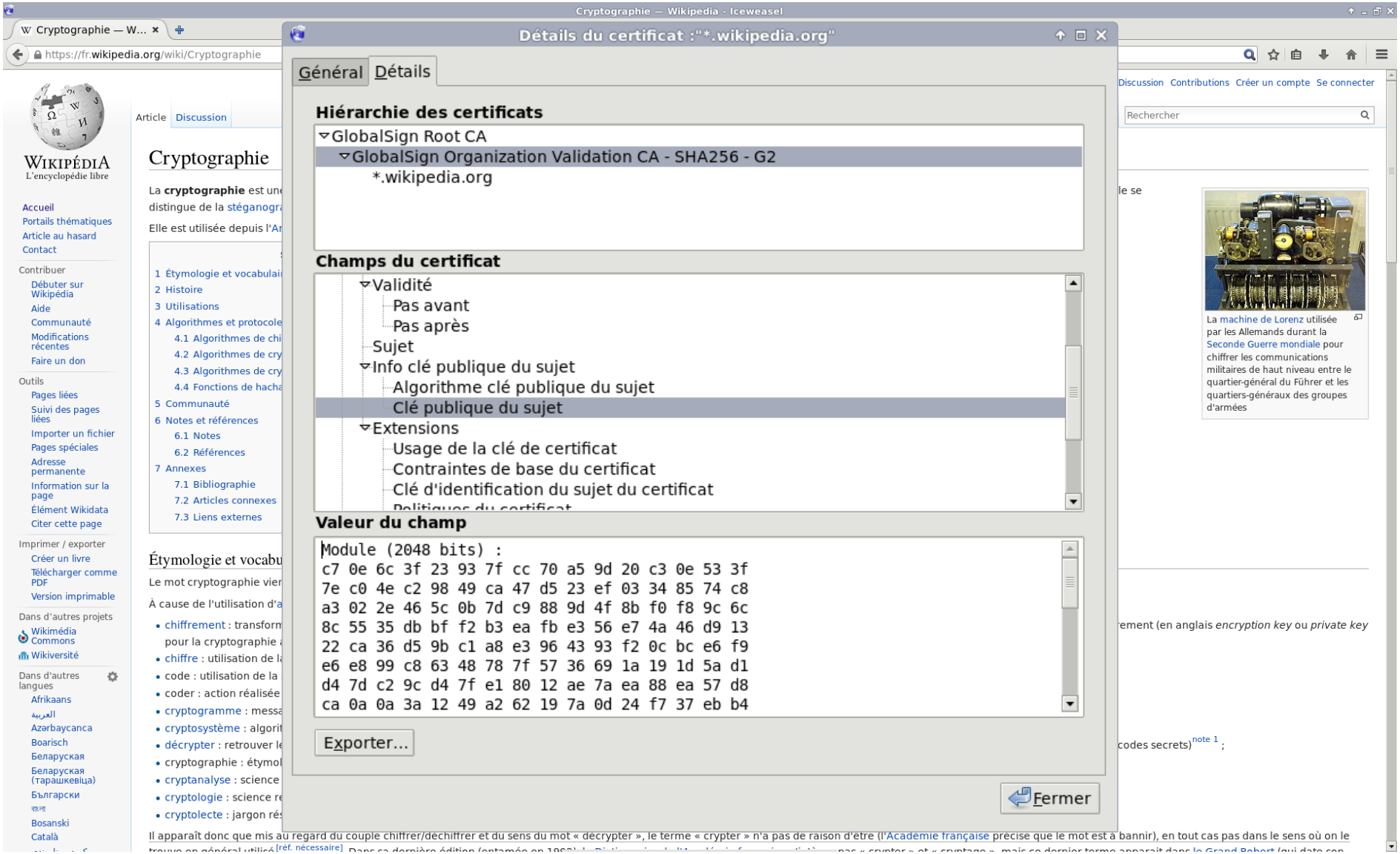
Connexion sur le site : https://fr.wikipedia.org



Connexion sur le site : https://fr.wikipedia.org



Connexion sur le site : https://fr.wikipedia.org



Première connexion sur un serveur ssh

Lors d'une première connexion sur un serveur ssh (*Secure Shell*), on a un dialogue du genre :

```
$ ssh 172.28.0.22
The authenticity of host '172.28.0.22 (172.28.0.22)'
can't be established. RSA key fingerprint is
99:7b:84:d8:01:fe:4e:e2:53:a4:9d:2f:3e:ae:05:3e.
Are you sure you want to continue connecting (yes/no)
```

Si on répond **yes**, comme on le fait généralement sans trop se poser de questions, cela signifie que l'on reconnaît comme *authentique* la clé publique du serveur ssh. On prend le risque d'être victime d'une attaque "man in the middle".
Une parade (non absolue) consiste à mettre l'empreinte de la clé publique du serveur ssh dans le DNS (normalisé dans le RFC 4255) :

Première connexion sur un serveur ssh

Lors d'une première connexion sur un serveur ssh (*Secure Shell*), on a un dialogue du genre :

```
$ ssh 172.28.0.22
The authenticity of host '172.28.0.22 (172.28.0.22)'
can't be established. RSA key fingerprint is
99:7b:84:d8:01:fe:4e:e2:53:a4:9d:2f:3e:ae:05:3e.
Are you sure you want to continue connecting (yes/no)
```

Si on répond yes, comme on le fait généralement sans trop se poser de questions, cela signifie que l'on reconnaît comme *authentique* la clé publique du serveur ssh. On prend le risque d'être victime d'une attaque "man in the middle".
Une parade (non absolue) consiste à mettre l'empreinte de la clé publique du serveur ssh dans le DNS (normalisé dans le RFC 4255) :

Première connexion sur un serveur ssh

Lors d'une première connexion sur un serveur ssh (*Secure Shell*), on a un dialogue du genre :

```
$ ssh 172.28.0.22
The authenticity of host '172.28.0.22 (172.28.0.22)'
can't be established. RSA key fingerprint is
99:7b:84:d8:01:fe:4e:e2:53:a4:9d:2f:3e:ae:05:3e.
Are you sure you want to continue connecting (yes/no)
```

Si on répond **yes**, comme on le fait généralement sans trop se poser de questions, cela signifie que l'on reconnaît comme *authentique* la clé publique du serveur ssh. On prend le risque d'être victime d'une attaque "man in the middle".

Une parade (non absolue) consiste à mettre l'empreinte de la clé publique du serveur ssh dans le DNS (normalisé dans le RFC 4255) :

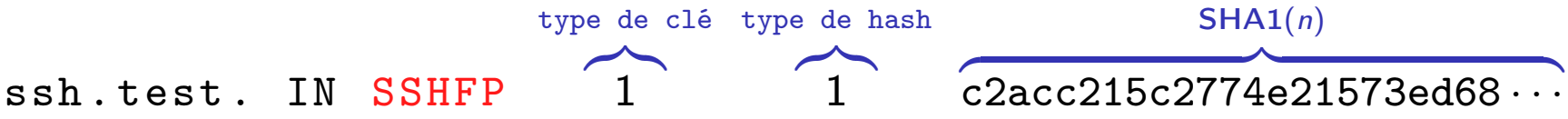
```
                                type de clé  type de hash   SHA1(n)
ssh.test. IN SSHFP              1            1             c2acc215c2774e21573ed68···
```

Première connexion sur un serveur ssh

Lors d'une première connexion sur un serveur ssh (*Secure Shell*), on a un dialogue du genre :

```
$ ssh 172.28.0.22
The authenticity of host '172.28.0.22 (172.28.0.22)'
can't be established. RSA key fingerprint is
99:7b:84:d8:01:fe:4e:e2:53:a4:9d:2f:3e:ae:05:3e.
Are you sure you want to continue connecting (yes/no)
```

Si on répond **yes**, comme on le fait généralement sans trop se poser de questions, cela signifie que l'on reconnaît comme *authentique* la clé publique du serveur ssh. On prend le risque d'être victime d'une attaque "man in the middle".
Une parade (non absolue) consiste à mettre l'empreinte de la clé publique du serveur ssh dans le DNS (normalisé dans le RFC 4255) :



Définition et propriétés

- L'opérateur logique XOR (*eXclusive OR*, ou exclusif), noté $\oplus$, est omniprésent en cryptographie :

$\oplus$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

La table de XOR

Définition et propriétés

- L'opérateur logique XOR (*eXclusive OR*, ou exclusif), noté $\oplus$, est omniprésent en cryptographie :

$\oplus$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

La table de XOR

- La propriété fondamentale :

$$\boxed{x \oplus x = 0}$$

Définition et propriétés

- L'opérateur logique XOR (*eXclusive OR*, ou exclusif), noté $\oplus$, est omniprésent en cryptographie :

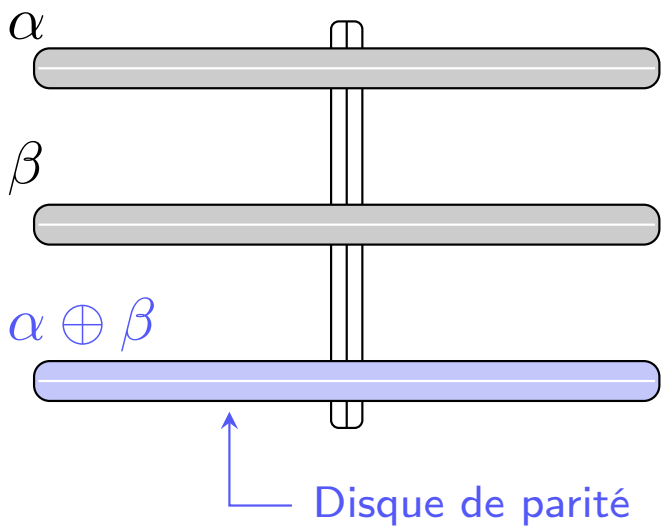
$\oplus$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

La table de XOR

- La propriété fondamentale :

$$\boxed{x \oplus x = 0}$$

- Un exemple : RAID4 (simplifié)



Définition et propriétés

- L'opérateur logique XOR (*eXclusive OR*, ou exclusif), noté $\oplus$, est omniprésent en cryptographie :

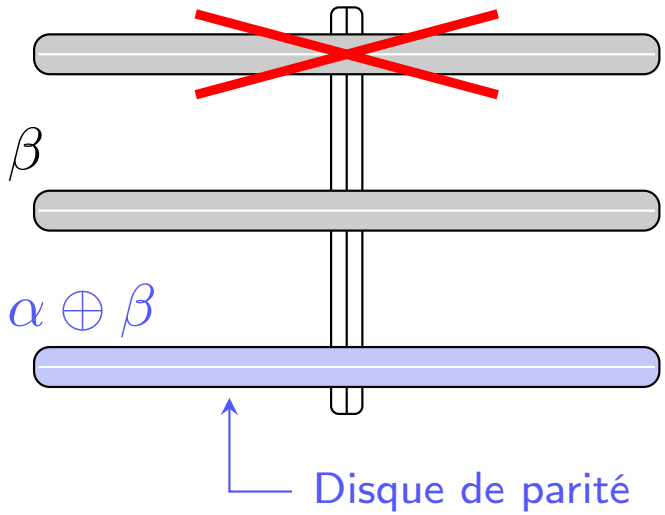
$\oplus$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

La table de XOR

- La propriété fondamentale :

$$\boxed{x \oplus x = 0}$$

- Un exemple : RAID4 (simplifié)



Définition et propriétés

- L'opérateur logique XOR (*eXclusive OR*, ou exclusif), noté $\oplus$, est omniprésent en cryptographie :

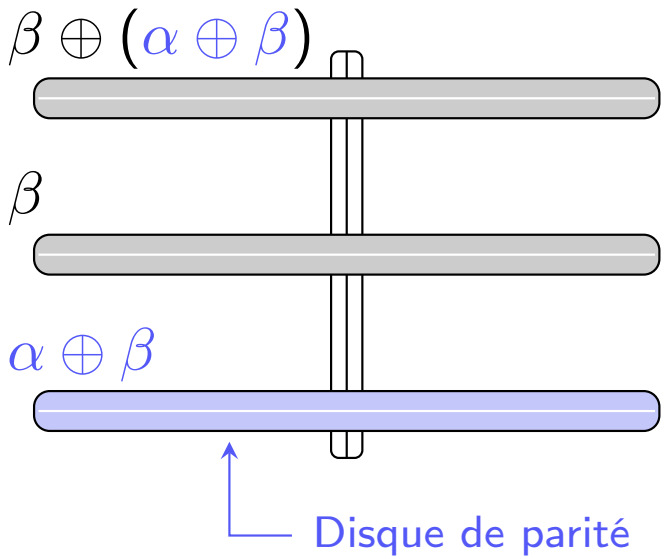
$\oplus$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

La table de XOR

- La propriété fondamentale :

$$\boxed{x \oplus x = 0}$$

- Un exemple : RAID4 (simplifié)



Définition et propriétés

- L'opérateur logique XOR (*eXclusive OR*, ou exclusif), noté $\oplus$, est omniprésent en cryptographie :

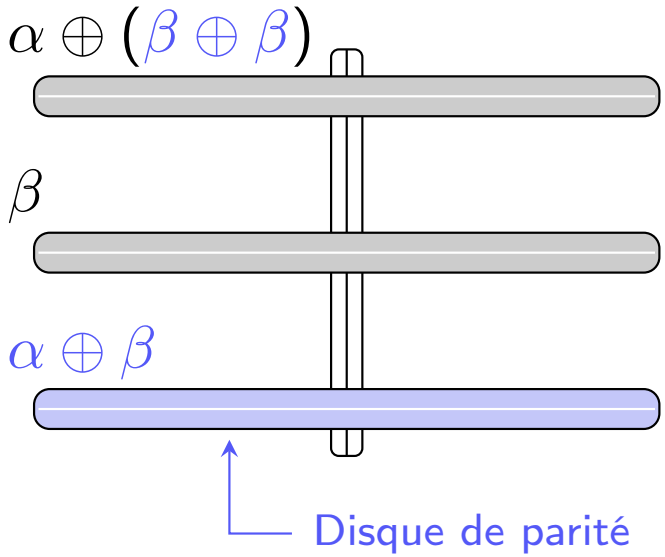
$\oplus$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

La table de XOR

- La propriété fondamentale :

$$\boxed{x \oplus x = 0}$$

- Un exemple : RAID4 (simplifié)



Définition et propriétés

- L'opérateur logique XOR (*eXclusive OR*, ou exclusif), noté $\oplus$, est omniprésent en cryptographie :

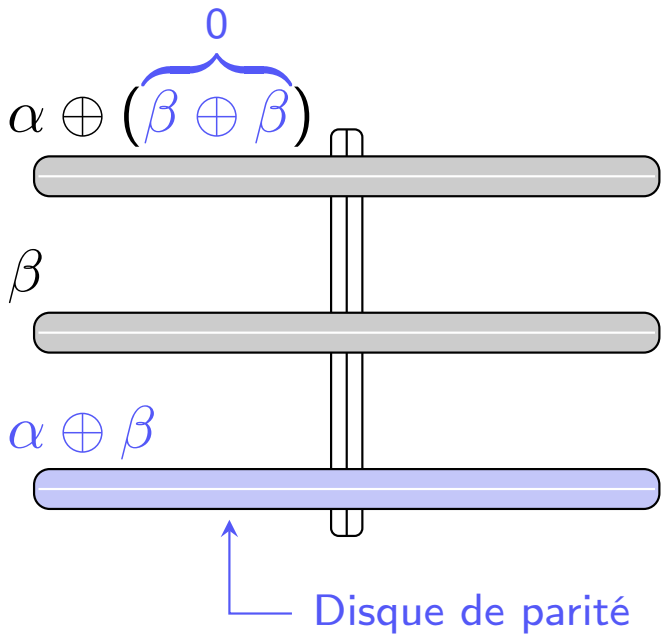
$\oplus$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

La table de XOR

- La propriété fondamentale :

$$\boxed{x \oplus x = 0}$$

- Un exemple : RAID4 (simplifié)



Définition et propriétés

- L'opérateur logique XOR (*eXclusive OR*, ou exclusif), noté $\oplus$, est omniprésent en cryptographie :

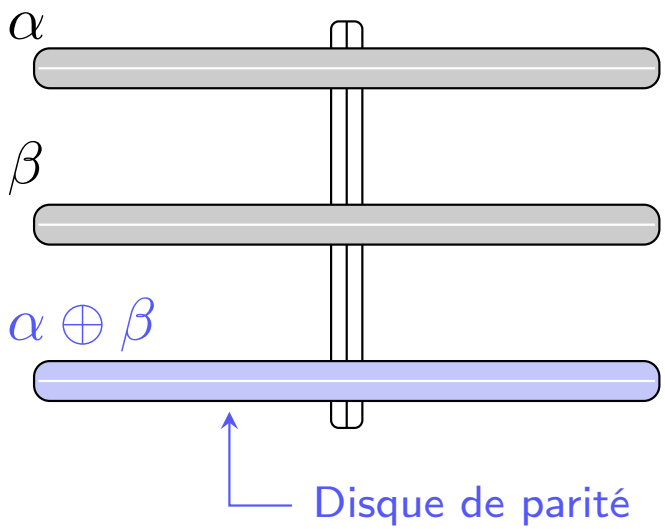
$\oplus$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

La table de XOR

- La propriété fondamentale :

$$\boxed{x \oplus x = 0}$$

- Un exemple : RAID4 (simplifié)



- La cryptographie repose de **manière cruciale** sur la **robustesse des mots de passe** utilisés. En effet, à quoi sert de déployer sur votre système les logiciels les plus performants et les plus récents, si vos utilisateurs (un seul suffit d'ailleurs) ont comme mot de passe le prénom de leur conjoint...
- Il faut **impérativement** respecter quelques règles de base :
  - le mot de passe doit avoir au moins 8 caractères (12 c'est mieux),
  - il doit comporter des majuscules **et** des minuscules **et** des chiffres **et** d'autres caractères (+, =, ?, :, etc.),
  - il faut éviter les répétitions de caractères, les suites bien connues ("azerty", "123456"), les mots des dictionnaires (français, anglais, russe, etc.), les codes postaux des 36681 communes de France, etc.,
  - il faut également éviter d'utiliser des informations vous concernant qui soient publiques (page WEB personnelle par exemple),
  - il faut aussi proscrire toute combinaison simple des items précédents,
  - le mot de passe doit être généré aléatoirement chaque fois que cela est possible. Par exemple, sous UNIX, un mot de passe de 64 bits :

```
$ dd if=/dev/random bs=1 count=8 2>/dev/null|xxd -ps
466caa3b32ac4184
```

- La cryptographie repose de **manière cruciale** sur la **robustesse des mots de passe** utilisés. En effet, à quoi sert de déployer sur votre système les logiciels les plus performants et les plus récents, si vos utilisateurs (un seul suffit d'ailleurs) ont comme mot de passe le prénom de leur conjoint...
- Il faut **impérativement** respecter quelques règles de base :
  - le mot de passe doit avoir au moins 8 caractères (12 c'est mieux),
  - il doit comporter des majuscules **et** des minuscules **et** des chiffres **et** d'autres caractères (+, =, ?, :, etc.),
  - il faut éviter les répétitions de caractères, les suites bien connues ("azerty", "123456"), les mots des dictionnaires (français, anglais, russe, etc.), les codes postaux des 36681 communes de France, etc.,
  - il faut également éviter d'utiliser des informations vous concernant qui soient publiques (page WEB personnelle par exemple),
  - il faut aussi proscrire toute combinaison simple des items précédents,
  - le mot de passe doit être généré aléatoirement chaque fois que cela est possible. Par exemple, sous UNIX, un mot de passe de 64 bits :

```
$ dd if=/dev/random bs=1 count=8 2>/dev/null|xxd -ps
466caa3b32ac4184
```

- La cryptographie repose de **manière cruciale** sur la **robustesse des mots de passe** utilisés. En effet, à quoi sert de déployer sur votre système les logiciels les plus performants et les plus récents, si vos utilisateurs (un seul suffit d'ailleurs) ont comme mot de passe le prénom de leur conjoint...
- Il faut **impérativement** respecter quelques règles de base :
  - le mot de passe doit avoir au moins 8 caractères (12 c'est mieux),
  - il doit comporter des majuscules **et** des minuscules **et** des chiffres **et** d'autres caractères (+, =, ?, :, etc.),
  - il faut éviter les répétitions de caractères, les suites bien connues ("azerty", "123456"), les mots des dictionnaires (français, anglais, russe, etc.), les codes postaux des 36681 communes de France, etc.,
  - il faut également éviter d'utiliser des informations vous concernant qui soient publiques (page WEB personnelle par exemple),
  - il faut aussi proscrire toute combinaison simple des items précédents,
  - le mot de passe doit être généré aléatoirement chaque fois que cela est possible. Par exemple, sous UNIX, un mot de passe de 64 bits :

```
$ dd if=/dev/random bs=1 count=8 2>/dev/null|xxd -ps
466caa3b32ac4184
```

- La cryptographie repose de **manière cruciale** sur la **robustesse des mots de passe** utilisés. En effet, à quoi sert de déployer sur votre système les logiciels les plus performants et les plus récents, si vos utilisateurs (un seul suffit d'ailleurs) ont comme mot de passe le prénom de leur conjoint...
- Il faut **impérativement** respecter quelques règles de base :
  - le mot de passe doit avoir au moins 8 caractères (12 c'est mieux),
  - il doit comporter des majuscules **et** des minuscules **et** des chiffres **et** d'autres caractères (+, =, ?, :, etc.),
  - il faut éviter les répétitions de caractères, les suites bien connues ("azerty", "123456"), les mots des dictionnaires (français, anglais, russe, etc.), les codes postaux des 36681 communes de France, etc.,
  - il faut également éviter d'utiliser des informations vous concernant qui soient publiques (page WEB personnelle par exemple),
  - il faut aussi proscrire toute combinaison simple des items précédents,
  - le mot de passe doit être généré aléatoirement chaque fois que cela est possible. Par exemple, sous UNIX, un mot de passe de 64 bits :

```
$ dd if=/dev/random bs=1 count=8 2>/dev/null|xxd -ps
466caa3b32ac4184
```

- La cryptographie repose de **manière cruciale** sur la **robustesse des mots de passe** utilisés. En effet, à quoi sert de déployer sur votre système les logiciels les plus performants et les plus récents, si vos utilisateurs (un seul suffit d'ailleurs) ont comme mot de passe le prénom de leur conjoint...
- Il faut **impérativement** respecter quelques règles de base :
  - le mot de passe doit avoir au moins 8 caractères (12 c'est mieux),
  - il doit comporter des majuscules **et** des minuscules **et** des chiffres **et** d'autres caractères (+, =, ?, :, etc.),
  - il faut éviter les répétitions de caractères, les suites bien connues ("azerty", "123456"), les mots des dictionnaires (français, anglais, russe, etc.), les codes postaux des 36681 communes de France, etc.,
  - il faut également éviter d'utiliser des informations vous concernant qui soient publiques (page WEB personnelle par exemple),
  - il faut aussi proscrire toute combinaison simple des items précédents,
  - le mot de passe doit être généré aléatoirement chaque fois que cela est possible. Par exemple, sous UNIX, un mot de passe de 64 bits :

```
$ dd if=/dev/random bs=1 count=8 2>/dev/null|xxd -ps
466caa3b32ac4184
```

- La cryptographie repose de **manière cruciale** sur la **robustesse des mots de passe** utilisés. En effet, à quoi sert de déployer sur votre système les logiciels les plus performants et les plus récents, si vos utilisateurs (un seul suffit d'ailleurs) ont comme mot de passe le prénom de leur conjoint...
- Il faut **impérativement** respecter quelques règles de base :
  - le mot de passe doit avoir au moins 8 caractères (12 c'est mieux),
  - il doit comporter des majuscules **et** des minuscules **et** des chiffres **et** d'autres caractères (+, =, ?, :, etc.),
  - il faut éviter les répétitions de caractères, les suites bien connues ("azerty", "123456"), les mots des dictionnaires (français, anglais, russe, etc.), les codes postaux des 36681 communes de France, etc.,
  - il faut également éviter d'utiliser des informations vous concernant qui soient publiques (page WEB personnelle par exemple),
  - il faut aussi proscrire toute combinaison simple des items précédents,
  - le mot de passe doit être généré aléatoirement chaque fois que cela est possible. Par exemple, sous UNIX, un mot de passe de 64 bits :

```
$ dd if=/dev/random bs=1 count=8 2>/dev/null|xxd -ps
466caa3b32ac4184
```

- La cryptographie repose de **manière cruciale** sur la **robustesse des mots de passe** utilisés. En effet, à quoi sert de déployer sur votre système les logiciels les plus performants et les plus récents, si vos utilisateurs (un seul suffit d'ailleurs) ont comme mot de passe le prénom de leur conjoint...
- Il faut **impérativement** respecter quelques règles de base :
  - le mot de passe doit avoir au moins 8 caractères (12 c'est mieux),
  - il doit comporter des majuscules **et** des minuscules **et** des chiffres **et** d'autres caractères (+, =, ?, :, etc.),
  - il faut éviter les répétitions de caractères, les suites bien connues ("azerty", "123456"), les mots des dictionnaires (français, anglais, russe, etc.), les codes postaux des 36681 communes de France, etc.,
  - il faut également éviter d'utiliser des informations vous concernant qui soient publiques (page WEB personnelle par exemple),
  - il faut aussi proscrire toute combinaison simple des items précédents,
  - le mot de passe doit être généré aléatoirement chaque fois que cela est possible. Par exemple, sous UNIX, un mot de passe de 64 bits :

```
$ dd if=/dev/random bs=1 count=8 2>/dev/null|xxd -ps
466caa3b32ac4184
```

- La cryptographie repose de **manière cruciale** sur la **robustesse des mots de passe** utilisés. En effet, à quoi sert de déployer sur votre système les logiciels les plus performants et les plus récents, si vos utilisateurs (un seul suffit d'ailleurs) ont comme mot de passe le prénom de leur conjoint...
- Il faut **impérativement** respecter quelques règles de base :
  - le mot de passe doit avoir au moins 8 caractères (12 c'est mieux),
  - il doit comporter des majuscules **et** des minuscules **et** des chiffres **et** d'autres caractères (+, =, ?, :, etc.),
  - il faut éviter les répétitions de caractères, les suites bien connues ("azerty", "123456"), les mots des dictionnaires (français, anglais, russe, etc.), les codes postaux des 36681 communes de France, etc.,
  - il faut également éviter d'utiliser des informations vous concernant qui soient publiques (page WEB personnelle par exemple),
  - il faut aussi proscrire toute combinaison simple des items précédents,
  - le mot de passe doit être généré aléatoirement chaque fois que cela est possible. Par exemple, sous UNIX, un mot de passe de 64 bits :

```
$ dd if=/dev/random bs=1 count=8 2>/dev/null|xxd -ps
466caa3b32ac4184
```

Robustesse des mots de passe (générés aléatoirement)

- On mesure de la robustesse d'un mot de passe avec l'**entropie**, une notion provenant de la *théorie de l'information* initialement développée par *Claude Shannon* en 1949. L'unité est le **bit** d'information.
- Dire qu'un mot de passe a une **robustesse** (ou **force**) de, par exemple, 64 bits signifie que pour le casser par force brute, il faudra au pire 2^{64} tentatives, soit 585 années au rythme de 10^9 essais par secondes !
- Plus précisément, si $\mathcal{A}$ est un alphabet fini (par exemple, les caractères alphanumériques) et $N = \#(\mathcal{A})$, le nombre de ses éléments, alors l'**entropie d'un mot de passe** de longueur L est :

$$\mathcal{E} = \log_2(N^L) = L \log_2(N)$$

- Par exemple, un mot de passe de force 64 et composé uniquement de chiffres, doit avoir au moins 20 chiffres. Si l'on s'autorise tous les caractères ASCII imprimables, pour la même robustesse, on devra avoir : $L \geq 10$, soit un mot de passe deux fois plus court.

Robustesse des mots de passe (générés aléatoirement)

- On mesure de la robustesse d'un mot de passe avec l'**entropie**, une notion provenant de la *théorie de l'information* initialement développée par *Claude Shannon* en 1949. L'unité est le **bit** d'information.
- Dire qu'un mot de passe a une **robustesse** (ou **force**) de, par exemple, 64 bits signifie que pour le casser par force brute, il faudra au pire 2^{64} tentatives, soit 585 années au rythme de 10^9 essais par secondes !
- Plus précisément, si $\mathcal{A}$ est un alphabet fini (par exemple, les caractères alphanumériques) et $N = \#(\mathcal{A})$, le nombre de ses éléments, alors l'**entropie d'un mot de passe** de longueur L est :

$$\mathcal{E} = \log_2(N^L) = L \log_2(N)$$

- Par exemple, un mot de passe de force 64 et composé uniquement de chiffres, doit avoir au moins 20 chiffres. Si l'on s'autorise tous les caractères ASCII imprimables, pour la même robustesse, on devra avoir : $L \geq 10$, soit un mot de passe deux fois plus court.

Robustesse des mots de passe (générés aléatoirement)

- On mesure de la robustesse d'un mot de passe avec l'**entropie**, une notion provenant de la *théorie de l'information* initialement développée par *Claude Shannon* en 1949. L'unité est le **bit** d'information.
- Dire qu'un mot de passe a une **robustesse** (ou **force**) de, par exemple, 64 bits signifie que pour le casser par force brute, il faudra au pire 2^{64} tentatives, soit 585 années au rythme de 10^9 essais par secondes !
- Plus précisément, si $\mathcal{A}$ est un alphabet fini (par exemple, les caractères alphanumériques) et $N = \#(\mathcal{A})$, le nombre de ses éléments, alors l'**entropie d'un mot de passe** de longueur L est :

$$\mathcal{E} = \log_2(N^L) = L \log_2(N)$$

- Par exemple, un mot de passe de force 64 et composé uniquement de chiffres, doit avoir au moins 20 chiffres. Si l'on s'autorise tous les caractères ASCII imprimables, pour la même robustesse, on devra avoir : $L \geq 10$, soit un mot de passe deux fois plus court.

Robustesse des mots de passe (générés aléatoirement)

- On mesure de la robustesse d'un mot de passe avec l'**entropie**, une notion provenant de la *théorie de l'information* initialement développée par *Claude Shannon* en 1949. L'unité est le **bit** d'information.
- Dire qu'un mot de passe a une **robustesse** (ou **force**) de, par exemple, 64 bits signifie que pour le casser par force brute, il faudra au pire 2^{64} tentatives, soit 585 années au rythme de 10^9 essais par secondes !
- Plus précisément, si $\mathcal{A}$ est un alphabet fini (par exemple, les caractères alphanumériques) et $N = \#(\mathcal{A})$, le nombre de ses éléments, alors l'**entropie d'un mot de passe** de longueur L est :

$$\mathcal{E} = \log_2(N^L) = L \log_2(N)$$

- Par exemple, un mot de passe de force 64 et composé uniquement de chiffres, doit avoir au moins 20 chiffres. Si l'on s'autorise tous les caractères ASCII imprimables, pour la même robustesse, on devra avoir : $L \geq 10$, soit un mot de passe deux fois plus court.

L'idée de la fonction à sens unique (*one-way function*)

- Informellement, une fonction à sens unique est une fonction facile à calculer mais en revanche *difficilement inversible*.
- Plus précisément, si $f : X \to Y$ est une telle fonction, alors, pour $x \in X$, $f(x)$ est facile à calculer et, pour $y \in f(X)$, il est difficile de trouver $x \in X$ tel que $y = f(x)$.
- En pratique, facile à calculer signifie qu'il existe un algorithme qui calcule $f(x)$ en temps polynomial en le nombre de bits nécessaires à l'écriture de x ($\log_2(\#(X))$).
- Pour chaque $y \in f(X)$, dire qu'il est difficile de trouver $x \in X$ tel que $y = f(x)$ signifie qu'avec les moyens et connaissances actuels, il est impossible de trouver un tel x en un temps raisonnable sauf à compter sur une chance, elle, tout à fait déraisonnable.
- Ces fonctions à sens unique ont été introduites au milieu des années 1970 et sont un des piliers de la cryptographie moderne.

L'idée de la fonction à sens unique (*one-way function*)

- Informellement, une fonction à sens unique est une fonction facile à calculer mais en revanche *difficilement inversible*.
- Plus précisément, si $f : X \to Y$ est une telle fonction, alors, pour $x \in X$, $f(x)$ est facile à calculer et, pour $y \in f(X)$, il est difficile de trouver $x \in X$ tel que $y = f(x)$.
- En pratique, facile à calculer signifie qu'il existe un algorithme qui calcule $f(x)$ en temps polynomial en le nombre de bits nécessaires à l'écriture de x ($\log_2(\#(X))$).
- Pour chaque $y \in f(X)$, dire qu'il est difficile de trouver $x \in X$ tel que $y = f(x)$ signifie qu'avec les moyens et connaissances actuels, il est impossible de trouver un tel x en un temps raisonnable sauf à compter sur une chance, elle, tout à fait déraisonnable.
- Ces fonctions à sens unique ont été introduites au milieu des années 1970 et sont un des piliers de la cryptographie moderne.

L'idée de la fonction à sens unique (*one-way function*)

- Informellement, une fonction à sens unique est une fonction facile à calculer mais en revanche *difficilement inversible*.
- Plus précisément, si $f : X \to Y$ est une telle fonction, alors, pour $x \in X$, $f(x)$ est facile à calculer et, pour $y \in f(X)$, il est difficile de trouver $x \in X$ tel que $y = f(x)$.
- En pratique, facile à calculer signifie qu'il existe un algorithme qui calcule $f(x)$ en temps polynomial en le nombre de bits nécessaires à l'écriture de x ($\log_2(\#(X))$).
- Pour chaque $y \in f(X)$, dire qu'il est difficile de trouver $x \in X$ tel que $y = f(x)$ signifie qu'avec les moyens et connaissances actuels, il est impossible de trouver un tel x en un temps raisonnable sauf à compter sur une chance, elle, tout à fait déraisonnable.
- Ces fonctions à sens unique ont été introduites au milieu des années 1970 et sont un des piliers de la cryptographie moderne.

L'idée de la fonction à sens unique (*one-way function*)

- Informellement, une fonction à sens unique est une fonction facile à calculer mais en revanche *difficilement inversible*.
- Plus précisément, si $f : X \to Y$ est une telle fonction, alors, pour $x \in X$, $f(x)$ est facile à calculer et, pour $y \in f(X)$, il est difficile de trouver $x \in X$ tel que $y = f(x)$.
- En pratique, facile à calculer signifie qu'il existe un algorithme qui calcule $f(x)$ en temps polynomial en le nombre de bits nécessaires à l'écriture de x ($\log_2(\#(X))$).
- Pour chaque $y \in f(X)$, dire qu'il est difficile de trouver $x \in X$ tel que $y = f(x)$ signifie qu'avec les moyens et connaissances actuels, il est impossible de trouver un tel x en un temps raisonnable sauf à compter sur une chance, elle, tout à fait déraisonnable.
- Ces fonctions à sens unique ont été introduites au milieu des années 1970 et sont un des piliers de la cryptographie moderne.

L'idée de la fonction à sens unique (*one-way function*)

- Informellement, une fonction à sens unique est une fonction facile à calculer mais en revanche *difficilement inversible*.
- Plus précisément, si $f : X \to Y$ est une telle fonction, alors, pour $x \in X$, $f(x)$ est facile à calculer et, pour $y \in f(X)$, il est difficile de trouver $x \in X$ tel que $y = f(x)$.
- En pratique, facile à calculer signifie qu'il existe un algorithme qui calcule $f(x)$ en temps polynomial en le nombre de bits nécessaires à l'écriture de x ($\log_2(\#(X))$).
- Pour chaque $y \in f(X)$, dire qu'il est difficile de trouver $x \in X$ tel que $y = f(x)$ signifie qu'avec les moyens et connaissances actuels, il est impossible de trouver un tel x en un temps raisonnable sauf à compter sur une chance, elle, tout à fait déraisonnable.
- Ces fonctions à sens unique ont été introduites au milieu des années 1970 et sont un des piliers de la cryptographie moderne.

Quelques applications

- L'une des premières applications a été le *stockage des mots de passe*. En effet, il n'est guère avisé de stocker un mot de passe m en clair sur un système. On stocke plutôt $f(m)$ où f est une fonction à sens unique. Ainsi la connaissance de $f(m)$ ne permet pas de retrouver le mot de passe m. Pour vérifier qu'un mot de passe m est correct, on calcule tout d'abord $f(m)$ et on le compare avec la valeur stockée.
- Le protocole de *Diffie-Hellman* qui permet l'établissement d'un secret partagé entre deux entités à travers un canal non sécurisé.
  Ce protocole repose sur des fonctions à sens unique, notamment l'exponentiation modulaire.
- Lors de l'échange de données entre deux entités à travers un canal non sécurisé, les fonctions à sens unique permettent d'authentifier les parties en présence et de garantir l'intégrité des données transmises.
- Et bien d'autres encore dont certaines plutôt inattendues...

Quelques applications

- L'une des premières applications a été le *stockage des mots de passe*. En effet, il n'est guère avisé de stocker un mot de passe m en clair sur un système. On stocke plutôt $f(m)$ où f est une fonction à sens unique. Ainsi la connaissance de $f(m)$ ne permet pas de retrouver le mot de passe m. Pour vérifier qu'un mot de passe m est correct, on calcule tout d'abord $f(m)$ et on le compare avec la valeur stockée.
- Le protocole de *Diffie-Hellman* qui permet l'établissement d'un secret partagé entre deux entités à travers un canal non sécurisé.
  Ce protocole repose sur des fonctions à sens unique, notamment l'exponentiation modulaire.
- Lors de l'échange de données entre deux entités à travers un canal non sécurisé, les fonctions à sens unique permettent d'authentifier les parties en présence et de garantir l'intégrité des données transmises.
- Et bien d'autres encore dont certaines plutôt inattendues...

Quelques applications

- L'une des premières applications a été le *stockage des mots de passe*. En effet, il n'est guère avisé de stocker un mot de passe m en clair sur un système. On stocke plutôt $f(m)$ où f est une fonction à sens unique. Ainsi la connaissance de $f(m)$ ne permet pas de retrouver le mot de passe m. Pour vérifier qu'un mot de passe m est correct, on calcule tout d'abord $f(m)$ et on le compare avec la valeur stockée.
- Le protocole de *Diffie-Hellman* qui permet l'établissement d'un secret partagé entre deux entités à travers un canal non sécurisé.
  Ce protocole repose sur des fonctions à sens unique, notamment l'exponentiation modulaire.
- Lors de l'échange de données entre deux entités à travers un canal non sécurisé, les fonctions à sens unique permettent d'authentifier les parties en présence et de garantir l'intégrité des données transmises.
- Et bien d'autres encore dont certaines plutôt inattendues...

Quelques applications

- L'une des premières applications a été le *stockage des mots de passe*. En effet, il n'est guère avisé de stocker un mot de passe m en clair sur un système. On stocke plutôt $f(m)$ où f est une fonction à sens unique. Ainsi la connaissance de $f(m)$ ne permet pas de retrouver le mot de passe m. Pour vérifier qu'un mot de passe m est correct, on calcule tout d'abord $f(m)$ et on le compare avec la valeur stockée.
- Le protocole de *Diffie-Hellman* qui permet l'établissement d'un secret partagé entre deux entités à travers un canal non sécurisé.
  Ce protocole repose sur des fonctions à sens unique, notamment l'exponentiation modulaire.
- Lors de l'échange de données entre deux entités à travers un canal non sécurisé, les fonctions à sens unique permettent d'authentifier les parties en présence et de garantir l'intégrité des données transmises.
- Et bien d'autres encore dont certaines plutôt inattendues...

Quelques définitions

- Une fonction de hachage (ou de condensation) est une fonction *à sens unique* qui prend en entrée des données de **longueur quelconque** et rend une **valeur de taille fixe**.
- Plus précisément, soit A un alphabet fini, par exemple l'ensemble des signes permettant d'écrire le français ou l'ensemble $\{0,1\}$. Pour tout entier $k \geq 1$, A^k est l'ensemble des mots de k lettres et l'ensemble $A^* = \cup_{k=1}^{\infty} A^k$ est l'ensemble de tous les mots possibles. Une fonction de hachage peut être vue comme une fonction à sens unique $f : A^* \to \{0,1\}^n$ où $\{0,1\}^n$ s'interprête comme l'ensemble des mots de n bits avec $n \geq 1$ (par exemple, pour MD5 et SHA-1, n vaut respectivement 128 et 160).
- $f(x)$ est appelé l'**empreinte** (ou aussi : **condensé**, **condensat**, **haché**, **hash**) du texte x.

Quelques définitions

- Une fonction de hachage (ou de condensation) est une fonction *à sens unique* qui prend en entrée des données de **longueur quelconque** et rend une **valeur de taille fixe**.
- Plus précisément, soit A un alphabet fini, par exemple l'ensemble des signes permettant d'écrire le français ou l'ensemble $\{0,1\}$. Pour tout entier $k \geq 1$, A^k est l'ensemble des mots de k lettres et l'ensemble $A^* = \cup_{k=1}^{\infty} A^k$ est l'ensemble de tous les mots possibles. Une fonction de hachage peut être vue comme une fonction à sens unique $f : A^* \to \{0,1\}^n$ où $\{0,1\}^n$ s'interprête comme l'ensemble des mots de n bits avec $n \geq 1$ (par exemple, pour MD5 et SHA-1, n vaut respectivement 128 et 160).
- $f(x)$ est appelé l'**empreinte** (ou aussi : **condensé**, **condensat**, **haché**, **hash**) du texte x.

Quelques définitions

- Une fonction de hachage (ou de condensation) est une fonction *à sens unique* qui prend en entrée des données de **longueur quelconque** et rend une **valeur de taille fixe**.
- Plus précisément, soit A un alphabet fini, par exemple l'ensemble des signes permettant d'écrire le français ou l'ensemble $\{0,1\}$. Pour tout entier $k \geq 1$, A^k est l'ensemble des mots de k lettres et l'ensemble $A^* = \cup_{k=1}^{\infty} A^k$ est l'ensemble de tous les mots possibles. Une fonction de hachage peut être vue comme une fonction à sens unique $f : A^* \to \{0,1\}^n$ où $\{0,1\}^n$ s'interprête comme l'ensemble des mots de n bits avec $n \geq 1$ (par exemple, pour MD5 et SHA-1, n vaut respectivement 128 et 160).
- $f(x)$ est appelé l'**empreinte** (ou aussi : **condensé**, **condensat**, **haché**, **hash**) du texte x.

Définition

- En plus d'être à sens unique, on voudrait que f soit telle que, si $x \neq x'$, alors $f(x) \neq f(x')$, autrement dit que f soit **injective**.
- La contrainte d'injectivité de la fonction de hachage f ne peut être satisfaite. En effet, il ne peut y avoir d'injection de A^* dans $\{0,1\}^n$ car A^* est un ensemble infini (dénombrable) et $\{0,1\}^n$ est un ensemble fini (2^n éléments).
- Par conséquent, il existe nécessairement $x, x' \in A^*$ tels que $x \neq x'$ et $f(x) = f(x')$. On appelle cela une **collision**.
- Donc tout ce que l'on peut demander à f, c'est qu'il soit difficile (au sens précédent) de fabriquer des collisions.
- Il va falloir, pour satisfaire la contrainte précédente que n soit suffisamment grand afin d'éviter *l'attaque par anniversaires*.

Définition

- En plus d'être à sens unique, on voudrait que f soit telle que, si $x \neq x'$, alors $f(x) \neq f(x')$, autrement dit que f soit **injective**.
- La contrainte d'injectivité de la fonction de hachage f ne peut être satisfaite. En effet, il ne peut y avoir d'injection de A^* dans $\{0,1\}^n$ car A^* est un ensemble infini (dénombrable) et $\{0,1\}^n$ est un ensemble fini (2^n éléments).
- Par conséquent, il existe nécessairement $x, x' \in A^*$ tels que $x \neq x'$ et $f(x) = f(x')$. On appelle cela une **collision**.
- Donc tout ce que l'on peut demander à f, c'est qu'il soit difficile (au sens précédent) de fabriquer des collisions.
- Il va falloir, pour satisfaire la contrainte précédente que n soit suffisamment grand afin d'éviter *l'attaque par anniversaires*.

Définition

- En plus d'être à sens unique, on voudrait que f soit telle que, si $x \neq x'$, alors $f(x) \neq f(x')$, autrement dit que f soit **injective**.
- La contrainte d'injectivité de la fonction de hachage f ne peut être satisfaite. En effet, il ne peut y avoir d'injection de A^* dans $\{0,1\}^n$ car A^* est un ensemble infini (dénombrable) et $\{0,1\}^n$ est un ensemble fini (2^n éléments).
- Par conséquent, il existe nécessairement $x, x' \in A^*$ tels que $x \neq x'$ et $f(x) = f(x')$. On appelle cela une **collision**.
- Donc tout ce que l'on peut demander à f, c'est qu'il soit difficile (au sens précédent) de fabriquer des collisions.
- Il va falloir, pour satisfaire la contrainte précédente que n soit suffisamment grand afin d'éviter *l'attaque par anniversaires*.

Définition

- En plus d'être à sens unique, on voudrait que f soit telle que, si $x \neq x'$, alors $f(x) \neq f(x')$, autrement dit que f soit **injective**.
- La contrainte d'injectivité de la fonction de hachage f ne peut être satisfaite. En effet, il ne peut y avoir d'injection de A^* dans $\{0,1\}^n$ car A^* est un ensemble infini (dénombrable) et $\{0,1\}^n$ est un ensemble fini (2^n éléments).
- Par conséquent, il existe nécessairement $x, x' \in A^*$ tels que $x \neq x'$ et $f(x) = f(x')$. On appelle cela une **collision**.
- Donc tout ce que l'on peut demander à f, c'est qu'il soit difficile (au sens précédent) de fabriquer des collisions.
- Il va falloir, pour satisfaire la contrainte précédente que n soit suffisamment grand afin d'éviter *l'attaque par anniversaires*.

Définition

- En plus d'être à sens unique, on voudrait que f soit telle que, si $x \neq x'$, alors $f(x) \neq f(x')$, autrement dit que f soit **injective**.
- La contrainte d'injectivité de la fonction de hachage f ne peut être satisfaite. En effet, il ne peut y avoir d'injection de A^* dans $\{0,1\}^n$ car A^* est un ensemble infini (dénombrable) et $\{0,1\}^n$ est un ensemble fini (2^n éléments).
- Par conséquent, il existe nécessairement $x, x' \in A^*$ tels que $x \neq x'$ et $f(x) = f(x')$. On appelle cela une **collision**.
- Donc tout ce que l'on peut demander à f, c'est qu'il soit difficile (au sens précédent) de fabriquer des collisions.
- Il va falloir, pour satisfaire la contrainte précédente que n soit suffisamment grand afin d'éviter *l'attaque par anniversaires*.

Paradoxe des anniversaires

- On se pose la question de savoir combien doit-on réunir de personnes afin d'avoir plus de 50% de chances d'avoir au moins deux personnes nées le même jour de l'année (pour simplifier, on suppose toutes les années non bissextiles, ce qui ne change guère le résultat).

Paradoxe des anniversaires

- On se pose la question de savoir combien doit-on réunir de personnes afin d'avoir plus de 50% de chances d'avoir au moins deux personnes nées le même jour de l'année (pour simplifier, on suppose toutes les années non bissextiles, ce qui ne change guère le résultat).
- La réponse, très contre-intuitive, est 23. Si on réunit 50 personnes, la probabilité est de 97,04% et, pour 80 personnes, elle est de 99,99%, une quasi certitude.

Paradoxe des anniversaires

- On se pose la question de savoir combien doit-on réunir de personnes afin d'avoir plus de 50% de chances d'avoir au moins deux personnes nées le même jour de l'année (pour simplifier, on suppose toutes les années non bissextiles, ce qui ne change guère le résultat).
- La réponse, très contre-intuitive, est 23. Si on réunit 50 personnes, la probabilité est de 97,04% et, pour 80 personnes, elle est de 99,99%, une quasi certitude.
- Le calcul est simple. Si n est le nombre personnes et $P(n)$ la probabilité cherchée, on a :

$$P(n) = 1 - A_{365}^{n} \cdot \frac{1}{365^n} = 1 - \frac{365!}{(365-n)!} \cdot \frac{1}{365^n}$$

Paradoxe des anniversaires

- On se pose la question de savoir combien doit-on réunir de personnes afin d'avoir plus de 50% de chances d'avoir au moins deux personnes nées le même jour de l'année (pour simplifier, on suppose toutes les années non bissextiles, ce qui ne change guère le résultat).
- La réponse, très contre-intuitive, est 23. Si on réunit 50 personnes, la probabilité est de 97,04% et, pour 80 personnes, elle est de 99,99%, une quasi certitude.
- Le calcul est simple. Si n est le nombre personnes et $P(n)$ la probabilité cherchée, on a :

$$P(n) = 1 - A_{365}^{n} \cdot \frac{1}{365^n} = 1 - \frac{365!}{(365-n)!} \cdot \frac{1}{365^n}$$

- En revanche, si on calcule la probabilité P_{23} que, sur un groupe de 23 personnes, une personne ait son anniversaire le 29 septembre, on constate que celle-ci est beaucoup plus faible. En effet, $P_{23} = 1 - (\frac{364}{365})^{23} \approx 6,11\%$.

Attaque des anniversaires

- Soit f une fonction de hachage. On choisit *aléatoirement* $x_1, x_2, \ldots \in A^*$ jusqu'à ce qu'on trouve une collision, c'est à dire x_i, x_j avec $i \neq j$ et tels que $f(x_i) = f(x_j)$.
- Grâce au paradoxe des anniversaires, cette méthode, pourtant rudimentaire, peut se révéler être très efficace surtout si la taille de l'empreinte est trop petite.
- Si, par exemple, la taille de l'empreinte est de 64 bits, il faudra environ $7,2 \times 10^9$ essais pour que l'on ait au moins 75% de chances de découvrir une collision (on suppose que toutes les empreintes sont équiprobables). C'est tout à fait à la portée d'un PC. Il faut également comparer ce nombre au nombre d'empreintes possibles, à savoir environ $1,8 \times 10^{19}$.
- Avec une empreinte de 512 bits et pour une même probabilité, il ne faudra pas moins de $1,9 \times 10^{77}$ essais, ce qui est hors de portée d'autant plus que cette attaque nécessite beaucoup de mémoire.

Attaque des anniversaires

- Soit f une fonction de hachage. On choisit *aléatoirement* $x_1, x_2, \ldots \in A^*$ jusqu'à ce qu'on trouve une collision, c'est à dire x_i, x_j avec $i \neq j$ et tels que $f(x_i) = f(x_j)$.
- Grâce au paradoxe des anniversaires, cette méthode, pourtant rudimentaire, peut se révéler être très efficace surtout si la taille de l'empreinte est trop petite.
- Si, par exemple, la taille de l'empreinte est de 64 bits, il faudra environ $7,2 \times 10^9$ essais pour que l'on ait au moins 75% de chances de découvrir une collision (on suppose que toutes les empreintes sont équiprobables). C'est tout à fait à la portée d'un PC. Il faut également comparer ce nombre au nombre d'empreintes possibles, à savoir environ $1,8 \times 10^{19}$.
- Avec une empreinte de 512 bits et pour une même probabilité, il ne faudra pas moins de $1,9 \times 10^{77}$ essais, ce qui est hors de portée d'autant plus que cette attaque nécessite beaucoup de mémoire.

Attaque des anniversaires

- Soit f une fonction de hachage. On choisit *aléatoirement* $x_1, x_2, \ldots \in A^*$ jusqu'à ce qu'on trouve une collision, c'est à dire x_i, x_j avec $i \neq j$ et tels que $f(x_i) = f(x_j)$.
- Grâce au paradoxe des anniversaires, cette méthode, pourtant rudimentaire, peut se révéler être très efficace surtout si la taille de l'empreinte est trop petite.
- Si, par exemple, la taille de l'empreinte est de 64 bits, il faudra environ $7,2 \times 10^9$ essais pour que l'on ait au moins 75% de chances de découvrir une collision (on suppose que toutes les empreintes sont équiprobables). C'est tout à fait à la portée d'un PC. Il faut également comparer ce nombre au nombre d'empreintes possibles, à savoir environ $1,8 \times 10^{19}$.
- Avec une empreinte de 512 bits et pour une même probabilité, il ne faudra pas moins de $1,9 \times 10^{77}$ essais, ce qui est hors de portée d'autant plus que cette attaque nécessite beaucoup de mémoire.

Attaque des anniversaires

- Soit f une fonction de hachage. On choisit *aléatoirement* $x_1, x_2, \ldots \in A^*$ jusqu'à ce qu'on trouve une collision, c'est à dire x_i, x_j avec $i \neq j$ et tels que $f(x_i) = f(x_j)$.
- Grâce au paradoxe des anniversaires, cette méthode, pourtant rudimentaire, peut se révéler être très efficace surtout si la taille de l'empreinte est trop petite.
- Si, par exemple, la taille de l'empreinte est de 64 bits, il faudra environ $7,2 \times 10^9$ essais pour que l'on ait au moins 75% de chances de découvrir une collision (on suppose que toutes les empreintes sont équiprobables). C'est tout à fait à la portée d'un PC. Il faut également comparer ce nombre au nombre d'empreintes possibles, à savoir environ $1,8 \times 10^{19}$.
- Avec une empreinte de 512 bits et pour une même probabilité, il ne faudra pas moins de $1,9 \times 10^{77}$ essais, ce qui est hors de portée d'autant plus que cette attaque nécessite beaucoup de mémoire.

Faut-il avoir peur des collisions ?

- Si, pour une fonction de hachage f, on sait produire une collision, cela ne veut pas dire pour autant que cela puisse être utile à un attaquant potentiel. En effet les textes x et x' tels que $f(x) = f(x')$ ont toutes les chances d'être des suites de bits sans signification particulière.
- Bien plus difficile, mais toujours sans grand intérêt pour un attaquant, est de, x **étant donné**, trouver x' tel que $f(x) = f(x')$. Là encore, x' sera presque sûrement une suite de bits sans signification particulière.
- Encore plus difficile est de, dans la situation précédente, trouver un texte x' faisant sens, "voisin" de x et tel que $f(x) = f(x')$. L'attaque décrite précédemment ne permet pas de réaliser cela.
- Il faut utiliser *l'attaque avec préfixes choisis* : soient p_1 et p_2 deux préfixes **donnés**, il faut trouver deux suffixes s_1 et s_2 tels que $f(p_1 \parallel s_1) = f(p_2 \parallel s_2)$. Cette attaque a permis en 2007 de construire deux certificats X.509 pour des **noms de domaine différents** ayant une **même empreinte MD5** !

Faut-il avoir peur des collisions ?

- Si, pour une fonction de hachage f, on sait produire une collision, cela ne veut pas dire pour autant que cela puisse être utile à un attaquant potentiel. En effet les textes x et x' tels que $f(x) = f(x')$ ont toutes les chances d'être des suites de bits sans signification particulière.
- Bien plus difficile, mais toujours sans grand intérêt pour un attaquant, est de, x **étant donné**, trouver x' tel que $f(x) = f(x')$. Là encore, x' sera presque sûrement une suite de bits sans signification particulière.
- Encore plus difficile est de, dans la situation précédente, trouver un texte x' faisant sens, "voisin" de x et tel que $f(x) = f(x')$. L'attaque décrite précédemment ne permet pas de réaliser cela.
- Il faut utiliser *l'attaque avec préfixes choisis* : soient p_1 et p_2 deux préfixes **donnés**, il faut trouver deux suffixes s_1 et s_2 tels que $f(p_1 \parallel s_1) = f(p_2 \parallel s_2)$. Cette attaque a permis en 2007 de construire deux certificats X.509 pour des **noms de domaine différents** ayant une **même empreinte MD5** !

Faut-il avoir peur des collisions ?

- Si, pour une fonction de hachage f, on sait produire une collision, cela ne veut pas dire pour autant que cela puisse être utile à un attaquant potentiel. En effet les textes x et x' tels que $f(x) = f(x')$ ont toutes les chances d'être des suites de bits sans signification particulière.
- Bien plus difficile, mais toujours sans grand intérêt pour un attaquant, est de, x **étant donné**, trouver x' tel que $f(x) = f(x')$. Là encore, x' sera presque sûrement une suite de bits sans signification particulière.
- Encore plus difficile est de, dans la situation précédente, trouver un texte x' faisant sens, "voisin" de x et tel que $f(x) = f(x')$. L'attaque décrite précédemment ne permet pas de réaliser cela.
- Il faut utiliser *l'attaque avec préfixes choisis* : soient p_1 et p_2 deux préfixes **donnés**, il faut trouver deux suffixes s_1 et s_2 tels que $f(p_1 \parallel s_1) = f(p_2 \parallel s_2)$. Cette attaque a permis en 2007 de construire deux certificats X.509 pour des **noms de domaine différents** ayant une **même empreinte MD5** !

Faut-il avoir peur des collisions ?

- Si, pour une fonction de hachage f, on sait produire une collision, cela ne veut pas dire pour autant que cela puisse être utile à un attaquant potentiel. En effet les textes x et x' tels que $f(x) = f(x')$ ont toutes les chances d'être des suites de bits sans signification particulière.
- Bien plus difficile, mais toujours sans grand intérêt pour un attaquant, est de, x **étant donné**, trouver x' tel que $f(x) = f(x')$. Là encore, x' sera presque sûrement une suite de bits sans signification particulière.
- Encore plus difficile est de, dans la situation précédente, trouver un texte x' faisant sens, "voisin" de x et tel que $f(x) = f(x')$. L'attaque décrite précédemment ne permet pas de réaliser cela.
- Il faut utiliser *l'attaque avec préfixes choisis* : soient p_1 et p_2 deux préfixes **donnés**, il faut trouver deux suffixes s_1 et s_2 tels que $f(p_1 \parallel s_1) = f(p_2 \parallel s_2)$. Cette attaque a permis en 2007 de construire deux certificats X.509 pour des **noms de domaine différents** ayant une **même empreinte MD5** !

Quelques fonctions de hachage

- **MD5** : *empreinte de 128 bits*. On sait depuis 2004 qu'il est peu sûr. On a vu qu'il était sensible aux attaques avec préfixes choisis. Il est recommandé de ne plus l'utiliser.
- **SHA-1** : *empreinte de 160 bits*. Encore très utilisé. Néanmoins, en 2005, *Wang et al.* ont annoncé qu'ils pouvaient produire une collision en environ 2^{63} opérations, ce qui est à la limite du faisable. C'est donc une faiblesse toute relative d'autant que les textes provoquant ces collisions sont sans signification.
- **SHA-2** : *empreinte de 224, 256, 384 ou 512 bits*. Cette famille de fonctions de hachage a été introduite en vue du remplacement de SHA-1. Pour l'instant aucune attaque significative n'existe.
- **SHA-3** : *empreinte de 224, 256, 384, 512 ou arbitraire bits*. Les fonctions de hachage précédentes, étant construites à partir des mêmes heuristiques, sont sensibles aux mêmes attaques. Il a donc été decidé (*NIST*) de fournir une alternative fondée sur des principes complètement différents. C'est SHA-3 normalisée en 2015.

Quelques fonctions de hachage

- **MD5** : *empreinte de 128 bits*. On sait depuis 2004 qu'il est peu sûr. On a vu qu'il était sensible aux attaques avec préfixes choisis. Il est recommandé de ne plus l'utiliser.
- **SHA-1** : *empreinte de 160 bits*. Encore très utilisé. Néanmoins, en 2005, *Wang et al.* ont annoncé qu'ils pouvaient produire une collision en environ 2^{63} opérations, ce qui est à la limite du faisable. C'est donc une faiblesse toute relative d'autant que les textes provoquant ces collisions sont sans signification.
- **SHA-2** : *empreinte de 224, 256, 384 ou 512 bits*. Cette famille de fonctions de hachage a été introduite en vue du remplacement de SHA-1. Pour l'instant aucune attaque significative n'existe.
- **SHA-3** : *empreinte de 224, 256, 384, 512 ou arbitraire bits*. Les fonctions de hachage précédentes, étant construites à partir des mêmes heuristiques, sont sensibles aux mêmes attaques. Il a donc été decidé (*NIST*) de fournir une alternative fondée sur des principes complètement différents. C'est SHA-3 normalisée en 2015.

Quelques fonctions de hachage

- **MD5** : *empreinte de 128 bits*. On sait depuis 2004 qu'il est peu sûr. On a vu qu'il était sensible aux attaques avec préfixes choisis. Il est recommandé de ne plus l'utiliser.
- **SHA-1** : *empreinte de 160 bits*. Encore très utilisé. Néanmoins, en 2005, *Wang et al.* ont annoncé qu'ils pouvaient produire une collision en environ 2^{63} opérations, ce qui est à la limite du faisable. C'est donc une faiblesse toute relative d'autant que les textes provoquant ces collisions sont sans signification.
- **SHA-2** : *empreinte de 224, 256, 384 ou 512 bits*. Cette famille de fonctions de hachage a été introduite en vue du remplacement de SHA-1. Pour l'instant aucune attaque significative n'existe.
- **SHA-3** : *empreinte de 224, 256, 384, 512 ou arbitraire bits*. Les fonctions de hachage précédentes, étant construites à partir des mêmes heuristiques, sont sensibles aux mêmes attaques. Il a donc été decidé (*NIST*) de fournir une alternative fondée sur des principes complètement différents. C'est SHA-3 normalisée en 2015.

Quelques fonctions de hachage

- **MD5** : *empreinte de 128 bits*. On sait depuis 2004 qu'il est peu sûr. On a vu qu'il était sensible aux attaques avec préfixes choisis. Il est recommandé de ne plus l'utiliser.
- **SHA-1** : *empreinte de 160 bits*. Encore très utilisé. Néanmoins, en 2005, *Wang et al.* ont annoncé qu'ils pouvaient produire une collision en environ 2^{63} opérations, ce qui est à la limite du faisable. C'est donc une faiblesse toute relative d'autant que les textes provoquant ces collisions sont sans signification.
- **SHA-2** : *empreinte de 224, 256, 384 ou 512 bits*. Cette famille de fonctions de hachage a été introduite en vue du remplacement de SHA-1. Pour l'instant aucune attaque significative n'existe.
- **SHA-3** : *empreinte de 224, 256, 384, 512 ou arbitraire bits*. Les fonctions de hachage précédentes, étant construites à partir des mêmes heuristiques, sont sensibles aux mêmes attaques. Il a donc été decidé (*NIST*) de fournir une alternative fondée sur des principes complètement différents. C'est SHA-3 normalisée en 2015.

Exemples en Python

```
>>> import hashlib # le module implémentant les fonctions
                   # de hachage vues précédemment
>>> m = hashlib.md5()
>>> m.update(b"Ceci␣est␣un␣cours")
>>> m.update(b"de␣cryptographie")
>>> m.hexdigest()
'ca357cd2d4b8c84ac96a580900c01024'
>>> m.digest_size
16
>>> s = hashlib.sha224()
>>> s.update(b"Ceci␣est␣un␣cours␣de␣cryptographie")
>>> s.hexdigest()
'8a08715e34a37939e33b1570dcd0ab445dc63354f77151cc026e2144'
>>> s.digest_size
28
```

Définition

- Un code d'authentification de message (en abrégé CAM ou MAC - *Message Authentication Code*) est un dispositif permettant de garantir **l'intégrité des données** transmises à travers un canal non sécurisé entre deux entités, c'est à dire de garantir que les données n'ont pas été modifiées pendant leur acheminement.
- Ce dispositif permet également au destinataire d'authentifier l'expéditeur des données.
- Un CAM repose sur l'utilisation d'un secret partagé entre les parties en présence et, le plus souvent, sur une fonction à sens unique du type fonction de hachage comme MD5 ou SHA-1.
- En résumé, un CAM est très similaire à une fonction de hachage hormis l'usage d'une clé secrète.

Définition

- Un code d'authentification de message (en abrégé CAM ou MAC - *Message Authentication Code*) est un dispositif permettant de garantir **l'intégrité des données** transmises à travers un canal non sécurisé entre deux entités, c'est à dire de garantir que les données n'ont pas été modifiées pendant leur acheminement.
- Ce dispositif permet également au destinataire d'authentifier l'expéditeur des données.
- Un CAM repose sur l'utilisation d'un secret partagé entre les parties en présence et, le plus souvent, sur une fonction à sens unique du type fonction de hachage comme MD5 ou SHA-1.
- En résumé, un CAM est très similaire à une fonction de hachage hormis l'usage d'une clé secrète.

Définition

- Un code d'authentification de message (en abrégé CAM ou MAC - *Message Authentication Code*) est un dispositif permettant de garantir **l'intégrité des données** transmises à travers un canal non sécurisé entre deux entités, c'est à dire de garantir que les données n'ont pas été modifiées pendant leur acheminement.
- Ce dispositif permet également au destinataire d'authentifier l'expéditeur des données.
- Un CAM repose sur l'utilisation d'un secret partagé entre les parties en présence et, le plus souvent, sur une fonction à sens unique du type fonction de hachage comme MD5 ou SHA-1.
- En résumé, un CAM est très similaire à une fonction de hachage hormis l'usage d'une clé secrète.

Définition

- Un code d'authentification de message (en abrégé CAM ou MAC - *Message Authentication Code*) est un dispositif permettant de garantir **l'intégrité des données** transmises à travers un canal non sécurisé entre deux entités, c'est à dire de garantir que les données n'ont pas été modifiées pendant leur acheminement.
- Ce dispositif permet également au destinataire d'authentifier l'expéditeur des données.
- Un CAM repose sur l'utilisation d'un secret partagé entre les parties en présence et, le plus souvent, sur une fonction à sens unique du type fonction de hachage comme MD5 ou SHA-1.
- En résumé, un CAM est très similaire à une fonction de hachage hormis l'usage d'une clé secrète.

Schéma de fonctionnement

Schéma de fonctionnement

Schéma de fonctionnement

Schéma de fonctionnement

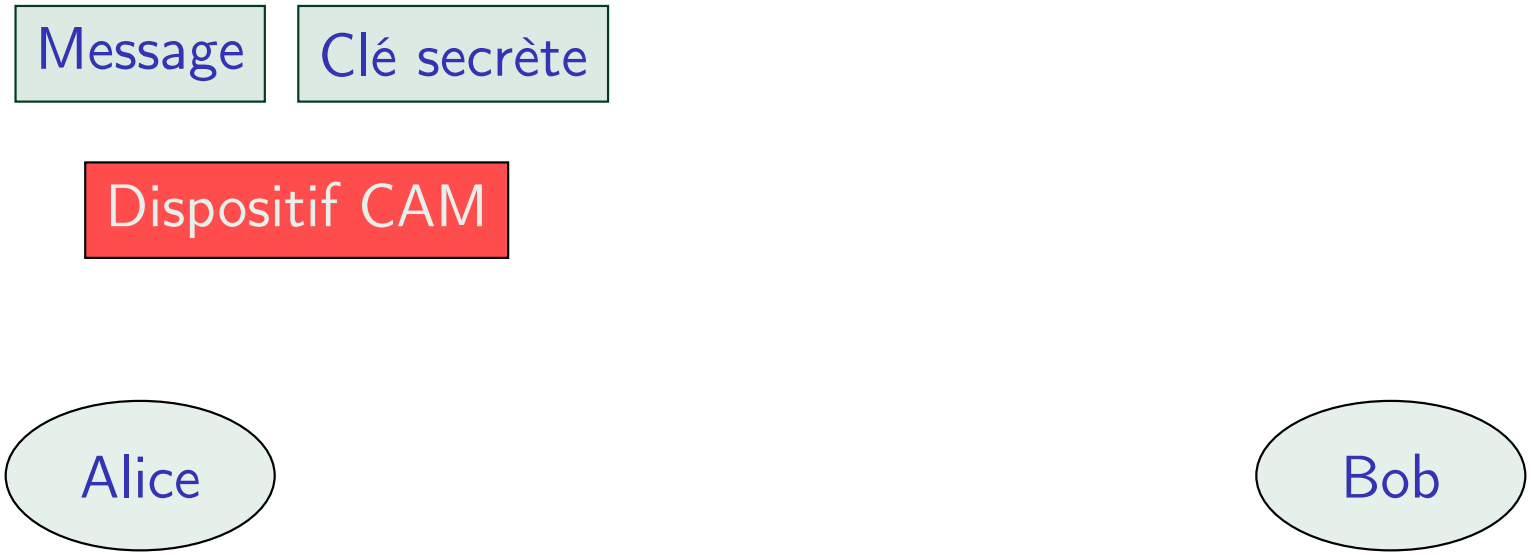


Schéma de fonctionnement

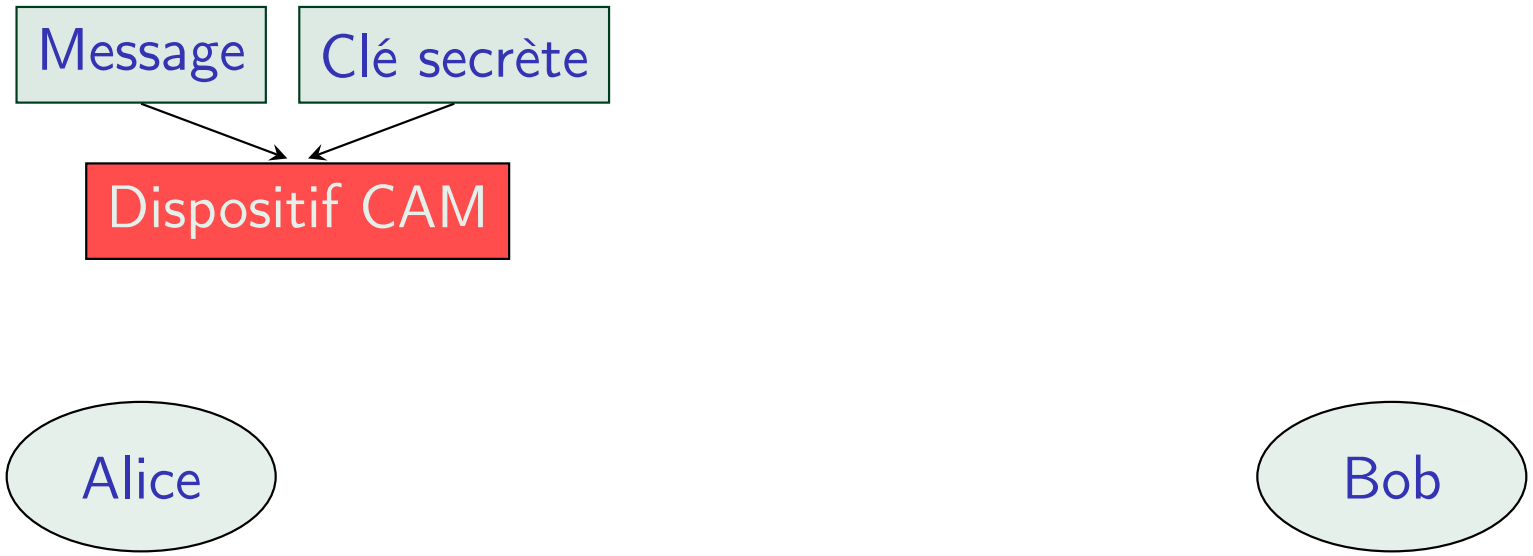


Schéma de fonctionnement

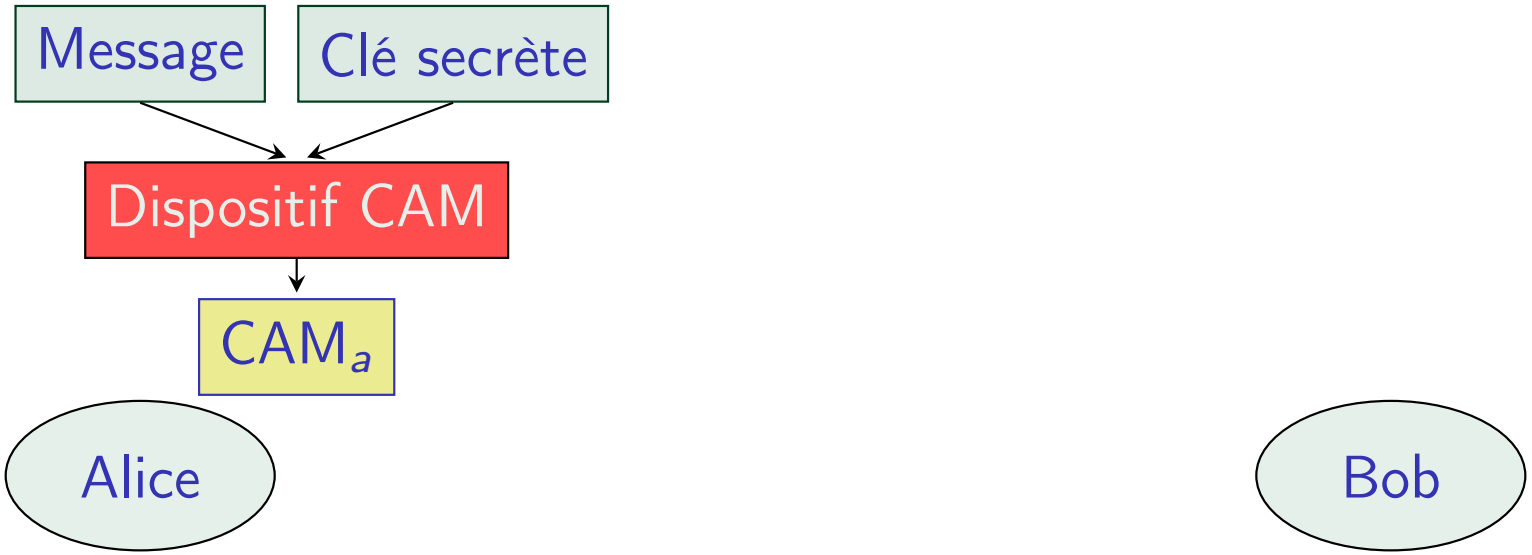


Schéma de fonctionnement

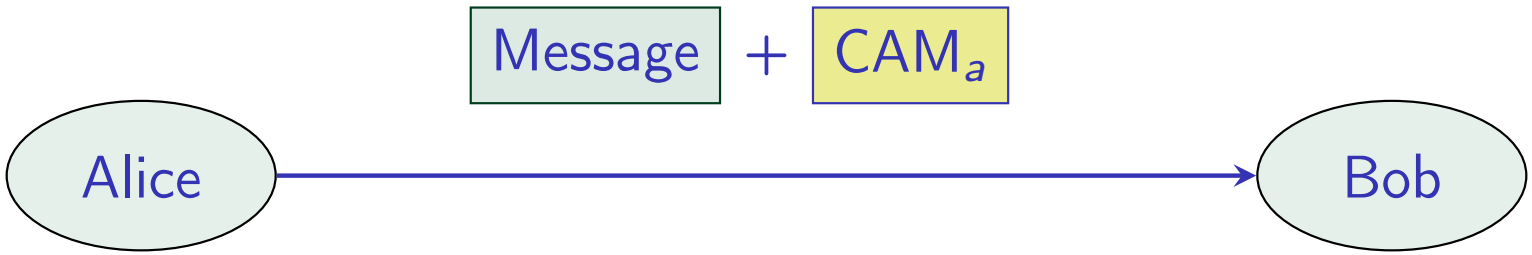


Schéma de fonctionnement

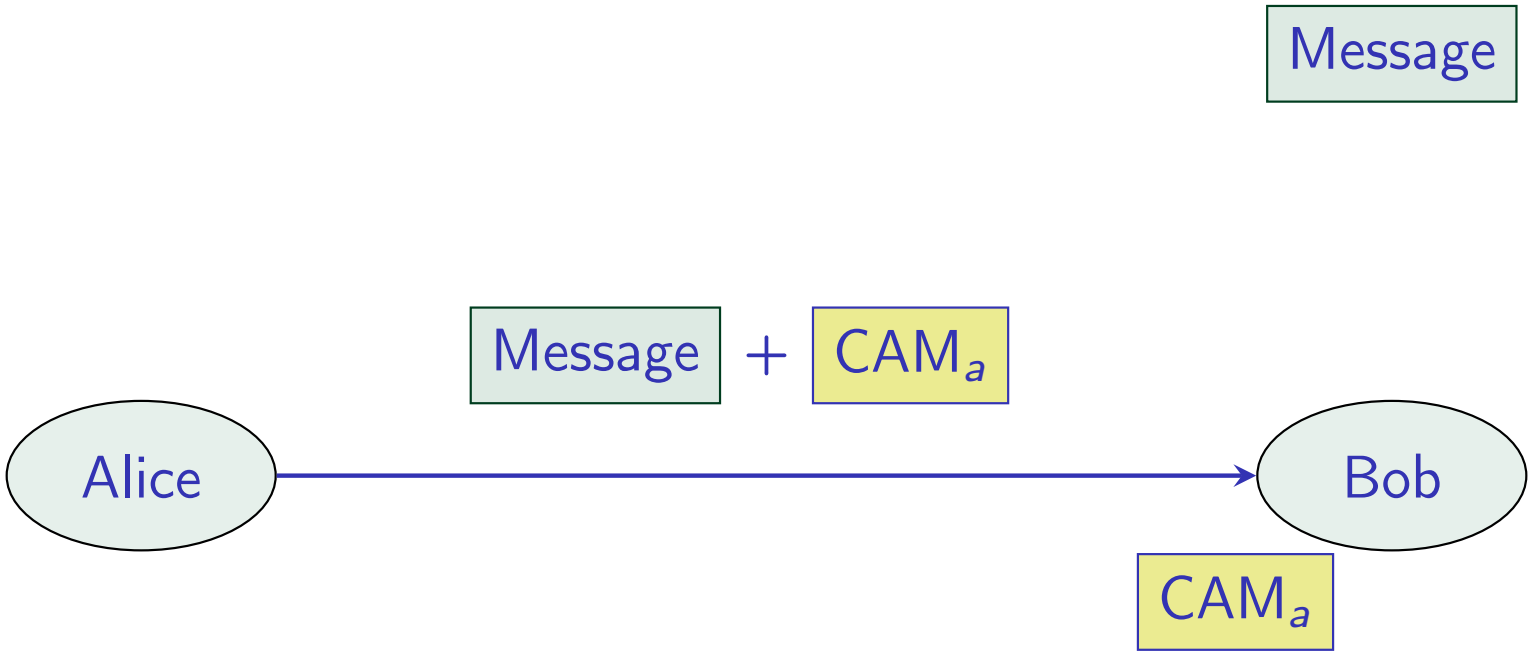


Schéma de fonctionnement

Schéma de fonctionnement

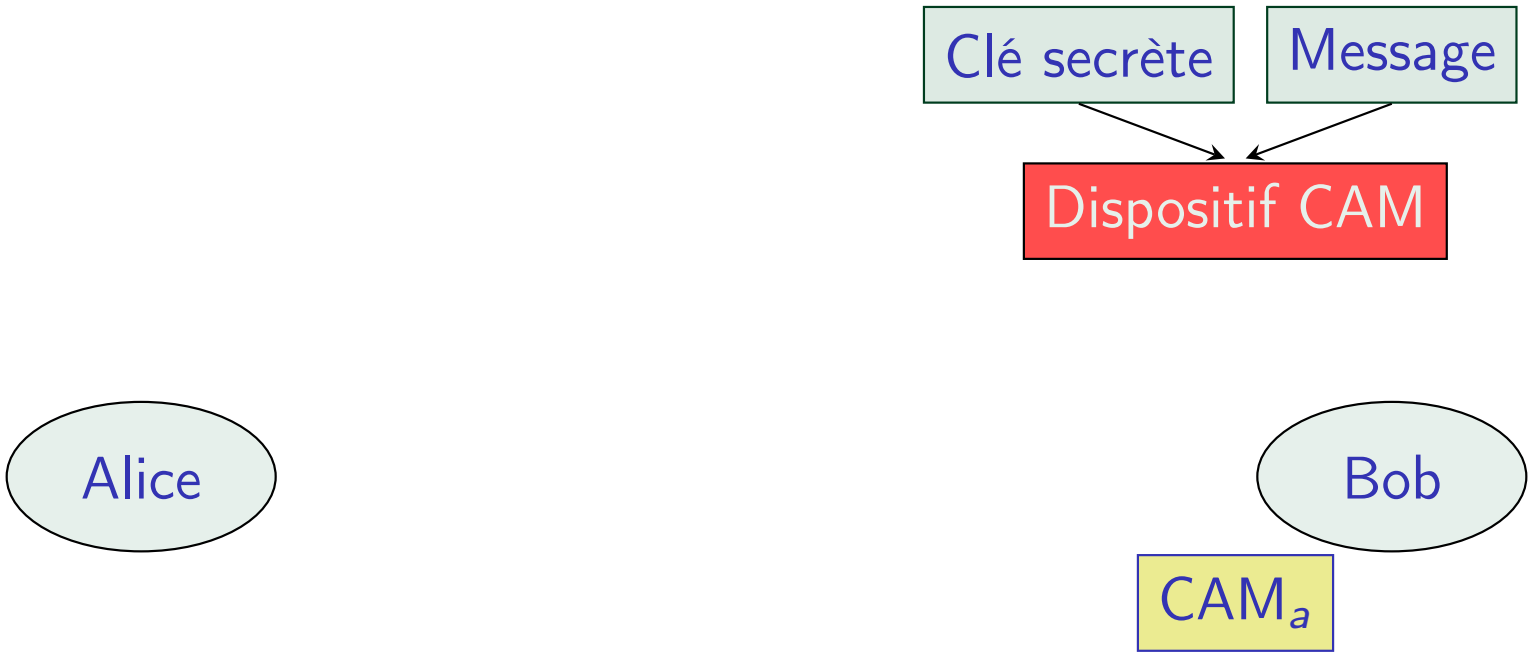


Schéma de fonctionnement

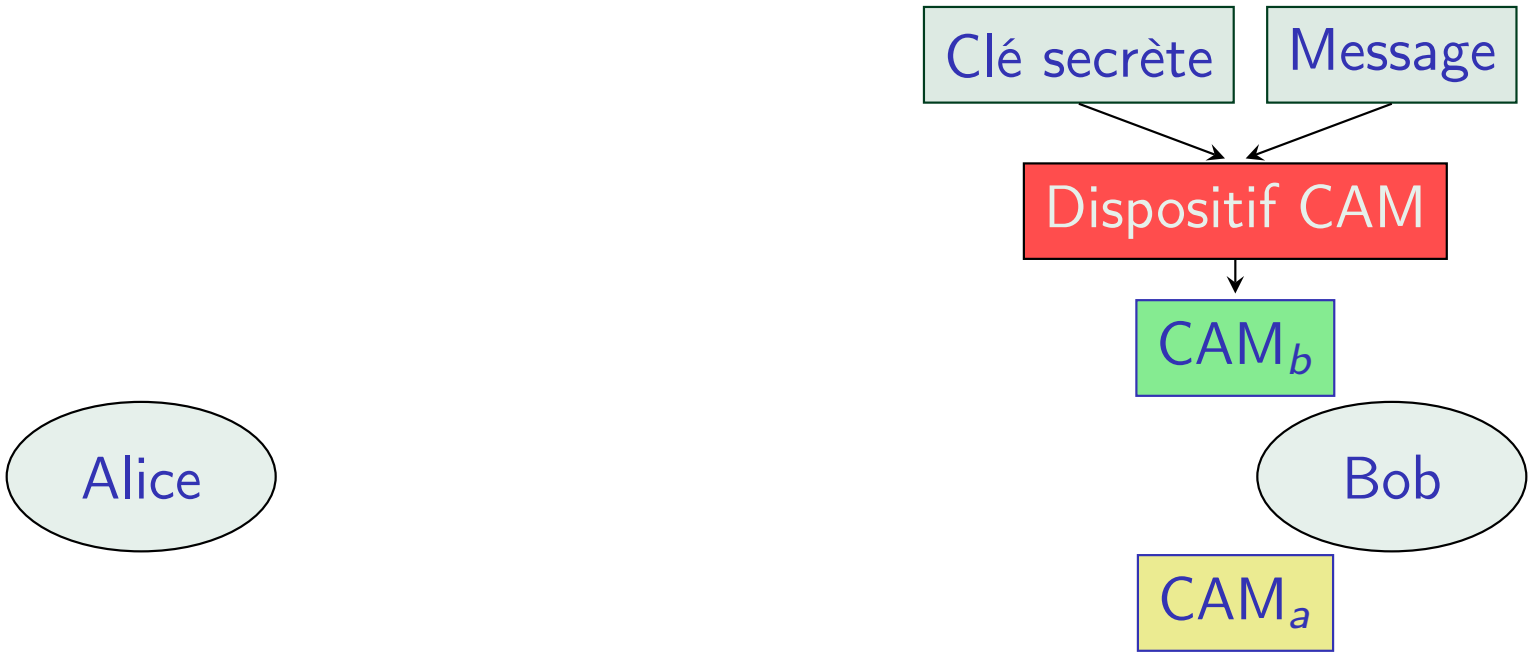
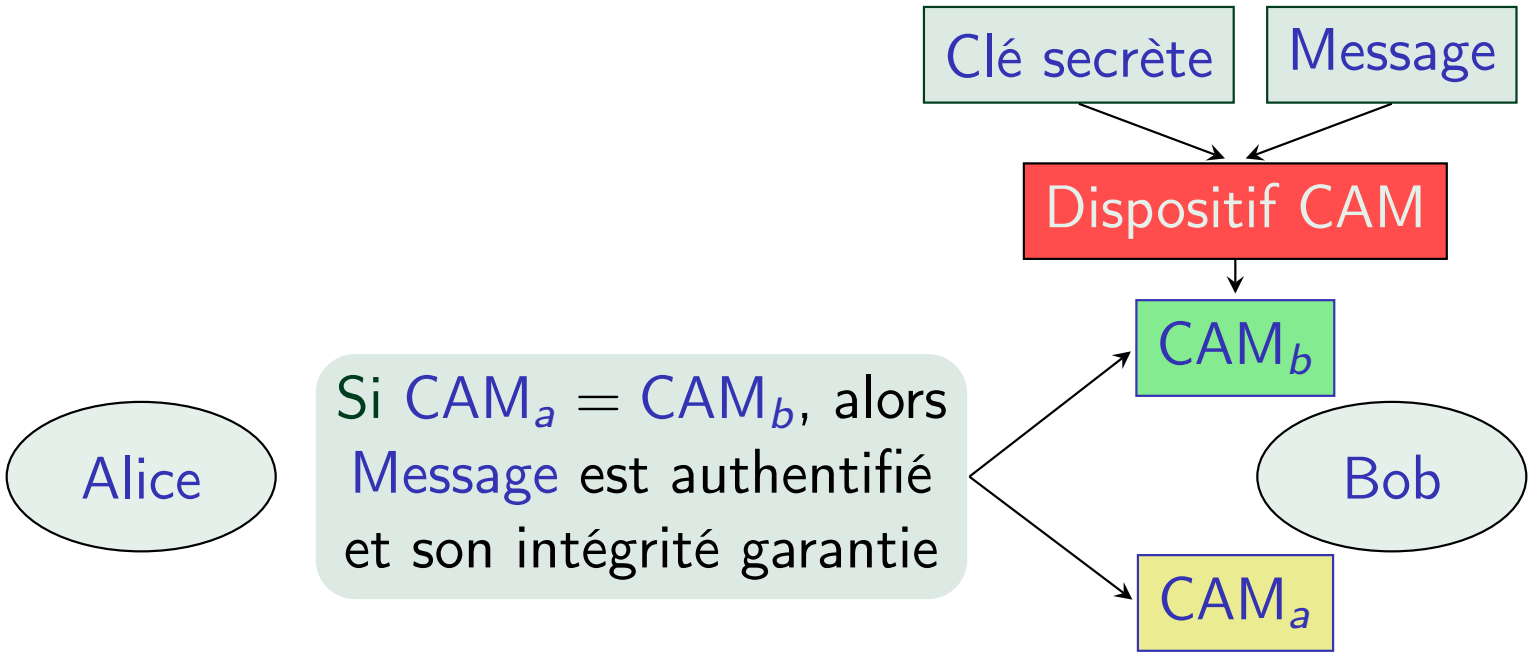


Schéma de fonctionnement



HMAC : *Keyed-Hashing for Message Authentication*

- HMAC est le code d'authentification de message le plus utilisé à l'heure actuelle (normalisé dans le ▸ RFC-2104).
- N'importe quelle fonction (itérative) de hachage peut-être utilisée avec HMAC. On parle alors de HMAC-MD5, HMAC-SHA-1, HMAC-SHA-224,...
- Soit h une fonction de hachage opérant sur des blocs de s octets (par exemple, $s = 64$ pour $h =$ MD5 ou $h =$ SHA-256) et soient m, k respectivement le message à authentifier et la clé secrète, on a :

$$HMAC_h(m, k) = h((k' \oplus opad) \parallel h((k' \oplus ipad) \parallel m)),$$

avec :

- $k' = \overbrace{k \parallel 0 \cdots 0}^{s \text{ octects}}$ si taille$(k) \leq s$. Si taille$(k) > s$, $k' = h(k)$.
- $opad = 0x\underbrace{5c \cdots 5c}_{s \text{ octects}}$, $ipad = 0x\underbrace{36 \cdots 36}_{s \text{ octects}}$

HMAC : *Keyed-Hashing for Message Authentication*

- HMAC est le code d'authentification de message le plus utilisé à l'heure actuelle (normalisé dans le ▸ RFC-2104).
- N'importe quelle fonction (itérative) de hachage peut-être utilisée avec HMAC. On parle alors de HMAC-MD5, HMAC-SHA-1, HMAC-SHA-224,...
- Soit h une fonction de hachage opérant sur des blocs de s octets (par exemple, $s = 64$ pour $h = \text{MD5}$ ou $h = \text{SHA-256}$) et soient m, k respectivement le message à authentifier et la clé secrète, on a :

$$HMAC_h(m, k) = h((k' \oplus opad) \parallel h((k' \oplus ipad) \parallel m)),$$

avec :

- $k' = \overbrace{k \parallel 0 \cdots 0}^{s \text{ octects}}$ si taille$(k) \leq s$. Si taille$(k) > s$, $k' = h(k)$.
- $opad = 0x\underbrace{5c \cdots 5c}_{s \text{ octects}}$, $ipad = 0x\underbrace{36 \cdots 36}_{s \text{ octects}}$

HMAC : *Keyed-Hashing for Message Authentication*

- HMAC est le code d'authentification de message le plus utilisé à l'heure actuelle (normalisé dans le ▸ RFC-2104).
- N'importe quelle fonction (itérative) de hachage peut-être utilisée avec HMAC. On parle alors de HMAC-MD5, HMAC-SHA-1, HMAC-SHA-224,...
- Soit h une fonction de hachage opérant sur des blocs de s octets (par exemple, $s = 64$ pour $h =$ MD5 ou $h =$ SHA-256) et soient m, k respectivement le message à authentifier et la clé secrète, on a :

$$HMAC_h(m, k) = h((k' \oplus opad) \parallel h((k' \oplus ipad) \parallel m)),$$

avec :

- $k' = \overbrace{k \parallel 0 \cdots 0}^{s \text{ octects}}$ si taille$(k) \leq s$. Si taille$(k) > s$, $k' = h(k)$.
- $opad = 0x\underbrace{5c \cdots 5c}_{s \text{ octects}}$, $ipad = 0x\underbrace{36 \cdots 36}_{s \text{ octects}}$

HMAC : un exemple en Python

```
>>> import hashlib # module fonctions de hachage
>>> import hmac    # module implémentant HMAC
>>> h = hmac.new(
          b"mon secret",          # clé secrète
          digestmod=hashlib.sha1 # fonction de hachage
        )
>>> h.update(b"Ceci est un cours")
>>> h.update(b"de cryptographie")
>>> h.hexdigest()
'3439f36eae2b94e65b056fd4cd21695839fdc862'
>>> h.digest_size
20 # 160 bits
>>> h.name
'hmac-sha1'
```

Les ensembles d'entiers

On suppose connus l'ensemble des entiers **naturels** $\mathbb{N} = \{0, 1, 2, \ldots\}$ et l'ensemble des entiers **relatifs** $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ ainsi que les propriétés élémentaires des quatres opérations : addition, soustraction, multiplication et division.

Théorème (division euclidienne dans $\mathbb{N}$)

Soient $a, b \in \mathbb{N}$ avec $b \neq 0$. Alors, il existe $q, r \in \mathbb{N}$, uniques, tels que :

$$a = bq + r \quad \text{avec } 0 \leq r < b.$$

Corollaire (division euclidienne dans $\mathbb{Z}$)

Soient $a \in \mathbb{Z}$ et $b \in \mathbb{N}$ avec $b \neq 0$. Alors, il existe $q \in \mathbb{Z}$ et $r \in \mathbb{N}$, uniques, tels que :

$$a = bq + r \quad \text{avec } 0 \leq r < b.$$

Les ensembles d'entiers

On suppose connus l'ensemble des entiers **naturels** $\mathbb{N} = \{0, 1, 2, \ldots\}$ et l'ensemble des entiers **relatifs** $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ ainsi que les propriétés élémentaires des quatres opérations : addition, soustraction, multiplication et division.

Théorème (division euclidienne dans $\mathbb{N}$)

Soient $a, b \in \mathbb{N}$ avec $b \neq 0$. Alors, il existe $q, r \in \mathbb{N}$, uniques, tels que :

$$a = bq + r \quad \text{avec } 0 \leq r < b.$$

Corollaire (division euclidienne dans $\mathbb{Z}$)

Soient $a \in \mathbb{Z}$ et $b \in \mathbb{N}$ avec $b \neq 0$. Alors, il existe $q \in \mathbb{Z}$ et $r \in \mathbb{N}$, uniques, tels que :

$$a = bq + r \quad \text{avec } 0 \leq r < b.$$

Les ensembles d'entiers

On suppose connus l'ensemble des entiers **naturels** $\mathbb{N} = \{0, 1, 2, \ldots\}$ et l'ensemble des entiers **relatifs** $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ ainsi que les propriétés élémentaires des quatres opérations : addition, soustraction, multiplication et division.

Théorème (division euclidienne dans $\mathbb{N}$)

Soient $a, b \in \mathbb{N}$ avec $b \neq 0$. Alors, il existe $q, r \in \mathbb{N}$, uniques, tels que :

$$a = bq + r \quad \text{avec } 0 \leq r < b.$$

Corollaire (division euclidienne dans $\mathbb{Z}$)

Soient $a \in \mathbb{Z}$ et $b \in \mathbb{N}$ avec $b \neq 0$. Alors, il existe $q \in \mathbb{Z}$ et $r \in \mathbb{N}$, uniques, tels que :

$$a = bq + r \quad \text{avec } 0 \leq r < b.$$

Les ensembles d'entiers

Le théorème précédent est une conséquence immédiate de la propriété fondamentale de l'ensemble des entiers naturels :

Propriété ($\mathbb{N}$ est bien ordonné)

Toute partie non vide de $\mathbb{N}$ a un plus petit élément.

Cette propriété est d'ailleurs équivalente au principe de récurrence :

Propriété (principe de récurrence)

Soit $P(n)$ un propriété de l'entier $n \in \mathbb{N}$. Supposons que l'on ait :

(1) *$P(0)$ est vraie,*

(2) *$\forall n \in \mathbb{N}, P(n) \Rightarrow P(n+1)$.*

Alors $P(n)$ est vraie pout tout $n \in \mathbb{N}$.

Les ensembles d'entiers

Le théorème précédent est une conséquence immédiate de la propriété fondamentale de l'ensemble des entiers naturels :

Propriété ($\mathbb{N}$ est bien ordonné)

Toute partie non vide de $\mathbb{N}$ a un plus petit élément.

Cette propriété est d'ailleurs équivalente au principe de récurrence :

Propriété (principe de récurrence)

Soit $P(n)$ un propriété de l'entier $n \in \mathbb{N}$. Supposons que l'on ait :

(1) *$P(0)$ est vraie,*

(2) *$\forall n \in \mathbb{N}, P(n) \Rightarrow P(n+1)$.*

Alors $P(n)$ est vraie pout tout $n \in \mathbb{N}$.

Définition et propriétés élémentaires

La notion de divisibilité est une notion centrale en arithmétique élémentaire.

Définition

Soient $a, b \in \mathbb{Z}$. On dit que b **divise** *a et on écrit $b \mid a$ s'il existe $q \in \mathbb{Z}$ tel que $a = bq$. On dit également que b est un* **diviseur** *de a ou que a est un* **multiple** *de b.*

Remarque

On voit que 0 ne divise que lui-même alors que tout entier $n \in \mathbb{Z}$ divise 0. Par ailleurs, les entiers 1 et -1 divisent tous les entiers.

Proposition

Pour tout $n \in \mathbb{Z}$, on a les propriétés suivantes :

(1) *si n divise a et b, alors n divise $a - b$ et $a + b$,*

(2) *si n divise a, alors n divise ka pour tout $k \in \mathbb{Z}$.*

Définition et propriétés élémentaires

La notion de divisibilité est une notion centrale en arithmétique élémentaire.

Définition

Soient $a, b \in \mathbb{Z}$. On dit que b **divise** *a et on écrit $b \mid a$ s'il existe $q \in \mathbb{Z}$ tel que $a = bq$. On dit également que b est un* **diviseur** *de a ou que a est un* **multiple** *de b.*

Remarque

On voit que 0 ne divise que lui-même alors que tout entier $n \in \mathbb{Z}$ divise 0. Par ailleurs, les entiers 1 et -1 divisent tous les entiers.

Proposition

Pour tout $n \in \mathbb{Z}$, on a les propriétés suivantes :

(1) *si n divise a et b, alors n divise $a - b$ et $a + b$,*

(2) *si n divise a, alors n divise ka pour tout $k \in \mathbb{Z}$.*

Définition et propriétés élémentaires

La notion de divisibilité est une notion centrale en arithmétique élémentaire.

Définition

Soient $a, b \in \mathbb{Z}$. On dit que b **divise** *a et on écrit $b \mid a$ s'il existe $q \in \mathbb{Z}$ tel que $a = bq$. On dit également que b est un* **diviseur** *de a ou que a est un* **multiple** *de b.*

Remarque

On voit que 0 ne divise que lui-même alors que tout entier $n \in \mathbb{Z}$ divise 0. Par ailleurs, les entiers 1 et -1 divisent tous les entiers.

Proposition

Pour tout $n \in \mathbb{Z}$, on a les propriétés suivantes :

1. *si n divise a et b, alors n divise $a - b$ et $a + b$,*
2. *si n divise a, alors n divise ka pour tout $k \in \mathbb{Z}$.*

Définition et propriétés élémentaires

La notion de divisibilité est une notion centrale en arithmétique élémentaire.

Définition

Soient $a, b \in \mathbb{Z}$. On dit que b **divise** *a et on écrit $b \mid a$ s'il existe $q \in \mathbb{Z}$ tel que $a = bq$. On dit également que b est un* **diviseur** *de a ou que a est un* **multiple** *de b.*

Remarque

On voit que 0 ne divise que lui-même alors que tout entier $n \in \mathbb{Z}$ divise 0. Par ailleurs, les entiers 1 et -1 divisent tous les entiers.

Proposition

Pour tout $n \in \mathbb{Z}$, on a les propriétés suivantes :

(1) *si n divise a et b, alors n divise $a - b$ et $a + b$,*

(2) *si n divise a, alors n divise ka pour tout $k \in \mathbb{Z}$.*

PGCD

Proposition (plus grand commun diviseur)

Soient $a, b \in \mathbb{Z}$ **non tous deux nuls**. *Alors l'ensemble des diviseurs* > 0 *communs à* a *et* b *admet un plus grand élément appelé plus grand commun diviseur de* a *et* b. *On le note :* $\text{pgcd}(a, b)$.

Attention : $\text{pgcd}(0, 0)$ n'a pas de sens.

Exemples

- $\text{pgcd}(24, 54) = 6$ ($54 = 9 \times 6$ *et* $24 = 6 \times 4$),
- $\text{pgcd}(1, b) = 1$ *pour tout* $b \in \mathbb{Z}$,
- $\text{pgcd}(0, b) = b$ *pour tout* $b > 0$.

Définition

Soient $a, b \in \mathbb{Z}$. a *et* b *sont dits* **premiers entre eux** *si* $\text{pgcd}(a, b) = 1$.

PGCD

Proposition (plus grand commun diviseur)

Soient $a, b \in \mathbb{Z}$ **non tous deux nuls**. *Alors l'ensemble des diviseurs* > 0 *communs à* a *et* b *admet un plus grand élément appelé plus grand commun diviseur de* a *et* b. *On le note :* $\text{pgcd}(a, b)$.

Attention : $\text{pgcd}(0, 0)$ n'a pas de sens.

Exemples

- $\text{pgcd}(24, 54) = 6$ ($54 = 9 \times 6$ *et* $24 = 6 \times 4$),
- $\text{pgcd}(1, b) = 1$ *pour tout* $b \in \mathbb{Z}$,
- $\text{pgcd}(0, b) = b$ *pour tout* $b > 0$.

Définition

Soient $a, b \in \mathbb{Z}$. a *et* b *sont dits* **premiers entre eux** *si* $\text{pgcd}(a, b) = 1$.

PGCD

Proposition (plus grand commun diviseur)

Soient $a, b \in \mathbb{Z}$ **non tous deux nuls**. *Alors l'ensemble des diviseurs* > 0 *communs à* a *et* b *admet un plus grand élément appelé plus grand commun diviseur de* a *et* b. *On le note :* $\text{pgcd}(a, b)$.

Attention : $\text{pgcd}(0, 0)$ n'a pas de sens.

Exemples

- $\text{pgcd}(24, 54) = 6$ ($54 = 9 \times 6$ *et* $24 = 6 \times 4$),
- $\text{pgcd}(1, b) = 1$ *pour tout* $b \in \mathbb{Z}$,
- $\text{pgcd}(0, b) = b$ *pour tout* $b > 0$.

Définition

Soient $a, b \in \mathbb{Z}$. a *et* b *sont dits* **premiers entre eux** *si* $\text{pgcd}(a, b) = 1$.

PGCD

Proposition (plus grand commun diviseur)

Soient $a, b \in \mathbb{Z}$ **non tous deux nuls**. *Alors l'ensemble des diviseurs* > 0 *communs à* a *et* b *admet un plus grand élément appelé plus grand commun diviseur de* a *et* b. *On le note :* $\text{pgcd}(a, b)$.

Attention : $\text{pgcd}(0, 0)$ n'a pas de sens.

Exemples

- $\text{pgcd}(24, 54) = 6$ ($54 = 9 \times 6$ *et* $24 = 6 \times 4$),
- $\text{pgcd}(1, b) = 1$ *pour tout* $b \in \mathbb{Z}$,
- $\text{pgcd}(0, b) = b$ *pour tout* $b > 0$.

Définition

Soient $a, b \in \mathbb{Z}$. a *et* b *sont dits* **premiers entre eux** *si* $\text{pgcd}(a, b) = 1$.

PGCD

Proposition (plus grand commun diviseur)

Soient $a, b \in \mathbb{Z}$ **non tous deux nuls**. *Alors l'ensemble des diviseurs* > 0 *communs à* a *et* b *admet un plus grand élément appelé plus grand commun diviseur de* a *et* b. *On le note :* $\text{pgcd}(a, b)$.

Attention : $\text{pgcd}(0, 0)$ n'a pas de sens.

Exemples

- $\text{pgcd}(24, 54) = 6$ ($54 = 9 \times 6$ *et* $24 = 6 \times 4$),
- $\text{pgcd}(1, b) = 1$ *pour tout* $b \in \mathbb{Z}$,
- $\text{pgcd}(0, b) = b$ *pour tout* $b > 0$.

Définition

Soient $a, b \in \mathbb{Z}$. a *et* b *sont dits* **premiers entre eux** *si* $\text{pgcd}(a, b) = 1$.

PGCD

Proposition (plus grand commun diviseur)

Soient $a, b \in \mathbb{Z}$ **non tous deux nuls**. *Alors l'ensemble des diviseurs* > 0 *communs à* a *et* b *admet un plus grand élément appelé plus grand commun diviseur de* a *et* b. *On le note :* $\text{pgcd}(a, b)$.

Attention : $\text{pgcd}(0, 0)$ n'a pas de sens.

Exemples

- $\text{pgcd}(24, 54) = 6$ ($54 = 9 \times 6$ *et* $24 = 6 \times 4$),
- $\text{pgcd}(1, b) = 1$ *pour tout* $b \in \mathbb{Z}$,
- $\text{pgcd}(0, b) = b$ *pour tout* $b > 0$.

Définition

Soient $a, b \in \mathbb{Z}$. a *et* b *sont dits* **premiers entre eux** *si* $\text{pgcd}(a, b) = 1$.

PGCD

Proposition (plus grand commun diviseur)

Soient $a, b \in \mathbb{Z}$ **non tous deux nuls**. *Alors l'ensemble des diviseurs* > 0 *communs à a et b admet un plus grand élément appelé plus grand commun diviseur de a et b. On le note :* $\text{pgcd}(a, b)$.

Attention : $\text{pgcd}(0, 0)$ n'a pas de sens.

Exemples

- $\text{pgcd}(24, 54) = 6$ ($54 = 9 \times 6$ *et* $24 = 6 \times 4$),
- $\text{pgcd}(1, b) = 1$ *pour tout* $b \in \mathbb{Z}$,
- $\text{pgcd}(0, b) = b$ *pour tout* $b > 0$.

Définition

Soient $a, b \in \mathbb{Z}$. *a et b sont dits* **premiers entre eux** *si* $\text{pgcd}(a, b) = 1$.

Définition et exemples

Définition

Soit $n \in \mathbb{N}$, $n > 0$. On dit que $a, b \in \mathbb{Z}$ sont **congrus modulo** *n et on note $a \equiv b \pmod{n}$ si n divise $a - b$.*

Exemples

- *Modulo 1, la notion de congruence n'a aucun intérêt car tous les nombres sont congrus.*
- *Modulo 2, un nombre est congru à 0 ou 1 selon qu'il est pair ou impair.*
- *Modulo 7, les jours de la semaine.*
- *Modulo 10, un nombre (en écriture décimale) est congru à son chiffre des unités : $2016 \equiv 6 \pmod{10}$.*
- *Modulo 24, les heures du jour.*

Définition et exemples

Définition

Soit $n \in \mathbb{N}$, $n > 0$. *On dit que* $a, b \in \mathbb{Z}$ *sont* **congrus modulo** n *et on note* $a \equiv b \pmod{n}$ *si* n *divise* $a - b$.

Exemples

- *Modulo* 1, *la notion de congruence n'a aucun intérêt car tous les nombres sont congrus.*
- *Modulo* 2, *un nombre est congru à* 0 *ou* 1 *selon qu'il est pair ou impair.*
- *Modulo* 7, *les jours de la semaine.*
- *Modulo* 10, *un nombre (en écriture décimale) est congru à son chiffre des unités :* $2016 \equiv 6 \pmod{10}$.
- *Modulo* 24, *les heures du jour.*

Définition et exemples

Définition

Soit $n \in \mathbb{N}$, $n > 0$. On dit que $a, b \in \mathbb{Z}$ sont **congrus modulo** *n et on note $a \equiv b \pmod{n}$ si n divise $a - b$.*

Exemples

- *Modulo 1, la notion de congruence n'a aucun intérêt car tous les nombres sont congrus.*
- *Modulo 2, un nombre est congru à 0 ou 1 selon qu'il est pair ou impair.*
- *Modulo 7, les jours de la semaine.*
- *Modulo 10, un nombre (en écriture décimale) est congru à son chiffre des unités : $2016 \equiv 6 \pmod{10}$.*
- *Modulo 24, les heures du jour.*

Définition et exemples

Définition

Soit $n \in \mathbb{N}$, $n > 0$. *On dit que* $a, b \in \mathbb{Z}$ *sont* **congrus modulo** n *et on note* $a \equiv b \pmod{n}$ *si* n *divise* $a - b$.

Exemples

- *Modulo* 1, *la notion de congruence n'a aucun intérêt car tous les nombres sont congrus.*
- *Modulo* 2, *un nombre est congru à* 0 *ou* 1 *selon qu'il est pair ou impair.*
- *Modulo* 7, *les jours de la semaine.*
- *Modulo* 10, *un nombre (en écriture décimale) est congru à son chiffre des unités :* $2016 \equiv 6 \pmod{10}$.
- *Modulo* 24, *les heures du jour.*

Définition et exemples

Définition

Soit $n \in \mathbb{N}$, $n > 0$. *On dit que* $a, b \in \mathbb{Z}$ *sont* **congrus modulo** n *et on note* $a \equiv b \pmod{n}$ *si* n *divise* $a - b$.

Exemples

- *Modulo* 1, *la notion de congruence n'a aucun intérêt car tous les nombres sont congrus.*
- *Modulo* 2, *un nombre est congru à* 0 *ou* 1 *selon qu'il est pair ou impair.*
- *Modulo* 7, *les jours de la semaine.*
- *Modulo* 10, *un nombre (en écriture décimale) est congru à son chiffre des unités :* $2016 \equiv 6 \pmod{10}$.
- *Modulo* 24, *les heures du jour.*

Définition et exemples

Définition

Soit $n \in \mathbb{N}$, $n > 0$. *On dit que* $a, b \in \mathbb{Z}$ *sont* **congrus modulo** n *et on note* $a \equiv b \pmod{n}$ *si* n *divise* $a - b$.

Exemples

- *Modulo* 1, *la notion de congruence n'a aucun intérêt car tous les nombres sont congrus.*
- *Modulo* 2, *un nombre est congru à* 0 *ou* 1 *selon qu'il est pair ou impair.*
- *Modulo* 7, *les jours de la semaine.*
- *Modulo* 10, *un nombre (en écriture décimale) est congru à son chiffre des unités :* $2016 \equiv 6 \pmod{10}$.
- *Modulo* 24, *les heures du jour.*

Définition et exemples

Définition

Soit $n \in \mathbb{N}$, $n > 0$. *On dit que* $a, b \in \mathbb{Z}$ *sont* **congrus modulo** n *et on note* $a \equiv b \pmod{n}$ *si* n *divise* $a - b$.

Exemples

- *Modulo* 1, *la notion de congruence n'a aucun intérêt car tous les nombres sont congrus.*
- *Modulo* 2, *un nombre est congru à* 0 *ou* 1 *selon qu'il est pair ou impair.*
- *Modulo* 7, *les jours de la semaine.*
- *Modulo* 10, *un nombre (en écriture décimale) est congru à son chiffre des unités :* $2016 \equiv 6 \pmod{10}$.
- *Modulo* 24, *les heures du jour.*

Propriétés

Proposition

La relation de congruence modulo n est une relation d'équivalence dans $\mathbb{Z}$. Autrement dit, modulo n, on a :

(1) $a \equiv a$ *(réflexivité),*

(2) $a \equiv b \Rightarrow b \equiv a$ *(symétrie),*

(3) $a \equiv b$ *et* $b \equiv c \Rightarrow a \equiv c$ *(transitivité).*

Proposition

Soit $n \in \mathbb{N}$, $n > 0$. Tout nombre $a \in \mathbb{Z}$ est congru modulo n à un et un seul des nombres $\{0, \ldots, n-1\}$.

Démonstration.

Il suffit d'effectuer la division euclidienne de a par n. □

Propriétés

Proposition

La relation de congruence modulo n est une relation d'équivalence dans $\mathbb{Z}$. Autrement dit, modulo n, on a :

(1) $a \equiv a$ *(réflexivité),*

(2) $a \equiv b \Rightarrow b \equiv a$ *(symétrie),*

(3) $a \equiv b$ *et* $b \equiv c \Rightarrow a \equiv c$ *(transitivité).*

Proposition

Soit $n \in \mathbb{N}$, $n > 0$. Tout nombre $a \in \mathbb{Z}$ est congru modulo n à un et un seul des nombres $\{0, \ldots, n-1\}$.

Démonstration.

Il suffit d'effectuer la division euclidienne de a par n. □

Propriétés

Proposition

La relation de congruence modulo n est une relation d'équivalence dans $\mathbb{Z}$. Autrement dit, modulo n, on a :

(1) $a \equiv a$ *(réflexivité),*

(2) $a \equiv b \Rightarrow b \equiv a$ *(symétrie),*

(3) $a \equiv b$ *et* $b \equiv c \Rightarrow a \equiv c$ *(transitivité).*

Proposition

Soit $n \in \mathbb{N}$, $n > 0$. Tout nombre $a \in \mathbb{Z}$ est congru modulo n à un et un seul des nombres $\{0, \ldots, n-1\}$.

Démonstration.

Il suffit d'effectuer la division euclidienne de a par n. □

Propriétés

Proposition (compatibilité avec l'addition et la multiplication)

Soit $n \in \mathbb{N}$, $n > 0$. Supposons que l'on ait, modulo n, $a \equiv a'$ et $b \equiv b'$, alors on a :

(1) $a + b \equiv a' + b' \pmod{n}$,

(2) $ab \equiv a'b' \pmod{n}$,

(3) $a^k \equiv a'^k \pmod{n}$ *pour tout* $k \in \mathbb{N}$.

Proposition (inverse modulo n pour la multiplication)

Soient $a, n \in \mathbb{N}$, $a, n > 0$ et **premiers entre eux**. *Alors il existe $b > 0$ tel que $ab \equiv 1 \pmod{n}$.*

Remarque

La condition a, n premiers entre eux est essentielle. Par exemple 2 n'est pas inversible modulo 4 car $2 \times 2 \equiv 0 \pmod{4}$.

Propriétés

Proposition (compatibilité avec l'addition et la multiplication)

Soit $n \in \mathbb{N}$, $n > 0$. Supposons que l'on ait, modulo n, $a \equiv a'$ et $b \equiv b'$, alors on a :

(1) $a + b \equiv a' + b' \pmod{n}$,

(2) $ab \equiv a'b' \pmod{n}$,

(3) $a^k \equiv a'^k \pmod{n}$ *pour tout* $k \in \mathbb{N}$.

Proposition (inverse modulo n pour la multiplication)

Soient $a, n \in \mathbb{N}$, $a, n > 0$ et **premiers entre eux**. *Alors il existe $b > 0$ tel que $ab \equiv 1 \pmod{n}$.*

Remarque

La condition a, n premiers entre eux est essentielle. Par exemple 2 n'est pas inversible modulo 4 car $2 \times 2 \equiv 0 \pmod{4}$.

Propriétés

Proposition (compatibilité avec l'addition et la multiplication)

Soit $n \in \mathbb{N}$, $n > 0$. Supposons que l'on ait, modulo n, $a \equiv a'$ et $b \equiv b'$, alors on a :

(1) $a + b \equiv a' + b' \pmod{n}$,

(2) $ab \equiv a'b' \pmod{n}$,

(3) $a^k \equiv a'^k \pmod{n}$ *pour tout* $k \in \mathbb{N}$.

Proposition (inverse modulo n pour la multiplication)

Soient $a, n \in \mathbb{N}$, $a, n > 0$ et **premiers entre eux**. *Alors il existe $b > 0$ tel que $ab \equiv 1 \pmod{n}$.*

Remarque

La condition a, n premiers entre eux est essentielle. Par exemple 2 n'est pas inversible modulo 4 car $2 \times 2 \equiv 0 \pmod{4}$.

Applications

Critère de divisibilité par 3 : soit $a \in \mathbb{N}$ dont l'écriture décimale est

$$a = a_n \times 10^n + a_{n-1} \times 10^{n-1} + \cdots + a_1 \times 10 + a_0,$$

avec $0 \leq a_i \leq 9$ pour $i = 0, \ldots, n$. Alors, comme $10 \equiv 1 \pmod 3$ on a :

$$a \equiv a_n + a_{n-1} + \cdots + a_1 + a_0 \ (\text{ mod } 3).$$

Donc a est multiple de 3 si et seulement si la somme de ses chiffres en écriture décimale est un multiple de 3.

Equations diophantiennes : l'équation $x^2 - 5y^7 = 2$ n'a pas de solution entière. En effet, si cette équation avait une solution, on aurait :

$$x^2 \equiv 2 \ (\text{ mod } 5).$$

Or, modulo 5, $x^2 \equiv 0$ ou $x^2 \equiv 1$ ou $x^2 \equiv 4$, d'où contradiction.

Applications

Critère de divisibilité par 3 : soit $a \in \mathbb{N}$ dont l'écriture décimale est

$$a = a_n \times 10^n + a_{n-1} \times 10^{n-1} + \cdots + a_1 \times 10 + a_0,$$

avec $0 \leq a_i \leq 9$ pour $i = 0, \ldots, n$. Alors, comme $10 \equiv 1 \pmod 3$ on a :

$$a \equiv a_n + a_{n-1} + \cdots + a_1 + a_0 \ (\text{ mod } 3).$$

Donc a est multiple de 3 si et seulement si la somme de ses chiffres en écriture décimale est un multiple de 3.

Equations diophantiennes : l'équation $x^2 - 5y^7 = 2$ n'a pas de solution entière. En effet, si cette équation avait une solution, on aurait :

$$x^2 \equiv 2 \ (\text{ mod } 5).$$

Or, modulo 5, $x^2 \equiv 0$ ou $x^2 \equiv 1$ ou $x^2 \equiv 4$, d'où contradiction.

Qu'est-ce que l'exponentiation modulaire ?

Nous verrons qu'en cryptographie asymétrique, en particulier avec l'*algorithme RSA*, on a souvent besoin de calculer $b^e \pmod m$ explicitement où b, e, m sont de "grands" entiers naturels. Ce calcul s'appelle **exponentiation modulaire**. b est appelé la **base**, e l'**exposant** et m le **module**.

Par exemple, considérons le calcul de $341^{943} \pmod{1403}$. La méthode naïve consisterait à élever 341 à la puissance 943 puis effectuer la division euclidienne du résultat par 1403, le reste de cette division étant le nombre cherché.

Une première difficulté est d'avoir à effectuer 942 multiplications (en pratique, les nombres sont beaucoup plus grands) suivies d'une division euclidienne.

Mais la difficulté principale (rédhibitoire en pratique) est, qu'avec cette méthode, on est conduit à manipuler des nombres de plus en plus grands.

Qu'est-ce que l'exponentiation modulaire ?

Nous verrons qu'en cryptographie asymétrique, en particulier avec l'*algorithme RSA*, on a souvent besoin de calculer $b^e \pmod m$ explicitement où b, e, m sont de "grands" entiers naturels. Ce calcul s'appelle **exponentiation modulaire**. b est appelé la **base**, e l'**exposant** et m le **module**.

Par exemple, considérons le calcul de $341^{943} \pmod{1403}$. La méthode naïve consisterait à élever 341 à la puissance 943 puis effectuer la division euclidienne du résultat par 1403, le reste de cette division étant le nombre cherché.

Une première difficulté est d'avoir à effectuer 942 multiplications (en pratique, les nombres sont beaucoup plus grands) suivies d'une division euclidienne.

Mais la difficulté principale (rédhibitoire en pratique) est, qu'avec cette méthode, on est conduit à manipuler des nombres de plus en plus grands.

Qu'est-ce que l'exponentiation modulaire ?

Nous verrons qu'en cryptographie asymétrique, en particulier avec l'*algorithme RSA*, on a souvent besoin de calculer $b^e \pmod m$ explicitement où b, e, m sont de "grands" entiers naturels. Ce calcul s'appelle **exponentiation modulaire**. b est appelé la **base**, e l'**exposant** et m le **module**.

Par exemple, considérons le calcul de $341^{943} \pmod{1403}$. La méthode naïve consisterait à élever 341 à la puissance 943 puis effectuer la division euclidienne du résultat par 1403, le reste de cette division étant le nombre cherché.

Une première difficulté est d'avoir à effectuer 942 multiplications (en pratique, les nombres sont beaucoup plus grands) suivies d'une division euclidienne.

Mais la difficulté principale (rédhibitoire en pratique) est, qu'avec cette méthode, on est conduit à manipuler des nombres de plus en plus grands.

Qu'est-ce que l'exponentiation modulaire ?

Nous verrons qu'en cryptographie asymétrique, en particulier avec l'*algorithme RSA*, on a souvent besoin de calculer $b^e \pmod m$ explicitement où b, e, m sont de "grands" entiers naturels. Ce calcul s'appelle **exponentiation modulaire**. b est appelé la **base**, e l'**exposant** et m le **module**.

Par exemple, considérons le calcul de $341^{943} \pmod{1403}$. La méthode naïve consisterait à élever 341 à la puissance 943 puis effectuer la division euclidienne du résultat par 1403, le reste de cette division étant le nombre cherché.

Une première difficulté est d'avoir à effectuer 942 multiplications (en pratique, les nombres sont beaucoup plus grands) suivies d'une division euclidienne.

Mais la difficulté principale (rédhibitoire en pratique) est, qu'avec cette méthode, on est conduit à manipuler des nombres de plus en plus grands.

Exponentiation par carrés

Fort heureusement, il existe un algorithme efficace, appelé **exponentiation par carrés** (ou exponentiation rapide) qui combine deux opérations élémentaires : l'élévation au carré et la multiplication par la base b.

On commence par décomposer l'exposant e de la manière suivante : si e est pair, on le divise par 2, sinon on lui retranche 1 et on le divise par 2 et ensuite on recommence avec le quotient jusqu'à ce qu'on arrive (nécessairement) à 1. On écrit dans l'ordre inverse d'obtention cette suite de nombres. Dans notre exemple avec $e = 943$, cela donne la suite : 1, 2, 3, 6, 7, 14, 28, 29, 58, 116, 117, 234, 235, 470, 471, 942, 943.

Pour $b = 341$, on a, modulo 1403 : $b^1 \equiv 341$, $b^2 \equiv 1235$, $b^3 \equiv 235$, $b^6 \equiv 508$, $b^7 \equiv 659$, $b^{14} \equiv 754$, $b^{28} \equiv 301$, $b^{29} \equiv 222$, $b^{58} \equiv 179$, $b^{116} \equiv 1175$, $b^{117} \equiv 820$, $b^{234} \equiv 363$, $b^{235} \equiv 319$, $b^{470} \equiv 745$, $b^{471} \equiv 102$, $b^{942} \equiv 583$ et $b^{943} \equiv 980$. Le calcul a nécessité seulement **16** multiplications au lieu de **942** avec la méthode naïve !

Exponentiation par carrés

Fort heureusement, il existe un algorithme efficace, appelé **exponentiation par carrés** (ou exponentiation rapide) qui combine deux opérations élémentaires : l'élévation au carré et la multiplication par la base b.

On commence par décomposer l'exposant e de la manière suivante : si e est pair, on le divise par 2, sinon on lui retranche 1 et on le divise par 2 et ensuite on recommence avec le quotient jusqu'à ce qu'on arrive (nécessairement) à 1. On écrit dans l'ordre inverse d'obtention cette suite de nombres. Dans notre exemple avec $e = 943$, cela donne la suite : 1, 2, 3, 6, 7, 14, 28, 29, 58, 116, 117, 234, 235, 470, 471, 942, 943.

Pour $b = 341$, on a, modulo 1403 : $b^1 \equiv 341$, $b^2 \equiv 1235$, $b^3 \equiv 235$, $b^6 \equiv 508$, $b^7 \equiv 659$, $b^{14} \equiv 754$, $b^{28} \equiv 301$, $b^{29} \equiv 222$, $b^{58} \equiv 179$, $b^{116} \equiv 1175$, $b^{117} \equiv 820$, $b^{234} \equiv 363$, $b^{235} \equiv 319$, $b^{470} \equiv 745$, $b^{471} \equiv 102$, $b^{942} \equiv 583$ et $b^{943} \equiv 980$. Le calcul a nécessité seulement **16** multiplications au lieu de **942** avec la méthode naïve !

Exponentiation par carrés

Fort heureusement, il existe un algorithme efficace, appelé **exponentiation par carrés** (ou exponentiation rapide) qui combine deux opérations élémentaires : l'élévation au carré et la multiplication par la base b.

On commence par décomposer l'exposant e de la manière suivante : si e est pair, on le divise par 2, sinon on lui retranche 1 et on le divise par 2 et ensuite on recommence avec le quotient jusqu'à ce qu'on arrive (nécessairement) à 1. On écrit dans l'ordre inverse d'obtention cette suite de nombres. Dans notre exemple avec $e = 943$, cela donne la suite : 1, 2, 3, 6, 7, 14, 28, 29, 58, 116, 117, 234, 235, 470, 471, 942, 943.

Pour $b = 341$, on a, modulo 1403 : $b^1 \equiv 341$, $b^2 \equiv 1235$, $b^3 \equiv 235$, $b^6 \equiv 508$, $b^7 \equiv 659$, $b^{14} \equiv 754$, $b^{28} \equiv 301$, $b^{29} \equiv 222$, $b^{58} \equiv 179$, $b^{116} \equiv 1175$, $b^{117} \equiv 820$, $b^{234} \equiv 363$, $b^{235} \equiv 319$, $b^{470} \equiv 745$, $b^{471} \equiv 102$, $b^{942} \equiv 583$ et $b^{943} \equiv 980$. Le calcul a nécessité seulement **16** multiplications au lieu de **942** avec la méthode naïve !

Exponentiation par carrés - une implémentation en Python

L'algorithme d'exponentiation par carrés est particulièrement facile à programmer de manière efficace.

Un exemple d'implémentation en langage Python :

```
def powm(b, e, m): # retourne b^e (mod m)
  ret = 1
  while e > 0:
    if (e & 1 > 0): # e est impair
      ret = (ret * b) % m # on multiplie par la base b
    e >>= 1 # on divise e par 2
    b = (b * b) % m # on élève la base b au carré
  return ret
```

Exponentiation par carrés - une implémentation en Python

L'algorithme d'exponentiation par carrés est particulièrement facile à programmer de manière efficace.

Un exemple d'implémentation en langage Python :

```
def powm(b, e, m): # retourne b^e (mod m)
  ret = 1
  while e > 0:
    if (e & 1 > 0): # e est impair
      ret = (ret * b) % m # on multiplie par la base b
    e >>= 1 # on divise e par 2
    b = (b * b) % m # on élève la base b au carré
  return ret
```

Une définition

Les nombres premiers sont un ingrédient essentiel de la cryptographie asymétrique.

Définition

Un nombre $p \in \mathbb{N}$ est **premier** *s'il a exactement deux diviseurs, à savoir 1 et lui-même. L'ensemble des nombres premiers sera désigné par $\mathbb{P}$.*

Remarque

0 n'est pas un nombre premier car il a une infinité de diviseurs. 1 n'est pas premier non plus car il n'a qu'un seul diviseur : lui-même. Le plus petit nombre premier est donc 2 et c'est le seul qui soit pair.

Les 25 nombres premiers ≤ 100 sont : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89 et 97.

Une définition

Les nombres premiers sont un ingrédient essentiel de la cryptographie asymétrique.

Définition

Un nombre $p \in \mathbb{N}$ est **premier** *s'il a exactement deux diviseurs, à savoir 1 et lui-même. L'ensemble des nombres premiers sera désigné par $\mathbb{P}$.*

Remarque

0 n'est pas un nombre premier car il a une infinité de diviseurs. 1 n'est pas premier non plus car il n'a qu'un seul diviseur : lui-même. Le plus petit nombre premier est donc 2 et c'est le seul qui soit pair.

Les 25 nombres premiers ≤ 100 sont : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89 et 97.

Une définition

Les nombres premiers sont un ingrédient essentiel de la cryptographie asymétrique.

Définition

Un nombre $p \in \mathbb{N}$ est **premier** *s'il a exactement deux diviseurs, à savoir 1 et lui-même. L'ensemble des nombres premiers sera désigné par $\mathbb{P}$.*

Remarque

0 n'est pas un nombre premier car il a une infinité de diviseurs. 1 n'est pas premier non plus car il n'a qu'un seul diviseur : lui-même. Le plus petit nombre premier est donc 2 et c'est le seul qui soit pair.

Les 25 nombres premiers ≤ 100 sont : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89 et 97.

Une définition

Les nombres premiers sont un ingrédient essentiel de la cryptographie asymétrique.

Définition

Un nombre $p \in \mathbb{N}$ est **premier** *s'il a exactement deux diviseurs, à savoir 1 et lui-même. L'ensemble des nombres premiers sera désigné par $\mathbb{P}$.*

Remarque

0 n'est pas un nombre premier car il a une infinité de diviseurs. 1 n'est pas premier non plus car il n'a qu'un seul diviseur : lui-même. Le plus petit nombre premier est donc 2 et c'est le seul qui soit pair.

Les 25 nombres premiers ≤ 100 sont : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89 et 97.

Quelques propriétés

L'importance de la notion de nombre premier vient de :

Proposition

Tout entier $n \geq 2$ a un diviseur premier.

et surtout du résultat suivant, connu sous le nom de *théorème fondamental de l'arithmétique* :

Théorème

Tout nombre entier $n > 0$ peut s'écrire comme produit de nombres premiers. Cette écriture est unique à permutation près des facteurs (par convention, 1 est le produit de 0 nombres premiers). Par conséquent, si $n > 1$, il existe $p_1, \ldots, p_k \in \mathbb{P}$, deux à deux distincts, et tels que :

$$n = p_1^{\alpha_1} \cdots p_k^{\alpha_k} \quad \text{avec } \alpha_1, \ldots, \alpha_k > 0$$

Quelques propriétés

L'importance de la notion de nombre premier vient de :

Proposition

Tout entier $n \geq 2$ a un diviseur premier.

et surtout du résultat suivant, connu sous le nom de *théorème fondamental de l'arithmétique* :

Théorème

Tout nombre entier $n > 0$ peut s'écrire comme produit de nombres premiers. Cette écriture est unique à permutation près des facteurs (par convention, 1 est le produit de 0 nombres premiers). Par conséquent, si $n > 1$, il existe $p_1, \ldots, p_k \in \mathbb{P}$, deux à deux distincts, et tels que :

$$n = p_1^{\alpha_1} \cdots p_k^{\alpha_k} \quad \text{avec } \alpha_1, \ldots, \alpha_k > 0$$

Le théorème fondamental de l'arithmétique

- Ce théorème est essentiellement un théorème d'*existence*. Dès que l'entier n est suffisamment grand, il est impossible de trouver (avec les connaissances actuelles et le matériel existant), en un temps raisonnable, sa décomposition en facteurs premiers.
- C'est sur cette impossibilité pratique que repose une importante partie de la cryptographie moderne. En particulier, c'est pour cela que les nombres premiers sont un ingrédient indispensable à la cryptographie asymétrique.
- Attention : on n'a jamais démontré que cette impossibilité ne puisse être levée dans un avenir plus ou moins proche.
- Il n'est d'ailleurs pas impossible que des algorithmes efficaces aient été découverts et gardés secrets par des agences gouvernementales de certains pays...

Le théorème fondamental de l'arithmétique

- Ce théorème est essentiellement un théorème d'*existence*. Dès que l'entier n est suffisamment grand, il est impossible de trouver (avec les connaissances actuelles et le matériel existant), en un temps raisonnable, sa décomposition en facteurs premiers.
- C'est sur cette impossibilité pratique que repose une importante partie de la cryptographie moderne. En particulier, c'est pour cela que les nombres premiers sont un ingrédient indispensable à la cryptographie asymétrique.
- Attention : on n'a jamais démontré que cette impossibilité ne puisse être levée dans un avenir plus ou moins proche.
- Il n'est d'ailleurs pas impossible que des algorithmes efficaces aient été découverts et gardés secrets par des agences gouvernementales de certains pays...

Le théorème fondamental de l'arithmétique

- Ce théorème est essentiellement un théorème d'*existence*. Dès que l'entier n est suffisamment grand, il est impossible de trouver (avec les connaissances actuelles et le matériel existant), en un temps raisonnable, sa décomposition en facteurs premiers.
- C'est sur cette impossibilité pratique que repose une importante partie de la cryptographie moderne. En particulier, c'est pour cela que les nombres premiers sont un ingrédient indispensable à la cryptographie asymétrique.
- Attention : on n'a jamais démontré que cette impossibilité ne puisse être levée dans un avenir plus ou moins proche.
- Il n'est d'ailleurs pas impossible que des algorithmes efficaces aient été découverts et gardés secrets par des agences gouvernementales de certains pays...

Le théorème fondamental de l'arithmétique

- Ce théorème est essentiellement un théorème d'*existence*. Dès que l'entier n est suffisamment grand, il est impossible de trouver (avec les connaissances actuelles et le matériel existant), en un temps raisonnable, sa décomposition en facteurs premiers.
- C'est sur cette impossibilité pratique que repose une importante partie de la cryptographie moderne. En particulier, c'est pour cela que les nombres premiers sont un ingrédient indispensable à la cryptographie asymétrique.
- Attention : on n'a jamais démontré que cette impossibilité ne puisse être levée dans un avenir plus ou moins proche.
- Il n'est d'ailleurs pas impossible que des algorithmes efficaces aient été découverts et gardés secrets par des agences gouvernementales de certains pays...

Le théorème fondamental de l'arithmétique

- Qu'entend-on par nombre *suffisamment grand* ?
- Le standard actuel : **2048 bits**
- Par exemple (616 chiffres décimaux) :

$$n = 27181501784100193306238314721811525084879824584884951$$
$$43922208659327747456729632620341110318790783087578119$$
$$88354945632976688073516892655564201675092778522532130$$
$$96505921488580477578338115679761292593464587095750148$$
$$35840800489504431311432589111969411117851225424201670$$
$$24499076844587952474195010367164123918315913262324411$$
$$32066747856455291972764455456654526786324450548427366$$
$$40817940347088982448452369195007395936837490245723384$$
$$39181312954791179861446503691312760425390664714316177$$
$$13002254620672948682526462860597948221872157623282332$$
$$12065758686337428941714284792206236873372070130152061$$
$$72721183658853150915092413782978 29$$

Le théorème fondamental de l'arithmétique

- Qu'entend-on par nombre *suffisamment grand* ?
- Le standard actuel : **2048 bits**
- Par exemple (616 chiffres décimaux) :

$$n = 27181501784100193306238314721811525084879824584884951$$
$$43922208659327747456729632620341110318790783087578119$$
$$88354945632976688073516892655564201675092778522532130$$
$$96505921488580477578338115679761292593464587095750148$$
$$35840800489504431311432589111969411117851225424201670$$
$$24499076844587952474195010367164123918315913262324411$$
$$32066747856455291972764455456654526786324450548427366$$
$$40817940347088982448452369195007395936837490245723384$$
$$39181312954791179861446503691312760425390664714316177$$
$$13002254620672948682526462860597948221872157623282332$$
$$12065758686337428941714284792206236873372070130152061$$
$$72721183658853150915092413782978 29$$

Le théorème fondamental de l'arithmétique

- Qu'entend-on par nombre *suffisamment grand* ?
- Le standard actuel : **2048 bits**
- Par exemple (616 chiffres décimaux) :

$$
\begin{aligned}
n = & 2718150178410019330623831472181152508487982458488495143922208659327747456729632620341110318790783087578119\\
& 8835494563297668807351689265556420167509277852253213096505921488580477578338115679761292593464587095750148\\
& 3584080048950443131143258911196941111785122542420167024499076844587952474195010367164123918315913262324411\\
& 3206674785645529197276445545665452678632445054842736640817940347088982448452369195007395936837490245723384\\
& 3918131295479117986144650369131276042539066471431617713002254620672948682526462860597948221872157623282332\\
& 1206575868633742894171428479220623687337207013015206172721183658853150915092413782978 29
\end{aligned}
$$

Le théorème fondamental de l'arithmétique

La décomposition de n en facteurs premiers est :

$$n = p_1 p_2 \quad \text{avec,}$$

$p_1 =$ 173789406634605235433275465352155827959718312732138809982146366470670867626910547595457580161807963099697466284427556074035342769780415072027651326074077093642650598375788951179919071844709441737373877648992706041962525504704140264811882983796139366833331593930911411921095512555385848402504645461985065052893

$p_2 =$ 156404825302440312859363018019547590513737250137430265476663608950645987679170650138382963386476338043706409577044975024871419655486596370687970536532826091601806542507347405985561760953047918233248038572903407378538039094571243005478163623025191898580257892957898204724656524290627339955215605202512424336553

Combien y en a-t-il ?

Une infinité ! Ce résultat est connu depuis *Euclide*.

Démonstration.

Par l'absurde. Supposons qu'il y ait qu'un nombre fini de nombres premiers $p_1, \ldots, p_k$ et soit $n = p_1 \cdots p_k + 1$. Alors, comme on l'a vu, n a un diviseur premier qui, par construction, ne peut être aucun des p_i pour $i = 1, \ldots, k$, d'où contradiction. □

En 1737, *Euler* a établi un résultat beaucoup plus fin :

$$\sum_{p \in \mathbb{P}} \frac{1}{p} = +\infty$$

Plus précisément, il a montré que :

$$\sum_{p \leq x} \frac{1}{p} \sim \log \log x \quad \text{quand } x \to +\infty$$

Combien y en a-t-il ?

Une infinité ! Ce résultat est connu depuis *Euclide*.

Démonstration.

Par l'absurde. Supposons qu'il y ait qu'un nombre fini de nombres premiers $p_1, \ldots, p_k$ et soit $n = p_1 \cdots p_k + 1$. Alors, comme on l'a vu, n a un diviseur premier qui, par construction, ne peut être aucun des p_i pour $i = 1, \ldots, k$, d'où contradiction. □

En 1737, *Euler* a établi un résultat beaucoup plus fin :

$$\sum_{p \in \mathbb{P}} \frac{1}{p} = +\infty$$

Plus précisément, il a montré que :

$$\sum_{p \leq x} \frac{1}{p} \sim \log \log x \quad \text{quand } x \to +\infty$$

Combien y en a-t-il ?

Une infinité ! Ce résultat est connu depuis *Euclide*.

Démonstration.

Par l'absurde. Supposons qu'il y ait qu'un nombre fini de nombres premiers $p_1, \ldots, p_k$ et soit $n = p_1 \cdots p_k + 1$. Alors, comme on l'a vu, n a un diviseur premier qui, par construction, ne peut être aucun des p_i pour $i = 1, \ldots, k$, d'où contradiction. □

En 1737, *Euler* a établi un résultat beaucoup plus fin :

$$\sum_{p \in \mathbb{P}} \frac{1}{p} = +\infty$$

Plus précisément, il a montré que :

$$\sum_{p \leq x} \frac{1}{p} \sim \log \log x \quad \text{quand } x \to +\infty$$

Combien y en a-t-il ?

Une infinité ! Ce résultat est connu depuis *Euclide*.

Démonstration.

Par l'absurde. Supposons qu'il y ait qu'un nombre fini de nombres premiers $p_1, \ldots, p_k$ et soit $n = p_1 \cdots p_k + 1$. Alors, comme on l'a vu, n a un diviseur premier qui, par construction, ne peut être aucun des p_i pour $i = 1, \ldots, k$, d'où contradiction. □

En 1737, *Euler* a établi un résultat beaucoup plus fin :

$$\sum_{p \in \mathbb{P}} \frac{1}{p} = +\infty$$

Plus précisément, il a montré que :

$$\sum_{p \leq x} \frac{1}{p} \sim \log\log x \quad \text{quand } x \to +\infty$$

Combien y en a-t-il ?

Une infinité ! Ce résultat est connu depuis *Euclide*.

Démonstration.

Par l'absurde. Supposons qu'il y ait qu'un nombre fini de nombres premiers $p_1, \ldots, p_k$ et soit $n = p_1 \cdots p_k + 1$. Alors, comme on l'a vu, n a un diviseur premier qui, par construction, ne peut être aucun des p_i pour $i = 1, \ldots, k$, d'où contradiction. □

En 1737, *Euler* a établi un résultat beaucoup plus fin :

$$\sum_{p \in \mathbb{P}} \frac{1}{p} = +\infty$$

Plus précisément, il a montré que :

$$\sum_{p \leq x} \frac{1}{p} \sim \log\log x \quad \text{quand } x \to +\infty$$

Quelle est leur densité ?

Une première proposition énoncée par *Joseph Bertrand* au 19ème siècle :

Proposition (*postulat de Bertrand*)

Pour tout entier $n > 1$, on a $]n, 2n[\cap \mathbb{P} \neq \emptyset$.

Est-ce un renseignement utile d'un point de vue pratique ? Prenons $n = 2^{64}$, nous aurons au pire $2^{64} - 1$ tests de primalité à effectuer avant de trouver un nombre premier dans cet intervalle. Supposons que l'on puisse effectuer 10^9 de ces tests par seconde, ce qui est très optimiste, alors cela risque de nous prendre 585 années !

En fait, tout cela va dépendre de $card(]n, 2n[\cap \mathbb{P})$, donc de la densité des nombres premiers dans l'intervalle $]n, 2n[$. Nous allons voir que cette dernière est plutôt conséquente.

Quelle est leur densité ?

Une première proposition énoncée par *Joseph Bertrand* au 19ème siècle :

Proposition (*postulat de Bertrand*)

Pour tout entier $n > 1$, *on a* $]n, 2n[\cap \mathbb{P} \neq \emptyset$.

Est-ce un renseignement utile d'un point de vue pratique ? Prenons $n = 2^{64}$, nous aurons au pire $2^{64} - 1$ tests de primalité à effectuer avant de trouver un nombre premier dans cet intervalle. Supposons que l'on puisse effectuer 10^9 de ces tests par seconde, ce qui est très optimiste, alors cela risque de nous prendre 585 années !

En fait, tout cela va dépendre de $card(]n, 2n[\cap \mathbb{P})$, donc de la densité des nombres premiers dans l'intervalle $]n, 2n[$. Nous allons voir que cette dernière est plutôt conséquente.

Quelle est leur densité ?

Une première proposition énoncée par *Joseph Bertrand* au 19ème siècle :

Proposition (*postulat de Bertrand*)

Pour tout entier $n > 1$, *on a* $]n, 2n[\cap\mathbb{P} \neq \emptyset$.

Est-ce un renseignement utile d'un point de vue pratique ? Prenons $n = 2^{64}$, nous aurons au pire $2^{64} - 1$ tests de primalité à effectuer avant de trouver un nombre premier dans cet intervalle. Supposons que l'on puisse effectuer 10^9 de ces tests par seconde, ce qui est très optimiste, alors cela risque de nous prendre 585 années !

En fait, tout cela va dépendre de $card(]n, 2n[\cap\mathbb{P})$, donc de la densité des nombres premiers dans l'intervalle $]n, 2n[$. Nous allons voir que cette dernière est plutôt conséquente.

Quelle est leur densité ?

Une première proposition énoncée par *Joseph Bertrand* au 19ème siècle :

Proposition (*postulat de Bertrand*)

Pour tout entier $n > 1$, *on a* $]n, 2n[\cap \mathbb{P} \neq \emptyset$.

Est-ce un renseignement utile d'un point de vue pratique ? Prenons $n = 2^{64}$, nous aurons au pire $2^{64} - 1$ tests de primalité à effectuer avant de trouver un nombre premier dans cet intervalle. Supposons que l'on puisse effectuer 10^9 de ces tests par seconde, ce qui est très optimiste, alors cela risque de nous prendre 585 années !

En fait, tout cela va dépendre de $card(]n, 2n[\cap \mathbb{P})$, donc de la densité des nombres premiers dans l'intervalle $]n, 2n[$. Nous allons voir que cette dernière est plutôt conséquente.

Le théorème des nombres premiers

Ce théorème, démontré indépendamment par *Hadamard* et *de la Vallée Poussin* en 1896 donne une valeur approchée de $\pi(x)$, le nombre de nombres premiers $\leq x$.

Théorème (*Hadamard, de la Vallée Poussin*)

On a :

$$\pi(x) \sim \frac{x}{\log x} \quad \textit{quand } x \to \infty$$

ou, de manière équivalente, si p_n désigne le n-ième nombre premier :

$$p_n \sim n \log n \quad \textit{quand } n \to \infty$$

On peut en déduire que la probabilité pour qu'un entier n de b bits tiré au hasard soit premier est de l'ordre de $1/(b \log 2)$. Reprenant l'exemple précédent, si $n \in [2^{64}, 2^{65}[$, cette probabilité est d'environ $1/(64 \times log 2) \approx 2.25\,\%$, ce qui est loin d'être négligeable.

Le théorème des nombres premiers

Ce théorème, démontré indépendamment par *Hadamard* et *de la Vallée Poussin* en 1896 donne une valeur approchée de $\pi(x)$, le nombre de nombres premiers $\leq x$.

Théorème (*Hadamard, de la Vallée Poussin*)

On a :

$$\pi(x) \sim \frac{x}{\log x} \quad \textit{quand } x \to \infty$$

ou, de manière équivalente, si p_n désigne le n-ième nombre premier :

$$p_n \sim n \log n \quad \textit{quand } n \to \infty$$

On peut en déduire que la probabilité pour qu'un entier n de b bits tiré au hasard soit premier est de l'ordre de $1/(b \log 2)$. Reprenant l'exemple précédent, si $n \in [2^{64}, 2^{65}[$, cette probabilité est d'environ $1/(64 \times log 2) \approx 2.25\,\%$, ce qui est loin d'être négligeable.

Le théorème des nombres premiers

Ce théorème, démontré indépendamment par *Hadamard* et *de la Vallée Poussin* en 1896 donne une valeur approchée de $\pi(x)$, le nombre de nombres premiers $\leq x$.

Théorème (*Hadamard, de la Vallée Poussin*)

On a :

$$\pi(x) \sim \frac{x}{\log x} \quad \textit{quand } x \to \infty$$

ou, de manière équivalente, si p_n désigne le n-ième nombre premier :

$$p_n \sim n \log n \quad \textit{quand } n \to \infty$$

On peut en déduire que la probabilité pour qu'un entier n de b bits tiré au hasard soit premier est de l'ordre de $1/(b \log 2)$. Reprenant l'exemple précédent, si $n \in [2^{64}, 2^{65}[$, cette probabilité est d'environ $1/(64 \times log 2) \approx 2.25\,\%$, ce qui est loin d'être négligeable.

Le théorème des nombres premiers

La preuve (difficile) de ce théorème repose sur l'étude de la fonction ζ de *Riemann*. Elle est ainsi définie : $\zeta(s) = \sum_{n \geq 1} \frac{1}{n^s}$, pour $s > 1$.

En s'appuyant sur le théorème fondamental de l'arithmétique, *Euler* a démontré cette identité tout à fait remarquable :

$$\zeta(s) = \prod_{p \in \mathbb{P}} \frac{1}{1 - p^{-s}}$$

Le membre gauche de l'identité fait intervenir tous les entiers, tandis que le membre droit ne fait intervenir que les nombres premiers.

C'est l'identité d'*Euler* qui explique le rôle fondamental de la fonction ζ de *Riemann* dans l'étude des nombres premiers.

Le théorème des nombres premiers

La preuve (difficile) de ce théorème repose sur l'étude de la fonction ζ de *Riemann*. Elle est ainsi définie : $\zeta(s) = \sum_{n \geq 1} \frac{1}{n^s}$, pour $s > 1$.

En s'appuyant sur le théorème fondamental de l'arithmétique, *Euler* a démontré cette identité tout à fait remarquable :

$$\zeta(s) = \prod_{p \in \mathbb{P}} \frac{1}{1 - p^{-s}}$$

Le membre gauche de l'identité fait intervenir tous les entiers, tandis que le membre droit ne fait intervenir que les nombres premiers.

C'est l'identité d'*Euler* qui explique le rôle fondamental de la fonction ζ de *Riemann* dans l'étude des nombres premiers.

Le théorème des nombres premiers

La preuve (difficile) de ce théorème repose sur l'étude de la fonction ζ de *Riemann*. Elle est ainsi définie : $\zeta(s) = \sum_{n \geq 1} \frac{1}{n^s}$, pour $s > 1$.

En s'appuyant sur le théorème fondamental de l'arithmétique, *Euler* a démontré cette identité tout à fait remarquable :

$$\zeta(s) = \prod_{p \in \mathbb{P}} \frac{1}{1 - p^{-s}}$$

Le membre gauche de l'identité fait intervenir tous les entiers, tandis que le membre droit ne fait intervenir que les nombres premiers.

C'est l'identité d'*Euler* qui explique le rôle fondamental de la fonction ζ de *Riemann* dans l'étude des nombres premiers.

L'hypothèse de Riemann

Pour démontrer le théorème des nombres premiers, on étend la fonction ζ à tout le plan complexe $\mathbb{C}$, ce qu'a fait *Riemann* en 1858 (ce théorème est en fait équivalent à l'énoncé : $\zeta(s) \neq 0$ pour $\Re(s) = 1$).

En outre, *Riemann*, sous l'hypothèse :

$$\text{Si } \zeta(s) = 0 \text{ pour } 0 < \Re(s) < 1, \text{ alors } \Re(s) = \frac{1}{2} \quad \textit{(hypothèse de Riemann)},$$

a prouvé un résultat beaucoup plus fort que celui de *Hadamard* et *de la Vallée Poussin* :

$$\pi(x) = \mathrm{Li}(x) + \mathcal{O}(\sqrt{x}\log x) \quad \text{où } \mathrm{Li}(x) = \int_2^x \frac{dt}{\log t}$$

L'ennui est que l'*hypothèse de Riemann* resiste à toutes les tentatives de démonstration depuis plus d'un siècle et demi...

L'hypothèse de Riemann

Pour démontrer le théorème des nombres premiers, on étend la fonction ζ à tout le plan complexe $\mathbb{C}$, ce qu'a fait *Riemann* en 1858 (ce théorème est en fait équivalent à l'énoncé : $\zeta(s) \neq 0$ pour $\Re(s) = 1$).

En outre, *Riemann*, sous l'hypothèse :

$$\text{Si } \zeta(s) = 0 \text{ pour } 0 < \Re(s) < 1, \text{ alors } \Re(s) = \frac{1}{2} \quad \textit{(hypothèse de Riemann)},$$

a prouvé un résultat beaucoup plus fort que celui de *Hadamard* et *de la Vallée Poussin* :

$$\pi(x) = \mathrm{Li}(x) + \mathcal{O}(\sqrt{x}\log x) \quad \text{où } \mathrm{Li}(x) = \int_2^x \frac{dt}{\log t}$$

L'ennui est que l'*hypothèse de Riemann* resiste à toutes les tentatives de démonstration depuis plus d'un siècle et demi...

L'hypothèse de Riemann

Pour démontrer le théorème des nombres premiers, on étend la fonction ζ à tout le plan complexe $\mathbb{C}$, ce qu'a fait *Riemann* en 1858 (ce théorème est en fait équivalent à l'énoncé : $\zeta(s) \neq 0$ pour $\Re(s) = 1$).

En outre, *Riemann*, sous l'hypothèse :

$$\text{Si } \zeta(s) = 0 \text{ pour } 0 < \Re(s) < 1, \text{ alors } \Re(s) = \frac{1}{2} \quad \textit{(hypothèse de Riemann)},$$

a prouvé un résultat beaucoup plus fort que celui de *Hadamard* et *de la Vallée Poussin* :

$$\pi(x) = \mathrm{Li}(x) + \mathcal{O}(\sqrt{x}\log x) \quad \text{où } \mathrm{Li}(x) = \int_2^x \frac{dt}{\log t}$$

L'ennui est que l'*hypothèse de Riemann* resiste à toutes les tentatives de démonstration depuis plus d'un siècle et demi...

Les nombres de *Fermat*

Ce sont les nombres de la forme $F_n = 2^{2^n} + 1,\ n \geq 0$. Les cinq premiers nombres de cette suite, $F_0 = 3$, $F_1 = 5, F_2 = 17, F_3 = 257$ et $F_4 = 65537$ sont premiers. *Fermat* croyait avoir découvert une suite de nombres premiers mais *Euler* a montré que $641 \mid F_5 = 4294967297$.

On n'a pas trouvé d'autres nombres de Fermat premiers bien que l'on dispose du critère suivant dû à *Pépin* : F_n est premier si et seulement si $F_n \mid 3^{(F_n-1)/2} + 1,\ n > 0$. Mais ce critère est inutilisable en pratique en raison de la taille gigantesque des nombres de Fermat.

Théorème (*Gauss*)

Un polygone régulier à n côtés est constructible à la règle et au compas si et seulement si n est le produit d'une puissance de 2 et de nombres de Fermat premiers distincts deux à deux.

Cela n'a rien à voir avec la cryptographie mais c'est beau !

Les nombres de *Fermat*

Ce sont les nombres de la forme $F_n = 2^{2^n} + 1,\ n \geq 0$. Les cinq premiers nombres de cette suite, $F_0 = 3$, $F_1 = 5, F_2 = 17, F_3 = 257$ et $F_4 = 65537$ sont premiers. *Fermat* croyait avoir découvert une suite de nombres premiers mais *Euler* a montré que $641 \mid F_5 = 4294967297$.

On n'a pas trouvé d'autres nombres de Fermat premiers bien que l'on dispose du critère suivant dû à *Pépin* : F_n est premier si et seulement si $F_n \mid 3^{(F_n-1)/2} + 1,\ n > 0$. Mais ce critère est inutilisable en pratique en raison de la taille gigantesque des nombres de Fermat.

Théorème (*Gauss*)

Un polygone régulier à n côtés est constructible à la règle et au compas si et seulement si n est le produit d'une puissance de 2 et de nombres de Fermat premiers distincts deux à deux.

Cela n'a rien à voir avec la cryptographie mais c'est beau !

Les nombres de *Fermat*

Ce sont les nombres de la forme $F_n = 2^{2^n} + 1,\ n \geq 0$. Les cinq premiers nombres de cette suite, $F_0 = 3$, $F_1 = 5, F_2 = 17, F_3 = 257$ et $F_4 = 65537$ sont premiers. *Fermat* croyait avoir découvert une suite de nombres premiers mais *Euler* a montré que $641 \mid F_5 = 4294967297$.

On n'a pas trouvé d'autres nombres de Fermat premiers bien que l'on dispose du critère suivant dû à *Pépin* : F_n est premier si et seulement si $F_n \mid 3^{(F_n-1)/2} + 1,\ n > 0$. Mais ce critère est inutilisable en pratique en raison de la taille gigantesque des nombres de Fermat.

Théorème (*Gauss*)

Un polygone régulier à n côtés est constructible à la règle et au compas si et seulement si n est le produit d'une puissance de 2 et de nombres de Fermat premiers distincts deux à deux.

Cela n'a rien à voir avec la cryptographie mais c'est beau !

Les nombres de *Mersenne*

Les nombres de Mersenne sont les nombres de la forme $M_p = 2^p - 1$. Pour que M_p soit premier, il est nécessaire que p le soit à cause de l'identité : $a^n - b^n = (a-b)(a^{n-1} + a^{n-2}b + \cdots + ab^{n-2} + b^{n-1})$. Mais cela n'est pas suffisant : $M_{11} = 2047 = 23 \times 89$.

Théorème (*Lucas, Lehmer*)

Soit p un nombre premier impair et soit $(s_n)_{n\geq 0}$ la suite définie par : $s_0 = 4$ et $s_i = s_{i-1}^2 - 2$, $i \geq 1$. Alors M_p est premier si et seulement si $M_p \mid s_{p-2}$.

Ce test de primalité est extrêmement efficace, à tel point qu'il fournit les plus grands nombres premiers connus à ce jour :

$$M_{82589933} = 2^{82589933} - 1 \quad \text{est premier !}$$

Ce nombre de Mersenne a été découvert le 21 décembre 2018 dans le cadre du projet ▸ GIMPS et a 24 862 048 chiffres décimaux.

Les nombres de *Mersenne*

Les nombres de Mersenne sont les nombres de la forme $M_p = 2^p - 1$. Pour que M_p soit premier, il est nécessaire que p le soit à cause de l'identité : $a^n - b^n = (a-b)(a^{n-1} + a^{n-2}b + \cdots + ab^{n-2} + b^{n-1})$. Mais cela n'est pas suffisant : $M_{11} = 2047 = 23 \times 89$.

Théorème (*Lucas, Lehmer*)

Soit p un nombre premier impair et soit $(s_n)_{n\geq 0}$ la suite définie par : $s_0 = 4$ et $s_i = s_{i-1}^2 - 2$, $i \geq 1$. Alors M_p est premier si et seulement si $M_p \mid s_{p-2}$.

Ce test de primalité est extrêmement efficace, à tel point qu'il fournit les plus grands nombres premiers connus à ce jour :

$$M_{82589933} = 2^{82589933} - 1 \quad \text{est premier !}$$

Ce nombre de Mersenne a été découvert le 21 décembre 2018 dans le cadre du projet ▸ GIMPS et a 24 862 048 chiffres décimaux.

Existe-t-il des formules ?

- Depuis des siècles, on a essayé d'obtenir des formules générant des nombres premiers.
- On vu que *Fermat* pensait (à tort) avoir trouvé une telle formule avec ses nombres.
- En 1970, *Matiyasevich*, un logicien, a obtenu un résultat étonnant : il existe des polynômes à coefficients et variables entières dont toutes les valeurs positives sont des nombres premiers.
- En 1976, *Jones, Sato, Wada* et *Wiens* ont même construit explicitement un tel polynôme. C'est un polynôme à 26 variables $P(x_1, \ldots, x_{26})$.
- Malheureusement, P n'est pas utilisable en pratique pour générer des nombres premiers car il est très rare que $P(x_1, \ldots, x_{26}) > 0$.

Existe-t-il des formules ?

- Depuis des siècles, on a essayé d'obtenir des formules générant des nombres premiers.
- On vu que *Fermat* pensait (à tort) avoir trouvé une telle formule avec ses nombres.
- En 1970, *Matiyasevich*, un logicien, a obtenu un résultat étonnant : il existe des polynômes à coefficients et variables entières dont toutes les valeurs positives sont des nombres premiers.
- En 1976, *Jones, Sato, Wada* et *Wiens* ont même construit explicitement un tel polynôme. C'est un polynôme à 26 variables $P(x_1, \ldots, x_{26})$.
- Malheureusement, P n'est pas utilisable en pratique pour générer des nombres premiers car il est très rare que $P(x_1, \ldots, x_{26}) > 0$.

Existe-t-il des formules ?

- Depuis des siècles, on a essayé d'obtenir des formules générant des nombres premiers.
- On vu que *Fermat* pensait (à tort) avoir trouvé une telle formule avec ses nombres.
- En 1970, *Matiyasevich*, un logicien, a obtenu un résultat étonnant : il existe des polynômes à coefficients et variables entières dont toutes les valeurs positives sont des nombres premiers.
- En 1976, *Jones, Sato, Wada* et *Wiens* ont même construit explicitement un tel polynôme. C'est un polynôme à 26 variables $P(x_1, \ldots, x_{26})$.
- Malheureusement, P n'est pas utilisable en pratique pour générer des nombres premiers car il est très rare que $P(x_1, \ldots, x_{26}) > 0$.

Existe-t-il des formules ?

- Depuis des siècles, on a essayé d'obtenir des formules générant des nombres premiers.
- On vu que *Fermat* pensait (à tort) avoir trouvé une telle formule avec ses nombres.
- En 1970, *Matiyasevich*, un logicien, a obtenu un résultat étonnant : il existe des polynômes à coefficients et variables entières dont toutes les valeurs positives sont des nombres premiers.
- En 1976, *Jones, Sato, Wada* et *Wiens* ont même construit explicitement un tel polynôme. C'est un polynôme à 26 variables $P(x_1, \ldots, x_{26})$.
- Malheureusement, P n'est pas utilisable en pratique pour générer des nombres premiers car il est très rare que $P(x_1, \ldots, x_{26}) > 0$.

Existe-t-il des formules ?

- Depuis des siècles, on a essayé d'obtenir des formules générant des nombres premiers.
- On vu que *Fermat* pensait (à tort) avoir trouvé une telle formule avec ses nombres.
- En 1970, *Matiyasevich*, un logicien, a obtenu un résultat étonnant : il existe des polynômes à coefficients et variables entières dont toutes les valeurs positives sont des nombres premiers.
- En 1976, *Jones, Sato, Wada* et *Wiens* ont même construit explicitement un tel polynôme. C'est un polynôme à 26 variables $P(x_1, \ldots, x_{26})$.
- Malheureusement, P n'est pas utilisable en pratique pour générer des nombres premiers car il est très rare que $P(x_1, \ldots, x_{26}) > 0$.

Crible d'Ératosthène

Si $n = pq$, alors p ou q est $\leq \sqrt{n}$. Par conséquent, pour montrer que n est premier, il suffit de vérifier qu'il n'a pas de diviseurs $\leq \sqrt{n}$.

Crible d'Ératosthène

Si $n = pq$, alors p ou q est $\leq \sqrt{n}$. Par conséquent, pour montrer que n est premier, il suffit de vérifier qu'il n'a pas de diviseurs $\leq \sqrt{n}$.

Recherche des nombres premiers ≤ 100...

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Crible d'Ératosthène

Si $n = pq$, alors p ou q est $\leq \sqrt{n}$. Par conséquent, pour montrer que n est premier, il suffit de vérifier qu'il n'a pas de diviseurs $\leq \sqrt{n}$.

On efface tous les nombres pairs (sauf 2)

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Crible d'Ératosthène

Si $n = pq$, alors p ou q est $\leq \sqrt{n}$. Par conséquent, pour montrer que n est premier, il suffit de vérifier qu'il n'a pas de diviseurs $\leq \sqrt{n}$.

On efface tous les multiples de 3 (sauf 3)

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Crible d'Ératosthène

Si $n = pq$, alors p ou q est $\leq \sqrt{n}$. Par conséquent, pour montrer que n est premier, il suffit de vérifier qu'il n'a pas de diviseurs $\leq \sqrt{n}$.

On efface tous les multiples de 5 (sauf 5)

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Crible d'Ératosthène

Si $n = pq$, alors p ou q est $\leq \sqrt{n}$. Par conséquent, pour montrer que n est premier, il suffit de vérifier qu'il n'a pas de diviseurs $\leq \sqrt{n}$.

On efface tous les multiples de 7 (sauf 7)

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Crible d'Ératosthène

Si $n = pq$, alors p ou q est $\leq \sqrt{n}$. Par conséquent, pour montrer que n est premier, il suffit de vérifier qu'il n'a pas de diviseurs $\leq \sqrt{n}$.

On a terminé car $11 > \sqrt{100} = 10$

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Crible d'Ératosthène

Si $n = pq$, alors p ou q est $\leq \sqrt{n}$. Par conséquent, pour montrer que n est premier, il suffit de vérifier qu'il n'a pas de diviseurs $\leq \sqrt{n}$.

Malheureusement cette techique est, elle aussi, inutilisable dès que n est suffisamment grand.

Il va donc falloir restreindre quelque peu nos ambitions...

Théorie des cribles

La théorie des cribles est une partie très active de la théorie des nombres. Elle a été initiée par le mathématicien norvégien *Viggo Brun* auquel on doit le résultat (assez inattendu) suivant :

Théorème (*Brun – 1919*)

La série des inverses des nombres premiers jumeaux converge :

$$\sum_{p,p+2\in\mathbb{P}} \left(\frac{1}{p}+\frac{1}{p+2}\right) = \left(\frac{1}{3}+\frac{1}{5}\right)+\left(\frac{1}{5}+\frac{1}{7}\right)+\cdots < +\infty$$

La somme de cette série est appelée **constante de Brun** *et est noté* B_2*. Sa valeur approchée est :*

$$B_2 \simeq 1,902160583104 \quad \textit{(Sebah, Demichel – 2002)}$$

Ainsi, la question de savoir s'il existe une infinité de nombres premiers jumeaux reste ouverte (on conjecture que c'est le cas).

Théorie des cribles

La théorie des cribles est une partie très active de la théorie des nombres. Elle a été initiée par le mathématicien norvégien *Viggo Brun* auquel on doit le résultat (assez inattendu) suivant :

Théorème (*Brun – 1919*)

La série des inverses des nombres premiers jumeaux converge :

$$\sum_{p,p+2\in\mathbb{P}} \left(\frac{1}{p}+\frac{1}{p+2}\right) = \left(\frac{1}{3}+\frac{1}{5}\right)+\left(\frac{1}{5}+\frac{1}{7}\right)+\cdots < +\infty$$

La somme de cette série est appelée **constante de Brun** *et est noté* B_2*. Sa valeur approchée est :*

$$B_2 \simeq 1,902160583104 \quad \textit{(Sebah, Demichel – 2002)}$$

Ainsi, la question de savoir s'il existe une infinité de nombres premiers jumeaux reste ouverte (on conjecture que c'est le cas).

Théorie des cribles

La théorie des cribles est une partie très active de la théorie des nombres. Elle a été initiée par le mathématicien norvégien *Viggo Brun* auquel on doit le résultat (assez inattendu) suivant :

Théorème (*Brun – 1919*)

La série des inverses des nombres premiers jumeaux converge :

$$\sum_{p,p+2\in\mathbb{P}} \left(\frac{1}{p}+\frac{1}{p+2}\right) = \left(\frac{1}{3}+\frac{1}{5}\right)+\left(\frac{1}{5}+\frac{1}{7}\right)+\cdots < +\infty$$

La somme de cette série est appelée **constante de Brun** *et est noté* B_2*. Sa valeur approchée est :*

$$B_2 \simeq 1,902160583104 \quad \textit{(Sebah, Demichel – 2002)}$$

Ainsi, la question de savoir s'il existe une infinité de nombres premiers jumeaux reste ouverte (on conjecture que c'est le cas).

Petit théorème de *Fermat*

Théorème (petit théorème de *Fermat*)

Soit p un nombre premier. Alors pour tout entier a premier avec p, on a :

$$a^{p-1} \equiv 1 \ (\mod p)$$

Attention : la réciproque de ce théorème est fausse. Il existe des nombres composés N tels que $a^{N-1} \equiv 1 \pmod N$ pour tout a premier avec N.

Ces nombres composés sont appelés nombres de *Carmichael*. Il y en a une infinité et le plus petit d'entre eux est : $561 = 3 \times 11 \times 17$.

L'existence de ces nombres enlève en général tout intérêt pratique à l'utilisation du petit théorème de *Fermat* comme test de primalité.

Petit théorème de *Fermat*

Théorème (petit théorème de *Fermat*)

Soit p un nombre premier. Alors pour tout entier a premier avec p, on a :

$$a^{p-1} \equiv 1 \ (\mod p)$$

Attention : la réciproque de ce théorème est fausse. Il existe des nombres composés N tels que $a^{N-1} \equiv 1 \pmod N$ pour tout a premier avec N.

Ces nombres composés sont appelés nombres de *Carmichael*. Il y en a une infinité et le plus petit d'entre eux est : $561 = 3 \times 11 \times 17$.

L'existence de ces nombres enlève en général tout intérêt pratique à l'utilisation du petit théorème de *Fermat* comme test de primalité.

Petit théorème de *Fermat*

Théorème (petit théorème de *Fermat*)

Soit p un nombre premier. Alors pour tout entier a premier avec p, on a :

$$a^{p-1} \equiv 1 \ (\mod p)$$

Attention : la réciproque de ce théorème est fausse. Il existe des nombres composés N tels que $a^{N-1} \equiv 1 \pmod N$ pour tout a premier avec N.

Ces nombres composés sont appelés nombres de *Carmichael*. Il y en a une infinité et le plus petit d'entre eux est : $561 = 3 \times 11 \times 17$.

L'existence de ces nombres enlève en général tout intérêt pratique à l'utilisation du petit théorème de *Fermat* comme test de primalité.

Petit théorème de *Fermat*

Théorème (petit théorème de *Fermat*)

Soit p un nombre premier. Alors pour tout entier a premier avec p, on a :

$$a^{p-1} \equiv 1 \ (\mod p)$$

Attention : la réciproque de ce théorème est fausse. Il existe des nombres composés N tels que $a^{N-1} \equiv 1 \pmod N$ pour tout a premier avec N.

Ces nombres composés sont appelés nombres de *Carmichael*. Il y en a une infinité et le plus petit d'entre eux est : $561 = 3 \times 11 \times 17$.

L'existence de ces nombres enlève en général tout intérêt pratique à l'utilisation du petit théorème de *Fermat* comme test de primalité.

Critère de *Rabin-Miller*

Théorème (Critère de *Rabin-Miller*)

Soit N un nombre premier impair et a un entier premier avec N. Si $N = 2^s m + 1$ avec m impair, alors :

(1) *soit* $a^m \equiv 1 \pmod{N}$,

(2) *soit il existe* $i \in \{0, 1, \ldots, s-1\}$ *tel que* $a^{2^i m} \equiv -1 \pmod{N}$.

Si un entier $a \in \{1, \ldots, N-1\}$ vérifie l'une des propriétés (1) ou (2), on dit que N est **pseudo-premier** en base a. On a la proposition suivante :

Proposition

Si N est composé, alors il est pseudo-premier dans au plus $1/4$ des bases a.

Donc, si on fait k essais réussis avec des bases a choisies aléatoirement dans l'intervalle $[2, N]$, la probabilité P_N qu'un nombre N de b bits soit composé est $\simeq b \log 2/(2 \times 4^k)$. Par exemple, pour $b = 64, k = 25$, $P_N \simeq 2 \times 10^{-14}$, ce qui est une probabilité très faible.

Critère de *Rabin-Miller*

Théorème (Critère de *Rabin-Miller*)

Soit N un nombre premier impair et a un entier premier avec N. Si $N = 2^s m + 1$ avec m impair, alors :

(1) *soit* $a^m \equiv 1 \pmod{N}$,

(2) *soit il existe* $i \in \{0, 1, \ldots, s-1\}$ *tel que* $a^{2^i m} \equiv -1 \pmod{N}$.

Si un entier $a \in \{1, \ldots, N-1\}$ vérifie l'une des propriétés (1) ou (2), on dit que N est **pseudo-premier** en base a. On a la proposition suivante :

Proposition

Si N est composé, alors il est pseudo-premier dans au plus $1/4$ des bases a.

Donc, si on fait k essais réussis avec des bases a choisies aléatoirement dans l'intervalle $[2, N]$, la probabilité P_N qu'un nombre N de b bits soit composé est $\simeq b \log 2/(2 \times 4^k)$. Par exemple, pour $b = 64, k = 25$, $P_N \simeq 2 \times 10^{-14}$, ce qui est une probabilité très faible.

Critère de *Rabin-Miller*

- Le résultat précédent peut être sensiblement amélioré. Si $P_{b,k}$ désigne la probabilité qu'un nombre impair composé de p bits passe avec succès k tests successifs de *Rabin-Miller*, alors $P_{b,k} \leq 4^{-k}$. Pour $k = 25$, cette probabilité est de l'ordre de 10^{-15}.
- On a même (*Damgård, Landrock et Pomerance*) pour tout $b \geq 2$ et pour $k = 1$ (**un seul test**) : $P_{b,1} < b^2 4^{2-b^{1/2}}$. Pour $b = 2048$, on a $P_{2048,1} < 4 \times 10^{-20}$.
- Le critère de *Rabin-Miller* est donc particulièrement efficace. Il est largement utilisé en cryptographie pratique.
- A noter qu'il existe des algorithmes **déterministes** en temps **polynomial** permettant de prouver la primalité d'un nombre. L'un des plus connus, découvert en 2002, est le critère AKS (*Agrawal, Kayal et Saxena*). Sa complexité en temps est en $\mathcal{O}((\log N)^{12})$.

Critère de *Rabin-Miller*

- Le résultat précédent peut être sensiblement amélioré. Si $P_{b,k}$ désigne la probabilité qu'un nombre impair composé de p bits passe avec succès k tests successifs de *Rabin-Miller*, alors $P_{b,k} \leq 4^{-k}$. Pour $k = 25$, cette probabilité est de l'ordre de 10^{-15}.
- On a même (*Damgård, Landrock et Pomerance*) pour tout $b \geq 2$ et pour $k = 1$ (**un seul test**) : $P_{b,1} < b^2 4^{2-b^{1/2}}$. Pour $b = 2048$, on a $P_{2048,1} < 4 \times 10^{-20}$.
- Le critère de *Rabin-Miller* est donc particulièrement efficace. Il est largement utilisé en cryptographie pratique.
- A noter qu'il existe des algorithmes **déterministes** en temps **polynomial** permettant de prouver la primalité d'un nombre. L'un des plus connus, découvert en 2002, est le critère AKS (*Agrawal, Kayal et Saxena*). Sa complexité en temps est en $\mathcal{O}((\log N)^{12})$.

Critère de *Rabin-Miller*

- Le résultat précédent peut être sensiblement amélioré. Si $P_{b,k}$ désigne la probabilité qu'un nombre impair composé de p bits passe avec succès k tests successifs de *Rabin-Miller*, alors $P_{b,k} \leq 4^{-k}$. Pour $k = 25$, cette probabilité est de l'ordre de 10^{-15}.
- On a même (*Damgård, Landrock et Pomerance*) pour tout $b \geq 2$ et pour $k = 1$ (**un seul test**) : $P_{b,1} < b^2 4^{2-b^{1/2}}$. Pour $b = 2048$, on a $P_{2048,1} < 4 \times 10^{-20}$.
- Le critère de *Rabin-Miller* est donc particulièrement efficace. Il est largement utilisé en cryptographie pratique.
- A noter qu'il existe des algorithmes **déterministes** en temps **polynomial** permettant de prouver la primalité d'un nombre. L'un des plus connus, découvert en 2002, est le critère AKS (*Agrawal, Kayal et Saxena*). Sa complexité en temps est en $\mathcal{O}((\log N)^{12})$.

Critère de *Rabin-Miller*

- Le résultat précédent peut être sensiblement amélioré. Si $P_{b,k}$ désigne la probabilité qu'un nombre impair composé de p bits passe avec succès k tests successifs de *Rabin-Miller*, alors $P_{b,k} \leq 4^{-k}$. Pour $k = 25$, cette probabilité est de l'ordre de 10^{-15}.
- On a même (*Damgård, Landrock et Pomerance*) pour tout $b \geq 2$ et pour $k = 1$ (**un seul test**) : $P_{b,1} < b^2 4^{2-b^{1/2}}$. Pour $b = 2048$, on a $P_{2048,1} < 4 \times 10^{-20}$.
- Le critère de *Rabin-Miller* est donc particulièrement efficace. Il est largement utilisé en cryptographie pratique.
- A noter qu'il existe des algorithmes **déterministes** en temps **polynomial** permettant de prouver la primalité d'un nombre. L'un des plus connus, découvert en 2002, est le critère AKS (*Agrawal, Kayal et Saxena*). Sa complexité en temps est en $\mathcal{O}((\log N)^{12})$.

Une implémentation de *Rabin-Miller* en Python

```
# Teste si N est pseudo-premier en base a. Les arguments
# s et m sont tels que N = 2^s m + 1. Retourne True si N
# est (certainement) non premier, False sinon.

def is_composite(a, s, m, N):
    if pow(a, m, N) == 1: # a^m ≡ 1 (mod N)
        return False
    for i in range(s): # i ∈ {0, 1, ..., s - 1}
        if pow(a, 2**i * m, N) == N - 1: # a^(2^i m) ≡ -1 (mod N)
            return False
    return True
```

Une implémentation de *Rabin-Miller* en Python

```
def rabin_miller(N, k): # k est le nombre de tests
    if N == 2:
        return True
    if not N & 1: # N est pair
        return False
    s, m = 0, N - 1
    while True: # On calcule s et m tels que N = 2^s m + 1
        q, r = divmod(m, 2)
        if r == 1:
            break
        s += 1
        m = q
    for i in range(k): # On fait k tests
        a = random.randrange(2, N) # a choisi au hasard
        if is_composite(a, s, m, N):
            return False
    return True
```

Une vue d'ensemble

- Le système RSA, du nom de ses concepteurs *Rivest*, *Shamir* et *Adleman* est le premier système de chiffrement à clé publique robuste à avoir été inventé (en 1977).
- Il est toujours largement utilisé lorque l'on veut échanger des données de manière sécurisée sur Internet.
- Le chiffrement RSA est **asymétrique** : il utilise une paire de clés, une clé *publique* pour le chiffrement et une clé *privée* pour le déchiffrement.
- Le système RSA permet également de **signer** des données. Celles-ci sont signées avec la clé privée et tout possesseur de la clé publique peut vérifier la signature.
- La robustesse du système RSA repose sur le fait que l'on ne sait pas, avec les moyens et savoirs actuels, obtenir la clé privée à partir de la simple connaissance de la clé publique.

Une vue d'ensemble

- Le système RSA, du nom de ses concepteurs *Rivest*, *Shamir* et *Adleman* est le premier système de chiffrement à clé publique robuste à avoir été inventé (en 1977).
- Il est toujours largement utilisé lorque l'on veut échanger des données de manière sécurisée sur Internet.
- Le chiffrement RSA est **asymétrique** : il utilise une paire de clés, une clé *publique* pour le chiffrement et une clé *privée* pour le déchiffrement.
- Le système RSA permet également de **signer** des données. Celles-ci sont signées avec la clé privée et tout possesseur de la clé publique peut vérifier la signature.
- La robustesse du système RSA repose sur le fait que l'on ne sait pas, avec les moyens et savoirs actuels, obtenir la clé privée à partir de la simple connaissance de la clé publique.

Une vue d'ensemble

- Le système RSA, du nom de ses concepteurs *Rivest*, *Shamir* et *Adleman* est le premier système de chiffrement à clé publique robuste à avoir été inventé (en 1977).
- Il est toujours largement utilisé lorque l'on veut échanger des données de manière sécurisée sur Internet.
- Le chiffrement RSA est **asymétrique** : il utilise une paire de clés, une clé *publique* pour le chiffrement et une clé *privée* pour le déchiffrement.
- Le système RSA permet également de **signer** des données. Celles-ci sont signées avec la clé privée et tout possesseur de la clé publique peut vérifier la signature.
- La robustesse du système RSA repose sur le fait que l'on ne sait pas, avec les moyens et savoirs actuels, obtenir la clé privée à partir de la simple connaissance de la clé publique.

Une vue d'ensemble

- Le système RSA, du nom de ses concepteurs *Rivest*, *Shamir* et *Adleman* est le premier système de chiffrement à clé publique robuste à avoir été inventé (en 1977).
- Il est toujours largement utilisé lorque l'on veut échanger des données de manière sécurisée sur Internet.
- Le chiffrement RSA est **asymétrique** : il utilise une paire de clés, une clé *publique* pour le chiffrement et une clé *privée* pour le déchiffrement.
- Le système RSA permet également de **signer** des données. Celles-ci sont signées avec la clé privée et tout possesseur de la clé publique peut vérifier la signature.
- La robustesse du système RSA repose sur le fait que l'on ne sait pas, avec les moyens et savoirs actuels, obtenir la clé privée à partir de la simple connaissance de la clé publique.

Une vue d'ensemble

- Le système RSA, du nom de ses concepteurs *Rivest*, *Shamir* et *Adleman* est le premier système de chiffrement à clé publique robuste à avoir été inventé (en 1977).
- Il est toujours largement utilisé lorque l'on veut échanger des données de manière sécurisée sur Internet.
- Le chiffrement RSA est **asymétrique** : il utilise une paire de clés, une clé *publique* pour le chiffrement et une clé *privée* pour le déchiffrement.
- Le système RSA permet également de **signer** des données. Celles-ci sont signées avec la clé privée et tout possesseur de la clé publique peut vérifier la signature.
- La robustesse du système RSA repose sur le fait que l'on ne sait pas, avec les moyens et savoirs actuels, obtenir la clé privée à partir de la simple connaissance de la clé publique.

Utilisation de RSA en pratique

- Comme tout système à clé publique, le chiffrement RSA est **coûteux** en temps calcul. Plus précisément, il est (beaucoup) plus coûteux qu'un chiffrement *symétrique* de robustesse équivalente comme AES par exemple.
- C'est pour cela, que souvent en pratique, on ne l'utilise que pour transmettre la clé *secrète* d'un chiffrement symétrique.
- Concrètement supposons qu'*Alice* veuille échanger des données de manière sécurisée avec *Bob*. Elle va ainsi procéder :
  - Elle va tout d'abord générer une clé secrète pour le chiffrement symétrique dont le choix (public) a été fait au préalable avec son correspondant.
  - Elle chiffre cette clé secrète avec la clé publique (RSA) de *Bob* et transmet la clé ansi chiffrée à ce dernier.
  - *Bob* déchiffre la clé secrète avec sa clé privée (RSA).
  - Les deux correspondants étant maintenant en possession de la clé secrète, le chiffrement symétrique des données peut commencer.

Utilisation de RSA en pratique

- Comme tout système à clé publique, le chiffrement RSA est **coûteux** en temps calcul. Plus précisément, il est (beaucoup) plus coûteux qu'un chiffrement *symétrique* de robustesse équivalente comme AES par exemple.
- C'est pour cela, que souvent en pratique, on ne l'utilise que pour transmettre la clé *secrète* d'un chiffrement symétrique.
- Concrètement supposons qu'*Alice* veuille échanger des données de manière sécurisée avec *Bob*. Elle va ainsi procéder :
  - Elle va tout d'abord générer une clé secrète pour le chiffrement symétrique dont le choix (public) a été fait au préalable avec son correspondant.
  - Elle chiffre cette clé secrète avec la clé publique (RSA) de *Bob* et transmet la clé ansi chiffrée à ce dernier.
  - *Bob* déchiffre la clé secrète avec sa clé privée (RSA).
  - Les deux correspondants étant maintenant en possession de la clé secrète, le chiffrement symétrique des données peut commencer.

Utilisation de RSA en pratique

- Comme tout système à clé publique, le chiffrement RSA est **coûteux** en temps calcul. Plus précisément, il est (beaucoup) plus coûteux qu'un chiffrement *symétrique* de robustesse équivalente comme AES par exemple.
- C'est pour cela, que souvent en pratique, on ne l'utilise que pour transmettre la clé *secrète* d'un chiffrement symétrique.
- Concrètement supposons qu'*Alice* veuille échanger des données de manière sécurisée avec *Bob*. Elle va ainsi procéder :
  - Elle va tout d'abord générer une clé secrète pour le chiffrement symétrique dont le choix (public) a été fait au préalable avec son correspondant.
  - Elle chiffre cette clé secrète avec la clé publique (RSA) de *Bob* et transmet la clé ansi chiffrée à ce dernier.
  - *Bob* déchiffre la clé secrète avec sa clé privée (RSA).
  - Les deux correspondants étant maintenant en possession de la clé secrète, le chiffrement symétrique des données peut commencer.

Utilisation de RSA en pratique

- Comme tout système à clé publique, le chiffrement RSA est **coûteux** en temps calcul. Plus précisément, il est (beaucoup) plus coûteux qu'un chiffrement *symétrique* de robustesse équivalente comme AES par exemple.
- C'est pour cela, que souvent en pratique, on ne l'utilise que pour transmettre la clé *secrète* d'un chiffrement symétrique.
- Concrètement supposons qu'*Alice* veuille échanger des données de manière sécurisée avec *Bob*. Elle va ainsi procéder :
  - Elle va tout d'abord générer une clé secrète pour le chiffrement symétrique dont le choix (public) a été fait au préalable avec son correspondant.
  - Elle chiffre cette clé secrète avec la clé publique (RSA) de *Bob* et transmet la clé ansi chiffrée à ce dernier.
  - *Bob* déchiffre la clé secrète avec sa clé privée (RSA).
  - Les deux correspondants étant maintenant en possession de la clé secrète, le chiffrement symétrique des données peut commencer.

Utilisation de RSA en pratique

- Comme tout système à clé publique, le chiffrement RSA est **coûteux** en temps calcul. Plus précisément, il est (beaucoup) plus coûteux qu'un chiffrement *symétrique* de robustesse équivalente comme AES par exemple.
- C'est pour cela, que souvent en pratique, on ne l'utilise que pour transmettre la clé *secrète* d'un chiffrement symétrique.
- Concrètement supposons qu'*Alice* veuille échanger des données de manière sécurisée avec *Bob*. Elle va ainsi procéder :
  - Elle va tout d'abord générer une clé secrète pour le chiffrement symétrique dont le choix (public) a été fait au préalable avec son correspondant.
  - Elle chiffre cette clé secrète avec la clé publique (RSA) de *Bob* et transmet la clé ansi chiffrée à ce dernier.
  - *Bob* déchiffre la clé secrète avec sa clé privée (RSA).
  - Les deux correspondants étant maintenant en possession de la clé secrète, le chiffrement symétrique des données peut commencer.

Utilisation de RSA en pratique

- Comme tout système à clé publique, le chiffrement RSA est **coûteux** en temps calcul. Plus précisément, il est (beaucoup) plus coûteux qu'un chiffrement *symétrique* de robustesse équivalente comme AES par exemple.
- C'est pour cela, que souvent en pratique, on ne l'utilise que pour transmettre la clé *secrète* d'un chiffrement symétrique.
- Concrètement supposons qu'*Alice* veuille échanger des données de manière sécurisée avec *Bob*. Elle va ainsi procéder :
  - Elle va tout d'abord générer une clé secrète pour le chiffrement symétrique dont le choix (public) a été fait au préalable avec son correspondant.
  - Elle chiffre cette clé secrète avec la clé publique (RSA) de *Bob* et transmet la clé ansi chiffrée à ce dernier.
  - *Bob* déchiffre la clé secrète avec sa clé privée (RSA).
  - Les deux correspondants étant maintenant en possession de la clé secrète, le chiffrement symétrique des données peut commencer.

Utilisation de RSA en pratique

- Comme tout système à clé publique, le chiffrement RSA est **coûteux** en temps calcul. Plus précisément, il est (beaucoup) plus coûteux qu'un chiffrement *symétrique* de robustesse équivalente comme AES par exemple.
- C'est pour cela, que souvent en pratique, on ne l'utilise que pour transmettre la clé *secrète* d'un chiffrement symétrique.
- Concrètement supposons qu'*Alice* veuille échanger des données de manière sécurisée avec *Bob*. Elle va ainsi procéder :
  - Elle va tout d'abord générer une clé secrète pour le chiffrement symétrique dont le choix (public) a été fait au préalable avec son correspondant.
  - Elle chiffre cette clé secrète avec la clé publique (RSA) de *Bob* et transmet la clé ansi chiffrée à ce dernier.
  - *Bob* déchiffre la clé secrète avec sa clé privée (RSA).
  - Les deux correspondants étant maintenant en possession de la clé secrète, le chiffrement symétrique des données peut commencer.

Indicatrice d'*Euler*

Avant d'aborder l'architecture de RSA, nous aurons encore besoin d'un outil mathématique : l'indicatrice d'*Euler*.

Définition (indicatrice d'*Euler*)

L'indicatrice d'Euler est la fonction $\varphi : \mathbb{N}^* \to \mathbb{N}^*$ *définie par :*

$$\varphi(n) = \text{card}(\{m \in \mathbb{N}^* \mid m \leq n \text{ et } \text{pgcd}(m, n) = 1\}).$$

Par exemple, on a : $\varphi(1) = 1$, $\varphi(2) = 1$, $\varphi(10) = 4$ et, pour tout $p \in \mathbb{P}$, $\varphi(p) = p - 1$.

Proposition

L'indicatrice d'Euler est une fonction multiplicative : si $m, n \in \mathbb{N}^*$ *sont premiers entre eux, alors* $\varphi(mn) = \varphi(m)\varphi(n)$.

Par conséquent si $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$, on a $\varphi(n) = n\,(1 - \frac{1}{p_1}) \cdots (1 - \frac{1}{p_k})$.

Indicatrice d'*Euler*

Avant d'aborder l'architecture de RSA, nous aurons encore besoin d'un outil mathématique : l'indicatrice d'*Euler*.

Définition (indicatrice d'*Euler*)

L'indicatrice d'Euler est la fonction $\varphi : \mathbb{N}^* \to \mathbb{N}^*$ *définie par :*

$$\varphi(n) = \text{card}(\{m \in \mathbb{N}^* \mid m \leq n \text{ et } \text{pgcd}(m, n) = 1\}).$$

Par exemple, on a : $\varphi(1) = 1$, $\varphi(2) = 1$, $\varphi(10) = 4$ et, pour tout $p \in \mathbb{P}$, $\varphi(p) = p - 1$.

Proposition

L'indicatrice d'Euler est une fonction multiplicative : si $m, n \in \mathbb{N}^*$ *sont premiers entre eux, alors* $\varphi(mn) = \varphi(m)\varphi(n)$.

Par conséquent si $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$, on a $\varphi(n) = n\,(1 - \frac{1}{p_1}) \cdots (1 - \frac{1}{p_k})$.

Indicatrice d'*Euler*

Avant d'aborder l'architecture de RSA, nous aurons encore besoin d'un outil mathématique : l'indicatrice d'*Euler*.

Définition (indicatrice d'*Euler*)

L'indicatrice d'Euler est la fonction $\varphi : \mathbb{N}^* \to \mathbb{N}^*$ *définie par :*

$$\varphi(n) = \text{card}(\{m \in \mathbb{N}^* \mid m \leq n \text{ et } \text{pgcd}(m, n) = 1\}).$$

Par exemple, on a : $\varphi(1) = 1$, $\varphi(2) = 1$, $\varphi(10) = 4$ et, pour tout $p \in \mathbb{P}$, $\varphi(p) = p - 1$.

Proposition

L'indicatrice d'Euler est une fonction multiplicative : si $m, n \in \mathbb{N}^*$ *sont premiers entre eux, alors* $\varphi(mn) = \varphi(m)\varphi(n)$.

Par conséquent si $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$, on a $\varphi(n) = n\left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_k}\right)$.

Indicatrice d'*Euler*

Avant d'aborder l'architecture de RSA, nous aurons encore besoin d'un outil mathématique : l'indicatrice d'*Euler*.

Définition (indicatrice d'*Euler*)

L'indicatrice d'Euler est la fonction $\varphi : \mathbb{N}^* \to \mathbb{N}^*$ *définie par :*

$$\varphi(n) = \text{card}(\{m \in \mathbb{N}^* \mid m \leq n \text{ et } \text{pgcd}(m, n) = 1\}).$$

Par exemple, on a : $\varphi(1) = 1$, $\varphi(2) = 1$, $\varphi(10) = 4$ et, pour tout $p \in \mathbb{P}$, $\varphi(p) = p - 1$.

Proposition

L'indicatrice d'Euler est une fonction multiplicative : si $m, n \in \mathbb{N}^*$ *sont premiers entre eux, alors* $\varphi(mn) = \varphi(m)\varphi(n)$.

Par conséquent si $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$, on a $\varphi(n) = n\,(1 - \frac{1}{p_1}) \cdots (1 - \frac{1}{p_k})$.

Indicatrice d'*Euler*

Une généralisation du petit théorème de *Fermat* due à *Euler* :

Théorème (*Euler-Fermat*)

Soient $a, n \in \mathbb{N}^*$ tels que a et n soient premiers entre eux. Alors on a :

$$a^{\varphi(n)} \equiv 1 \ (\mod n).$$

Une application amusante : quel est le chiffre des unités de 2017^{2016} ?

Comme $\varphi(10) = 4$ et $\text{pgcd}(10, 2017) = 1$, on a :

$$2017^{2016} = 2017^{4\times 504} = (2017^4)^{504} = (2017^{\varphi(10)})^{504} \equiv 1 \ (\mod 10),$$

le chiffre des unités de 2017^{2016} est donc 1.

Une identité remarquable (toujours due à *Euler*) :

$$\sum_{d|n} \varphi(d) = n$$

Indicatrice d'*Euler*

Une généralisation du petit théorème de *Fermat* due à *Euler* :

Théorème (*Euler-Fermat*)

Soient $a, n \in \mathbb{N}^*$ tels que a et n soient premiers entre eux. Alors on a :

$$a^{\varphi(n)} \equiv 1 \ (\mod n).$$

Une application amusante : quel est le chiffre des unités de 2017^{2016} ?

Comme $\varphi(10) = 4$ et $\text{pgcd}(10, 2017) = 1$, on a :

$$2017^{2016} = 2017^{4\times 504} = (2017^4)^{504} = (2017^{\varphi(10)})^{504} \equiv 1 \ (\mod 10),$$

le chiffre des unités de 2017^{2016} est donc 1.

Une identité remarquable (toujours due à *Euler*) :

$$\sum_{d|n} \varphi(d) = n$$

Indicatrice d'*Euler*

Une généralisation du petit théorème de *Fermat* due à *Euler* :

Théorème (*Euler-Fermat*)

Soient $a, n \in \mathbb{N}^*$ tels que a et n soient premiers entre eux. Alors on a :

$$a^{\varphi(n)} \equiv 1 \ (\mod n).$$

Une application amusante : quel est le chiffre des unités de 2017^{2016} ?

Comme $\varphi(10) = 4$ et $\text{pgcd}(10, 2017) = 1$, on a :

$$2017^{2016} = 2017^{4\times 504} = (2017^4)^{504} = (2017^{\varphi(10)})^{504} \equiv 1 \ (\mod 10),$$

le chiffre des unités de 2017^{2016} est donc 1.

Une identité remarquable (toujours due à *Euler*) :

$$\sum_{d|n} \varphi(d) = n$$

Indicatrice d'*Euler*

Une généralisation du petit théorème de *Fermat* due à *Euler* :

Théorème (*Euler-Fermat*)

Soient $a, n \in \mathbb{N}^*$ tels que a et n soient premiers entre eux. Alors on a :

$$a^{\varphi(n)} \equiv 1 \ (\mod n).$$

Une application amusante : quel est le chiffre des unités de 2017^{2016} ?

Comme $\varphi(10) = 4$ et $\text{pgcd}(10, 2017) = 1$, on a :

$$2017^{2016} = 2017^{4\times 504} = (2017^4)^{504} = (2017^{\varphi(10)})^{504} \equiv 1 \ (\mod 10),$$

le chiffre des unités de 2017^{2016} est donc 1.

Une identité remarquable (toujours due à *Euler*) :

$$\sum_{d|n} \varphi(d) = n$$

Le principe

- La **clé privée** est constituée de deux "grands" nombres premiers p et q.
- La **clé publique** est constituée du produit $n = pq$ et d'un entier e inversible modulo $\varphi(n)$, c'est à dire un entier e tel qu'il existe un entier d avec $ed \equiv 1 \pmod{\varphi(n)}$.
- Le **chiffré d'un message** représenté par un entier M modulo n est $M^e \pmod{n}$.
- Pour **déchiffer le message** chiffré C, il suffit de calculer $C^d \pmod{n}$ où d est l'inverse de e modulo $\varphi(n)$.
- Justification de l'algorithme :
  - Supposons que M soit premier avec n. Alors, d'après le théorème d'*Euler-Fermat*, on a : $M^{\varphi(n)} \equiv 1 \pmod{n}$. Par conséquent : $C^d = (M^e)^d = M^{ed} = M^{1+k\varphi(n)} = M \times (M^{\varphi(n)})^k \equiv M \pmod{n}$
  - Lorsque M n'est pas premier avec n, trois cas se présentent : $p \mid M$ et $q \nmid M$; $p \nmid M$ et $q \mid M$; $n \mid M$. Le dernier cas est trivial tandis que les deux premiers nécessitent l'utilisation du *théorème chinois* en plus du théorème d'*Euler-Fermat* pour conclure.

Le principe

- La **clé privée** est constituée de deux "grands" nombres premiers p et q.
- La **clé publique** est constituée du produit $n = pq$ et d'un entier e inversible modulo $\varphi(n)$, c'est à dire un entier e tel qu'il existe un entier d avec $ed \equiv 1 \pmod{\varphi(n)}$.
- Le **chiffré d'un message** représenté par un entier M modulo n est $M^e \pmod{n}$.
- Pour **déchiffer le message** chiffré C, il suffit de calculer $C^d \pmod{n}$ où d est l'inverse de e modulo $\varphi(n)$.
- Justification de l'algorithme :
  - Supposons que M soit premier avec n. Alors, d'après le théorème d'*Euler-Fermat*, on a : $M^{\varphi(n)} \equiv 1 \pmod{n}$. Par conséquent : $C^d = (M^e)^d = M^{ed} = M^{1+k\varphi(n)} = M \times (M^{\varphi(n)})^k \equiv M \pmod{n}$
  - Lorsque M n'est pas premier avec n, trois cas se présentent : $p \mid M$ et $q \nmid M$; $p \nmid M$ et $q \mid M$; $n \mid M$. Le dernier cas est trivial tandis que les deux premiers nécessitent l'utilisation du *théorème chinois* en plus du théorème d'*Euler-Fermat* pour conclure.

Le principe

- La **clé privée** est constituée de deux "grands" nombres premiers p et q.
- La **clé publique** est constituée du produit $n = pq$ et d'un entier e inversible modulo $\varphi(n)$, c'est à dire un entier e tel qu'il existe un entier d avec $ed \equiv 1 \pmod{\varphi(n)}$.
- Le **chiffré d'un message** représenté par un entier M modulo n est $M^e \pmod{n}$.
- Pour **déchiffer le message** chiffré C, il suffit de calculer $C^d \pmod{n}$ où d est l'inverse de e modulo $\varphi(n)$.
- Justification de l'algorithme :
  - Supposons que M soit premier avec n. Alors, d'après le théorème d'*Euler-Fermat*, on a : $M^{\varphi(n)} \equiv 1 \pmod{n}$. Par conséquent : $C^d = (M^e)^d = M^{ed} = M^{1+k\varphi(n)} = M \times (M^{\varphi(n)})^k \equiv M \pmod{n}$
  - Lorsque M n'est pas premier avec n, trois cas se présentent : $p \mid M$ et $q \nmid M$; $p \nmid M$ et $q \mid M$; $n \mid M$. Le dernier cas est trivial tandis que les deux premiers nécessitent l'utilisation du *théorème chinois* en plus du théorème d'*Euler-Fermat* pour conclure.

Le principe

- La **clé privée** est constituée de deux "grands" nombres premiers p et q.
- La **clé publique** est constituée du produit $n = pq$ et d'un entier e inversible modulo $\varphi(n)$, c'est à dire un entier e tel qu'il existe un entier d avec $ed \equiv 1 \pmod{\varphi(n)}$.
- Le **chiffré d'un message** représenté par un entier M modulo n est $M^e \pmod{n}$.
- Pour **déchiffer le message** chiffré C, il suffit de calculer $C^d \pmod{n}$ où d est l'inverse de e modulo $\varphi(n)$.
- Justification de l'algorithme :
  - Supposons que M soit premier avec n. Alors, d'après le théorème d'*Euler-Fermat*, on a : $M^{\varphi(n)} \equiv 1 \pmod{n}$. Par conséquent : $C^d = (M^e)^d = M^{ed} = M^{1+k\varphi(n)} = M \times (M^{\varphi(n)})^k \equiv M \pmod{n}$
  - Lorsque M n'est pas premier avec n, trois cas se présentent : $p \mid M$ et $q \nmid M$; $p \nmid M$ et $q \mid M$; $n \mid M$. Le dernier cas est trivial tandis que les deux premiers nécessitent l'utilisation du *théorème chinois* en plus du théorème d'*Euler-Fermat* pour conclure.

Le principe

- La **clé privée** est constituée de deux "grands" nombres premiers p et q.
- La **clé publique** est constituée du produit $n = pq$ et d'un entier e inversible modulo $\varphi(n)$, c'est à dire un entier e tel qu'il existe un entier d avec $ed \equiv 1 \pmod{\varphi(n)}$.
- Le **chiffré d'un message** représenté par un entier M modulo n est $M^e \pmod{n}$.
- Pour **déchiffer le message** chiffré C, il suffit de calculer $C^d \pmod{n}$ où d est l'inverse de e modulo $\varphi(n)$.
- Justification de l'algorithme :
  - Supposons que M soit premier avec n. Alors, d'après le théorème d'*Euler-Fermat*, on a : $M^{\varphi(n)} \equiv 1 \pmod{n}$. Par conséquent : $C^d = (M^e)^d = M^{ed} = M^{1+k\varphi(n)} = M \times (M^{\varphi(n)})^k \equiv M \pmod{n}$
  - Lorsque M n'est pas premier avec n, trois cas se présentent : $p \mid M$ et $q \nmid M$; $p \nmid M$ et $q \mid M$; $n \mid M$. Le dernier cas est trivial tandis que les deux premiers nécessitent l'utilisation du *théorème chinois* en plus du théorème d'*Euler-Fermat* pour conclure.

Le principe

- La **clé privée** est constituée de deux "grands" nombres premiers p et q.
- La **clé publique** est constituée du produit $n = pq$ et d'un entier e inversible modulo $\varphi(n)$, c'est à dire un entier e tel qu'il existe un entier d avec $ed \equiv 1 \pmod{\varphi(n)}$.
- Le **chiffré d'un message** représenté par un entier M modulo n est $M^e \pmod{n}$.
- Pour **déchiffer le message** chiffré C, il suffit de calculer $C^d \pmod{n}$ où d est l'inverse de e modulo $\varphi(n)$.
- Justification de l'algorithme :
  - Supposons que M soit premier avec n. Alors, d'après le théorème d'*Euler-Fermat*, on a : $M^{\varphi(n)} \equiv 1 \pmod{n}$. Par conséquent : $C^d = (M^e)^d = M^{ed} = M^{1+k\varphi(n)} = M \times (M^{\varphi(n)})^k \equiv M \pmod{n}$
  - Lorsque M n'est pas premier avec n, trois cas se présentent : $p \mid M$ et $q \nmid M$; $p \nmid M$ et $q \mid M$; $n \mid M$. Le dernier cas est trivial tandis que les deux premiers nécessitent l'utilisation du *théorème chinois* en plus du théorème d'*Euler-Fermat* pour conclure.

Le principe

- La **clé privée** est constituée de deux "grands" nombres premiers p et q.
- La **clé publique** est constituée du produit $n = pq$ et d'un entier e inversible modulo $\varphi(n)$, c'est à dire un entier e tel qu'il existe un entier d avec $ed \equiv 1 \pmod{\varphi(n)}$.
- Le **chiffré d'un message** représenté par un entier M modulo n est $M^e \pmod{n}$.
- Pour **déchiffer le message** chiffré C, il suffit de calculer $C^d \pmod{n}$ où d est l'inverse de e modulo $\varphi(n)$.
- Justification de l'algorithme :
  - Supposons que M soit premier avec n. Alors, d'après le théorème d'*Euler-Fermat*, on a : $M^{\varphi(n)} \equiv 1 \pmod{n}$. Par conséquent : $C^d = (M^e)^d = M^{ed} = M^{1+k\varphi(n)} = M \times (M^{\varphi(n)})^k \equiv M \pmod{n}$
  - Lorsque M n'est pas premier avec n, trois cas se présentent : $p \mid M$ et $q \nmid M$; $p \nmid M$ et $q \mid M$; $n \mid M$. Le dernier cas est trivial tandis que les deux premiers nécessitent l'utilisation du *théorème chinois* en plus du théorème d'*Euler-Fermat* pour conclure.

Schéma de fonctionnement

Schéma de fonctionnement

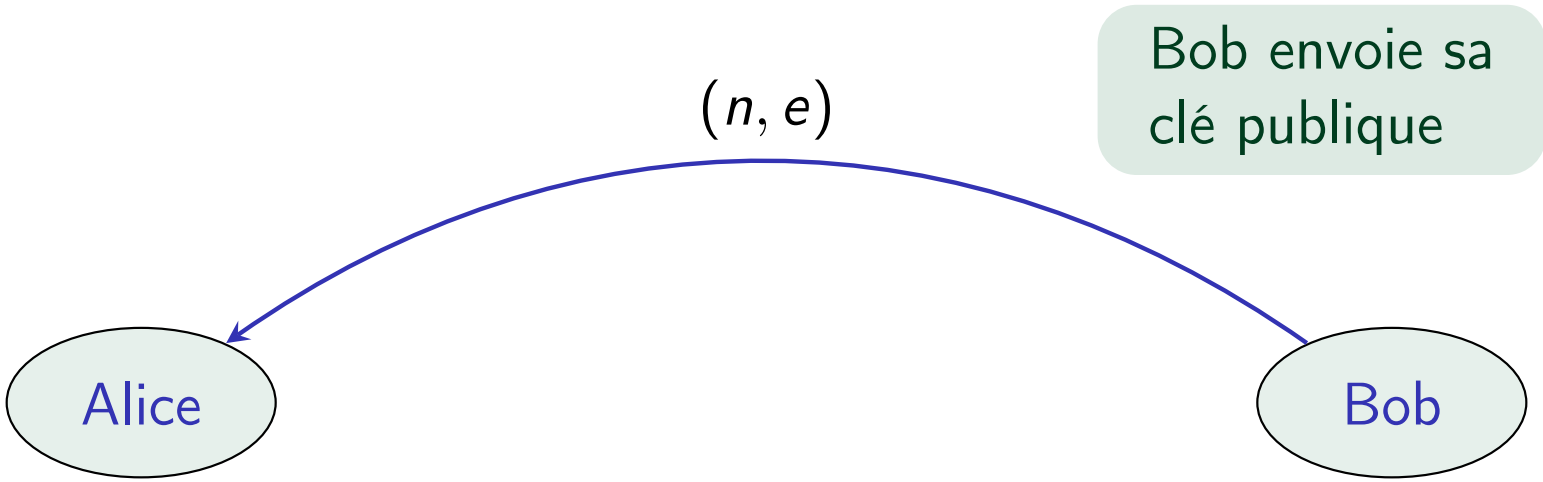


Schéma de fonctionnement

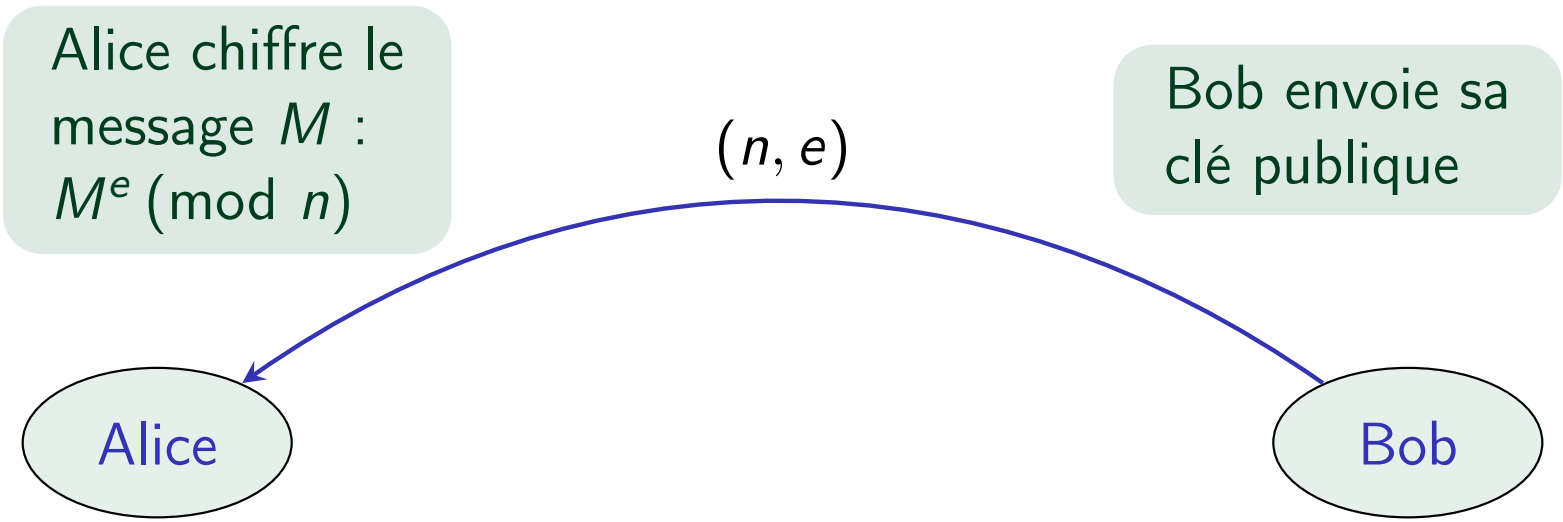


Schéma de fonctionnement

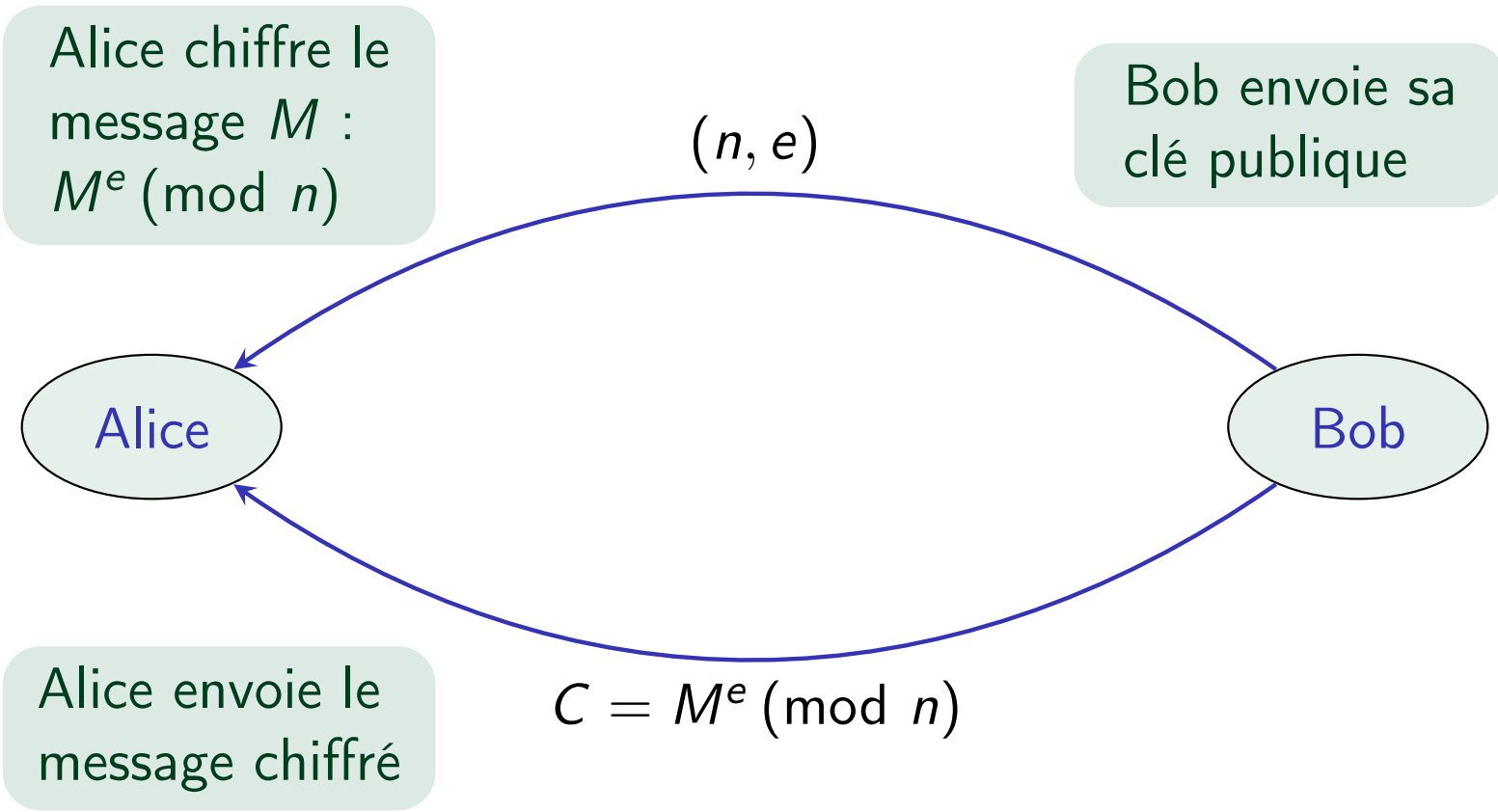
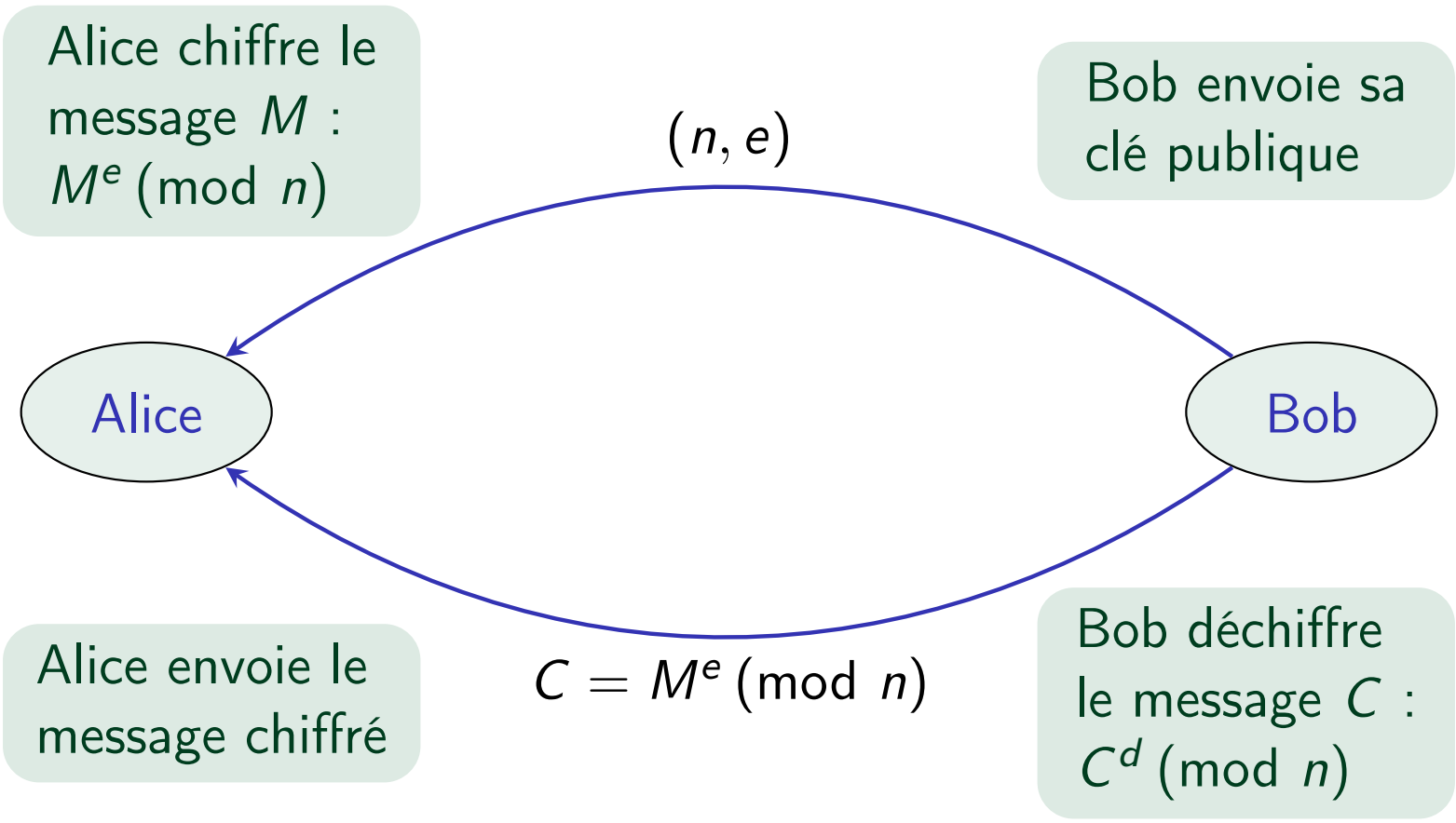


Schéma de fonctionnement



Un exemple concret

Prenons $p = 23$ et $q = 61$, on aura $n = 23 \times 61 = 1403$. Par ailleurs :

$$\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1) = 22 \times 60 = 1320.$$

Il faut maintenant choisir e premier avec $\varphi(n) = 1320$. Prenons $e = 7$. Il reste maintenant à calculer d tel que $ed \equiv 1 \pmod{1320}$.

Théorème (Bezout)

Soient $a, b \in \mathbb{Z}$ non tous deux nuls. Alors, il existe $\lambda, \mu \in \mathbb{Z}$ tels que : $\lambda a + \mu b = \text{pgcd}(a, b)$. Les coefficients λ et μ peuvent être calculés de manière efficace à l'aide l'algorithme d'Euclide étendu.

Donc comme e et $\varphi(n)$ sont premiers entre eux, il existe d, f tels que : $de + f\varphi(n) = \text{pgcd}(e, \varphi(n)) = 1$, soit, modulo $\varphi(n)$, $de \equiv 1$. d est donc l'inverse de e cherché. Dans notre exemple, on peut prendre $d = 943$.

Un exemple concret

Prenons $p = 23$ et $q = 61$, on aura $n = 23 \times 61 = 1403$. Par ailleurs :

$$\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1) = 22 \times 60 = 1320.$$

Il faut maintenant choisir e premier avec $\varphi(n) = 1320$. Prenons $e = 7$. Il reste maintenant à calculer d tel que $ed \equiv 1 \pmod{1320}$.

Théorème (Bezout)

Soient $a, b \in \mathbb{Z}$ non tous deux nuls. Alors, il existe $\lambda, \mu \in \mathbb{Z}$ tels que : $\lambda a + \mu b = \text{pgcd}(a, b)$. Les coefficients λ et μ peuvent être calculés de manière efficace à l'aide l'algorithme d'Euclide étendu.

Donc comme e et $\varphi(n)$ sont premiers entre eux, il existe d, f tels que : $de + f\varphi(n) = \text{pgcd}(e, \varphi(n)) = 1$, soit, modulo $\varphi(n)$, $de \equiv 1$. d est donc l'inverse de e cherché. Dans notre exemple, on peut prendre $d = 943$.

Un exemple concret

Prenons $p = 23$ et $q = 61$, on aura $n = 23 \times 61 = 1403$. Par ailleurs :

$$\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1) = 22 \times 60 = 1320.$$

Il faut maintenant choisir e premier avec $\varphi(n) = 1320$. Prenons $e = 7$. Il reste maintenant à calculer d tel que $ed \equiv 1 \pmod{1320}$.

Théorème (Bezout)

Soient $a, b \in \mathbb{Z}$ non tous deux nuls. Alors, il existe $\lambda, \mu \in \mathbb{Z}$ tels que : $\lambda a + \mu b = \text{pgcd}(a, b)$. Les coefficients λ et μ peuvent être calculés de manière efficace à l'aide l'algorithme d'Euclide étendu.

Donc comme e et $\varphi(n)$ sont premiers entre eux, il existe d, f tels que : $de + f\varphi(n) = \text{pgcd}(e, \varphi(n)) = 1$, soit, modulo $\varphi(n)$, $de \equiv 1$. d est donc l'inverse de e cherché. Dans notre exemple, on peut prendre $d = 943$.

Un exemple concret

Prenons $p = 23$ et $q = 61$, on aura $n = 23 \times 61 = 1403$. Par ailleurs :

$$\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1) = 22 \times 60 = 1320.$$

Il faut maintenant choisir e premier avec $\varphi(n) = 1320$. Prenons $e = 7$. Il reste maintenant à calculer d tel que $ed \equiv 1 \pmod{1320}$.

Théorème (Bezout)

Soient $a, b \in \mathbb{Z}$ non tous deux nuls. Alors, il existe $\lambda, \mu \in \mathbb{Z}$ tels que : $\lambda a + \mu b = \text{pgcd}(a, b)$. Les coefficients λ et μ peuvent être calculés de manière efficace à l'aide l'algorithme d'Euclide étendu.

Donc comme e et $\varphi(n)$ sont premiers entre eux, il existe d, f tels que : $de + f\varphi(n) = \text{pgcd}(e, \varphi(n)) = 1$, soit, modulo $\varphi(n)$, $de \equiv 1$. d est donc l'inverse de e cherché. Dans notre exemple, on peut prendre $d = 943$.

Un exemple concret

Donc, si on résume les calculs précédents, on a :

- La clé publique est la paire d'entiers $n = 1403$, $e = 7$.
- La clé privée est l'entier $d = 943$.
- Pour chiffrer le message $M = 345$, il faut calculer $M^e \pmod{n}$, ce qui nous donne $C = 345^7 \pmod{1403} = 1058$.
- Pour déchiffrer C, il faut calculer $C^d \pmod{n}$, c'est à dire évaluer $1058^{943} \pmod{1403}$. On retrouve bien $M = 345$.

Bien entendu, cet exemple n'est pas réaliste en raison de la trop petite taille de la clé publique n. La recommandation actuelle pour la taille de la clé publique n est 2048 bits.

En pratique, on fabrique rarement ses clés, on utilise plutôt des logiciels comme *OpenSSL* pour générer des clés RSA.

Un exemple concret

Donc, si on résume les calculs précédents, on a :

- La clé publique est la paire d'entiers $n = 1403$, $e = 7$.
- La clé privée est l'entier $d = 943$.
- Pour chiffrer le message $M = 345$, il faut calculer $M^e \pmod{n}$, ce qui nous donne $C = 345^7 \pmod{1403} = 1058$.
- Pour déchiffrer C, il faut calculer $C^d \pmod{n}$, c'est à dire évaluer $1058^{943} \pmod{1403}$. On retrouve bien $M = 345$.

Bien entendu, cet exemple n'est pas réaliste en raison de la trop petite taille de la clé publique n. La recommandation actuelle pour la taille de la clé publique n est 2048 bits.

En pratique, on fabrique rarement ses clés, on utilise plutôt des logiciels comme *OpenSSL* pour générer des clés RSA.

Un exemple concret

Donc, si on résume les calculs précédents, on a :

- La clé publique est la paire d'entiers $n = 1403$, $e = 7$.
- La clé privée est l'entier $d = 943$.
- Pour chiffrer le message $M = 345$, il faut calculer $M^e \pmod{n}$, ce qui nous donne $C = 345^7 \pmod{1403} = 1058$.
- Pour déchiffrer C, il faut calculer $C^d \pmod{n}$, c'est à dire évaluer $1058^{943} \pmod{1403}$. On retrouve bien $M = 345$.

Bien entendu, cet exemple n'est pas réaliste en raison de la trop petite taille de la clé publique n. La recommandation actuelle pour la taille de la clé publique n est 2048 bits.

En pratique, on fabrique rarement ses clés, on utilise plutôt des logiciels comme *OpenSSL* pour générer des clés RSA.

Un exemple concret

Donc, si on résume les calculs précédents, on a :

- La clé publique est la paire d'entiers $n = 1403$, $e = 7$.
- La clé privée est l'entier $d = 943$.
- Pour chiffrer le message $M = 345$, il faut calculer $M^e \pmod{n}$, ce qui nous donne $C = 345^7 \pmod{1403} = 1058$.
- Pour déchiffrer C, il faut calculer $C^d \pmod{n}$, c'est à dire évaluer $1058^{943} \pmod{1403}$. On retrouve bien $M = 345$.

Bien entendu, cet exemple n'est pas réaliste en raison de la trop petite taille de la clé publique n. La recommandation actuelle pour la taille de la clé publique n est 2048 bits.

En pratique, on fabrique rarement ses clés, on utilise plutôt des logiciels comme *OpenSSL* pour générer des clés RSA.

Un exemple concret

Donc, si on résume les calculs précédents, on a :

- La clé publique est la paire d'entiers $n = 1403$, $e = 7$.
- La clé privée est l'entier $d = 943$.
- Pour chiffrer le message $M = 345$, il faut calculer $M^e \pmod{n}$, ce qui nous donne $C = 345^7 \pmod{1403} = 1058$.
- Pour déchiffrer C, il faut calculer $C^d \pmod{n}$, c'est à dire évaluer $1058^{943} \pmod{1403}$. On retrouve bien $M = 345$.

Bien entendu, cet exemple n'est pas réaliste en raison de la trop petite taille de la clé publique n. La recommandation actuelle pour la taille de la clé publique n est 2048 bits.

En pratique, on fabrique rarement ses clés, on utilise plutôt des logiciels comme *OpenSSL* pour générer des clés RSA.

Un exemple concret

Donc, si on résume les calculs précédents, on a :

- La clé publique est la paire d'entiers $n = 1403$, $e = 7$.
- La clé privée est l'entier $d = 943$.
- Pour chiffrer le message $M = 345$, il faut calculer $M^e \pmod{n}$, ce qui nous donne $C = 345^7 \pmod{1403} = 1058$.
- Pour déchiffrer C, il faut calculer $C^d \pmod{n}$, c'est à dire évaluer $1058^{943} \pmod{1403}$. On retrouve bien $M = 345$.

Bien entendu, cet exemple n'est pas réaliste en raison de la trop petite taille de la clé publique n. La recommandation actuelle pour la taille de la clé publique n est 2048 bits.

En pratique, on fabrique rarement ses clés, on utilise plutôt des logiciels comme *OpenSSL* pour générer des clés RSA.

Un exemple concret

Donc, si on résume les calculs précédents, on a :

- La clé publique est la paire d'entiers $n = 1403$, $e = 7$.
- La clé privée est l'entier $d = 943$.
- Pour chiffrer le message $M = 345$, il faut calculer $M^e \pmod{n}$, ce qui nous donne $C = 345^7 \pmod{1403} = 1058$.
- Pour déchiffrer C, il faut calculer $C^d \pmod{n}$, c'est à dire évaluer $1058^{943} \pmod{1403}$. On retrouve bien $M = 345$.

Bien entendu, cet exemple n'est pas réaliste en raison de la trop petite taille de la clé publique n. La recommandation actuelle pour la taille de la clé publique n est 2048 bits.

En pratique, on fabrique rarement ses clés, on utilise plutôt des logiciels comme *OpenSSL* pour générer des clés RSA.

Génération de clés RSA avec *OpenSSL*

Pour générer une clé RSA de 2048 bits avec $e = F_4 = 2^{2^4} + 1 = 65537$ (e est le 5ème nombre de *Fermat*) :

```
$ openssl genrsa -F4 -out rsa.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++ # Rabin-Miller
.....................................................+++
e is 65537 (0x10001)
```

Le fichier généré *rsa.key* est au format *base64*. Attention : la clé privée est **stockée en clair** dans le fichier. Donc il vaut mieux procéder ainsi :

```
$ openssl genrsa -aes128 -F4 -out rsa.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++
.....................................................+++
e is 65537 (0x10001)
Enter pass phrase for rsa.key: # MDP chiffrement AES-128
Verifying - Enter pass phrase for rsa.key: # confirmation
```

Génération de clés RSA avec *OpenSSL*

Pour générer une clé RSA de 2048 bits avec $e = F_4 = 2^{2^4} + 1 = 65537$ (e est le 5ème nombre de *Fermat*) :

```
$ openssl genrsa -F4 -out rsa.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++ # Rabin-Miller
.............................................+++
e is 65537 (0x10001)
```

Le fichier généré *rsa.key* est au format *base64*. Attention : la clé privée est **stockée en clair** dans le fichier. Donc il vaut mieux procéder ainsi :

```
$ openssl genrsa -aes128 -F4 -out rsa.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++
.....................................+++
e is 65537 (0x10001)
Enter pass phrase for rsa.key: # MDP chiffrement AES-128
Verifying - Enter pass phrase for rsa.key: # confirmation
```

Génération de clés RSA avec *OpenSSL*

Pour générer une clé RSA de 2048 bits avec $e = F_4 = 2^{2^4} + 1 = 65537$ (e est le 5ème nombre de *Fermat*) :

```
$ openssl genrsa -F4 -out rsa.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++ # Rabin-Miller
.............................................+++
e is 65537 (0x10001)
```

Le fichier généré *rsa.key* est au format *base64*. Attention : la clé privée est **stockée en clair** dans le fichier. Donc il vaut mieux procéder ainsi :

```
$ openssl genrsa -aes128 -F4 -out rsa.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++
.............................................+++
e is 65537 (0x10001)
Enter pass phrase for rsa.key: # MDP chiffrement AES-128
Verifying - Enter pass phrase for rsa.key: # confirmation
```

Génération de clés RSA avec *OpenSSL*

Pour générer une clé RSA de 2048 bits avec $e = F_4 = 2^{2^4} + 1 = 65537$ (e est le 5ème nombre de *Fermat*) :

```
$ openssl genrsa -F4 -out rsa.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++ # Rabin-Miller
.....................................................+++
e is 65537 (0x10001)
```

Le fichier généré *rsa.key* est au format *base64*. Attention : la clé privée est **stockée en clair** dans le fichier. Donc il vaut mieux procéder ainsi :

```
$ openssl genrsa -aes128 -F4 -out rsa.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++
.....................................................+++
e is 65537 (0x10001)
Enter pass phrase for rsa.key: # MDP chiffrement AES-128
Verifying - Enter pass phrase for rsa.key: # confirmation
```

Génération de clés RSA avec *OpenSSL*

Pour voir en clair le fichier *rsa.key* :

```
$ openssl rsa -in rsa.key -noout -text
Private-Key: (2048 bit)
modulus: # clé publique n = pq
    00:f2:22:be:06:40:b9:55:5a:a6:b3:52:8c:72:8a:
    ..........................................
publicExponent: 65537 (0x10001) # clé publique e = F_4
privateExponent: # clé privée d, ed ≡ 1 (mod φ(n))
    57:a9:aa:60:7b:28:5e:35:8e:aa:d7:95:0f:96:dc:
    ..........................................
prime1: # p pseudo-premier (3 tests Rabin-Miller)
    00:fb:26:f8:56:9b:fa:05:0e:16:7e:78:d8:70:1b:
    ..........................................
prime2: # q pseudo-premier (3 tests Rabin-Miller)
    00:f6:cf:37:ef:61:ed:3a:e7:86:04:b4:1e:1f:e0:
    ..........................................
```

Génération de clés RSA avec *OpenSSL*

Pour voir en clair le fichier *rsa.key* :

```
$ openssl rsa -in rsa.key -noout -text
Private-Key: (2048 bit)
modulus: # clé publique n = pq
    00:f2:22:be:06:40:b9:55:5a:a6:b3:52:8c:72:8a:
    ..............................................
publicExponent: 65537 (0x10001) # clé publique e = F4
privateExponent: # clé privée d, ed ≡ 1 (mod φ(n))
    57:a9:aa:60:7b:28:5e:35:8e:aa:d7:95:0f:96:dc:
    ..............................................
prime1: # p pseudo-premier (3 tests Rabin-Miller)
    00:fb:26:f8:56:9b:fa:05:0e:16:7e:78:d8:70:1b:
    ..............................................
prime2: # q pseudo-premier (3 tests Rabin-Miller)
    00:f6:cf:37:ef:61:ed:3a:e7:86:04:b4:1e:1f:e0:
    ..............................................
```

Signature RSA

- Le système RSA permet également de *signer* des données numériques.
- Pour ce faire, il suffit d'inverser les rôles de e et d dans l'algorithme décrit précédemment.
- Supposons qu'*Alice* veuille envoyer à *Bob* un message M signé. Avec sa clé privée d, elle va générer la signature $S = M^d \pmod{n}$ du message M, où n est la clé publique.
- Elle envoie ensuite à *Bob* le couple (M, S).
- Pour authentifier la signature, *Bob*, lorsqu'il reçoit le couple (M, S), doit d'abord calculer $M' = S^e \pmod{n}$, où e est l'exposant public de la clé d'*Alice*, et ensuite vérifier que $M' = M$.
- Une telle signature est beaucoup plus fiable qu'une signature manuelle sur un document faxé !

Signature RSA

- Le système RSA permet également de *signer* des données numériques.
- Pour ce faire, il suffit d'inverser les rôles de e et d dans l'algorithme décrit précédemment.
- Supposons qu'*Alice* veuille envoyer à *Bob* un message M signé. Avec sa clé privée d, elle va générer la signature $S = M^d \pmod{n}$ du message M, où n est la clé publique.
- Elle envoie ensuite à *Bob* le couple (M, S).
- Pour authentifier la signature, *Bob*, lorsqu'il reçoit le couple (M, S), doit d'abord calculer $M' = S^e \pmod{n}$, où e est l'exposant public de la clé d'*Alice*, et ensuite vérifier que $M' = M$.
- Une telle signature est beaucoup plus fiable qu'une signature manuelle sur un document faxé !

Signature RSA

- Le système RSA permet également de *signer* des données numériques.
- Pour ce faire, il suffit d'inverser les rôles de e et d dans l'algorithme décrit précédemment.
- Supposons qu'*Alice* veuille envoyer à *Bob* un message M signé. Avec sa clé privée d, elle va générer la signature $S = M^d \pmod{n}$ du message M, où n est la clé publique.
- Elle envoie ensuite à *Bob* le couple (M, S).
- Pour authentifier la signature, *Bob*, lorsqu'il reçoit le couple (M, S), doit d'abord calculer $M' = S^e \pmod{n}$, où e est l'exposant public de la clé d'*Alice*, et ensuite vérifier que $M' = M$.
- Une telle signature est beaucoup plus fiable qu'une signature manuelle sur un document faxé !

Signature RSA

- Le système RSA permet également de *signer* des données numériques.
- Pour ce faire, il suffit d'inverser les rôles de e et d dans l'algorithme décrit précédemment.
- Supposons qu'*Alice* veuille envoyer à *Bob* un message M signé. Avec sa clé privée d, elle va générer la signature $S = M^d \pmod{n}$ du message M, où n est la clé publique.
- Elle envoie ensuite à *Bob* le couple (M, S).
- Pour authentifier la signature, *Bob*, lorsqu'il reçoit le couple (M, S), doit d'abord calculer $M' = S^e \pmod{n}$, où e est l'exposant public de la clé d'*Alice*, et ensuite vérifier que $M' = M$.
- Une telle signature est beaucoup plus fiable qu'une signature manuelle sur un document faxé !

Signature RSA

- Le système RSA permet également de *signer* des données numériques.
- Pour ce faire, il suffit d'inverser les rôles de e et d dans l'algorithme décrit précédemment.
- Supposons qu'*Alice* veuille envoyer à *Bob* un message M signé. Avec sa clé privée d, elle va générer la signature $S = M^d \pmod{n}$ du message M, où n est la clé publique.
- Elle envoie ensuite à *Bob* le couple (M, S).
- Pour authentifier la signature, *Bob*, lorsqu'il reçoit le couple (M, S), doit d'abord calculer $M' = S^e \pmod{n}$, où e est l'exposant public de la clé d'*Alice*, et ensuite vérifier que $M' = M$.
- Une telle signature est beaucoup plus fiable qu'une signature manuelle sur un document faxé !

Signature RSA

- Le système RSA permet également de *signer* des données numériques.
- Pour ce faire, il suffit d'inverser les rôles de e et d dans l'algorithme décrit précédemment.
- Supposons qu'*Alice* veuille envoyer à *Bob* un message M signé. Avec sa clé privée d, elle va générer la signature $S = M^d \pmod{n}$ du message M, où n est la clé publique.
- Elle envoie ensuite à *Bob* le couple (M, S).
- Pour authentifier la signature, *Bob*, lorsqu'il reçoit le couple (M, S), doit d'abord calculer $M' = S^e \pmod{n}$, où e est l'exposant public de la clé d'*Alice*, et ensuite vérifier que $M' = M$.
- Une telle signature est beaucoup plus fiable qu'une signature manuelle sur un document faxé !

Considérations générales

- Rappelons que la solidité de RSA repose sur l'impossibilité, **avec les moyens et connaissances actuels**, de factoriser, en un temps raisonnable, $n = pq$ lorsque n est suffisamment grand (2048 bits par exemple).
- Il faut bien évidemment qu'aucun des facteurs p et q ne soit trop petit. Il faut donc que p et q soient d'une taille analogue sans pour autant être **trop proches**. En effet, s'il existe $c > 0$ tel que $|p - q| < c\sqrt[4]{n}$, alors on peut factoriser n en un temps polynomial dépendant de c. Donc cette attaque est praticable dès que la constante c est suffisamment petite.
- On pourrait penser à attaquer le système RSA via $\varphi(n)$. Or connaître $\varphi(n)$ équivaut à savoir factoriser n. Effet, on a $p + q = n + 1 - \varphi(n)$ et $pq = n$. Donc, si on connaît $\varphi(n)$, on obtient immédiatement p et q.
- On peut aussi se demander si l'on sait obtenir d sans connaître $\varphi(n)$. On montre, qu'en fait, cela revient encore à savoir factoriser n.

Considérations générales

- Rappelons que la solidité de RSA repose sur l'impossibilité, **avec les moyens et connaissances actuels**, de factoriser, en un temps raisonnable, $n = pq$ lorsque n est suffisamment grand (2048 bits par exemple).
- Il faut bien évidemment qu'aucun des facteurs p et q ne soit trop petit. Il faut donc que p et q soient d'une taille analogue sans pour autant être **trop proches**. En effet, s'il existe $c > 0$ tel que $|p - q| < c\sqrt[4]{n}$, alors on peut factoriser n en un temps polynomial dépendant de c. Donc cette attaque est praticable dès que la constante c est suffisamment petite.
- On pourrait penser à attaquer le système RSA via $\varphi(n)$. Or connaître $\varphi(n)$ équivaut à savoir factoriser n. Effet, on a $p + q = n + 1 - \varphi(n)$ et $pq = n$. Donc, si on connaît $\varphi(n)$, on obtient immédiatement p et q.
- On peut aussi se demander si l'on sait obtenir d sans connaître $\varphi(n)$. On montre, qu'en fait, cela revient encore à savoir factoriser n.

Considérations générales

- Rappelons que la solidité de RSA repose sur l'impossibilité, **avec les moyens et connaissances actuels**, de factoriser, en un temps raisonnable, $n = pq$ lorsque n est suffisamment grand (2048 bits par exemple).
- Il faut bien évidemment qu'aucun des facteurs p et q ne soit trop petit. Il faut donc que p et q soient d'une taille analogue sans pour autant être **trop proches**. En effet, s'il existe $c > 0$ tel que $|p - q| < c\sqrt[4]{n}$, alors on peut factoriser n en un temps polynomial dépendant de c. Donc cette attaque est praticable dès que la constante c est suffisamment petite.
- On pourrait penser à attaquer le système RSA via $\varphi(n)$. Or connaître $\varphi(n)$ équivaut à savoir factoriser n. Effet, on a $p + q = n + 1 - \varphi(n)$ et $pq = n$. Donc, si on connaît $\varphi(n)$, on obtient immédiatement p et q.
- On peut aussi se demander si l'on sait obtenir d sans connaître $\varphi(n)$. On montre, qu'en fait, cela revient encore à savoir factoriser n.

Considérations générales

- Rappelons que la solidité de RSA repose sur l'impossibilité, **avec les moyens et connaissances actuels**, de factoriser, en un temps raisonnable, $n = pq$ lorsque n est suffisamment grand (2048 bits par exemple).
- Il faut bien évidemment qu'aucun des facteurs p et q ne soit trop petit. Il faut donc que p et q soient d'une taille analogue sans pour autant être **trop proches**. En effet, s'il existe $c > 0$ tel que $|p - q| < c\sqrt[4]{n}$, alors on peut factoriser n en un temps polynomial dépendant de c. Donc cette attaque est praticable dès que la constante c est suffisamment petite.
- On pourrait penser à attaquer le système RSA via $\varphi(n)$. Or connaître $\varphi(n)$ équivaut à savoir factoriser n. Effet, on a $p + q = n + 1 - \varphi(n)$ et $pq = n$. Donc, si on connaît $\varphi(n)$, on obtient immédiatement p et q.
- On peut aussi se demander si l'on sait obtenir d sans connaître $\varphi(n)$. On montre, qu'en fait, cela revient encore à savoir factoriser n.

Quelques attaques

- La cryptanalyse du système RSA a été largement étudiée. Beaucoup d'attaques possibles ont été ainsi découvertes dont certaines très astucieuses. En voici deux.
- Si on choisit l'exposant public e trop petit (pour accélérer le chiffrement), par exemple $e = 3$, une attaque possible est l'attaque due à *Håstad* en 1985. Supposons qu'*Alice* envoie un même message M à trois destinataires dont les clés publiques sont $(n_1, 3)$, $(n_2, 3)$, et $(n_3, 3)$. Si l'on suppose que les entiers n_i, $i = 1, 2, 3$, sont premiers entre eux et que $M < \inf(n_1, n_2, n_3)$, on obtient M facilement par simple application du théorème chinois.
- De même, si l'exposant privé d est choisi trop petit (pour accélérer le déchiffrement), *Wiener* a montré en 1989 l'existence d'un algorithme en temps polynomial permettant de retrouver d. Plus précisément, si p et q ont le même nombre de bits et si $d < \frac{1}{3}\sqrt[4]{n}$, on peut calculer d à l'aide du développement en fraction continue de $\frac{e}{n}$.

Quelques attaques

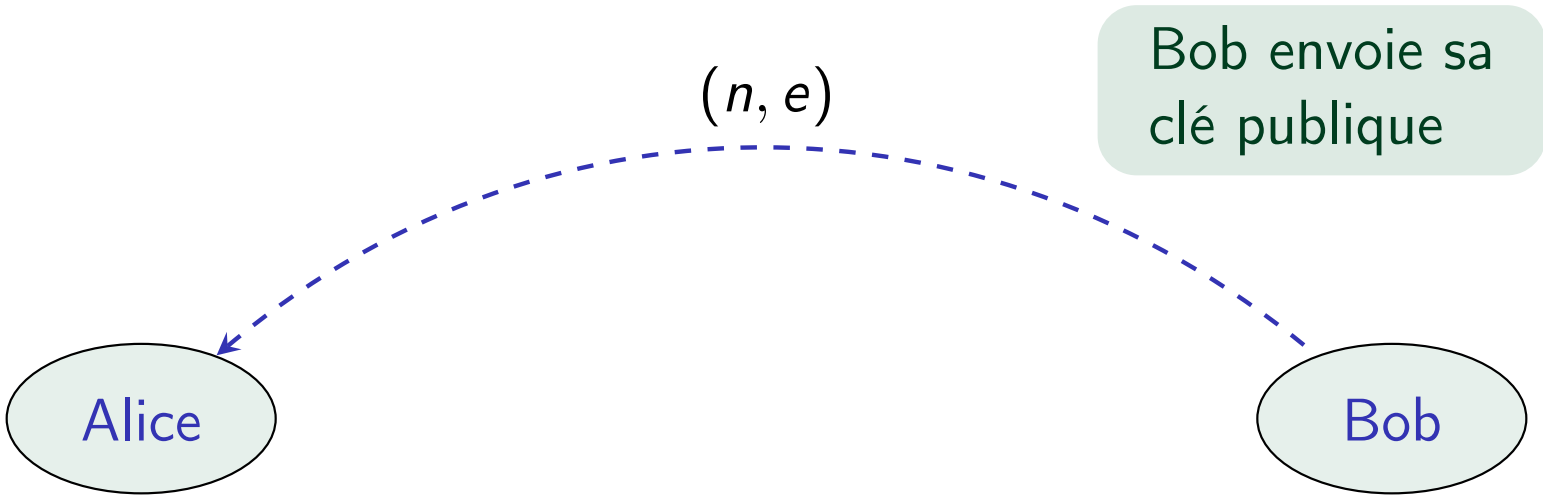
- La cryptanalyse du système RSA a été largement étudiée. Beaucoup d'attaques possibles ont été ainsi découvertes dont certaines très astucieuses. En voici deux.
- Si on choisit l'exposant public e trop petit (pour accélérer le chiffrement), par exemple $e = 3$, une attaque possible est l'attaque due à *Håstad* en 1985. Supposons qu'*Alice* envoie un même message M à trois destinataires dont les clés publiques sont $(n_1, 3)$, $(n_2, 3)$, et $(n_3, 3)$. Si l'on suppose que les entiers n_i, $i = 1, 2, 3$, sont premiers entre eux et que $M < \inf(n_1, n_2, n_3)$, on obtient M facilement par simple application du théorème chinois.
- De même, si l'exposant privé d est choisi trop petit (pour accélérer le déchiffrement), *Wiener* a montré en 1989 l'existence d'un algorithme en temps polynomial permettant de retrouver d. Plus précisément, si p et q ont le même nombre de bits et si $d < \frac{1}{3}\sqrt[4]{n}$, on peut calculer d à l'aide du développement en fraction continue de $\frac{e}{n}$.

Quelques attaques

- La cryptanalyse du système RSA a été largement étudiée. Beaucoup d'attaques possibles ont été ainsi découvertes dont certaines très astucieuses. En voici deux.
- Si on choisit l'exposant public e trop petit (pour accélérer le chiffrement), par exemple $e = 3$, une attaque possible est l'attaque due à *Håstad* en 1985. Supposons qu'*Alice* envoie un même message M à trois destinataires dont les clés publiques sont $(n_1, 3)$, $(n_2, 3)$, et $(n_3, 3)$. Si l'on suppose que les entiers n_i, $i = 1, 2, 3$, sont premiers entre eux et que $M < \inf(n_1, n_2, n_3)$, on obtient M facilement par simple application du théorème chinois.
- De même, si l'exposant privé d est choisi trop petit (pour accélérer le déchiffrement), *Wiener* a montré en 1989 l'existence d'un algorithme en temps polynomial permettant de retrouver d. Plus précisément, si p et q ont le même nombre de bits et si $d < \frac{1}{3}\sqrt[4]{n}$, on peut calculer d à l'aide du développement en fraction continue de $\frac{e}{n}$.

Attaque “man in the middle”

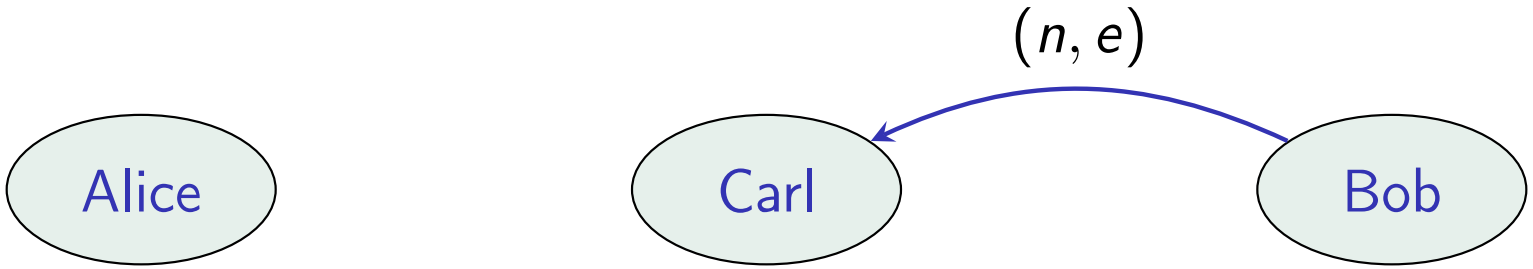
Attaque "man in the middle"



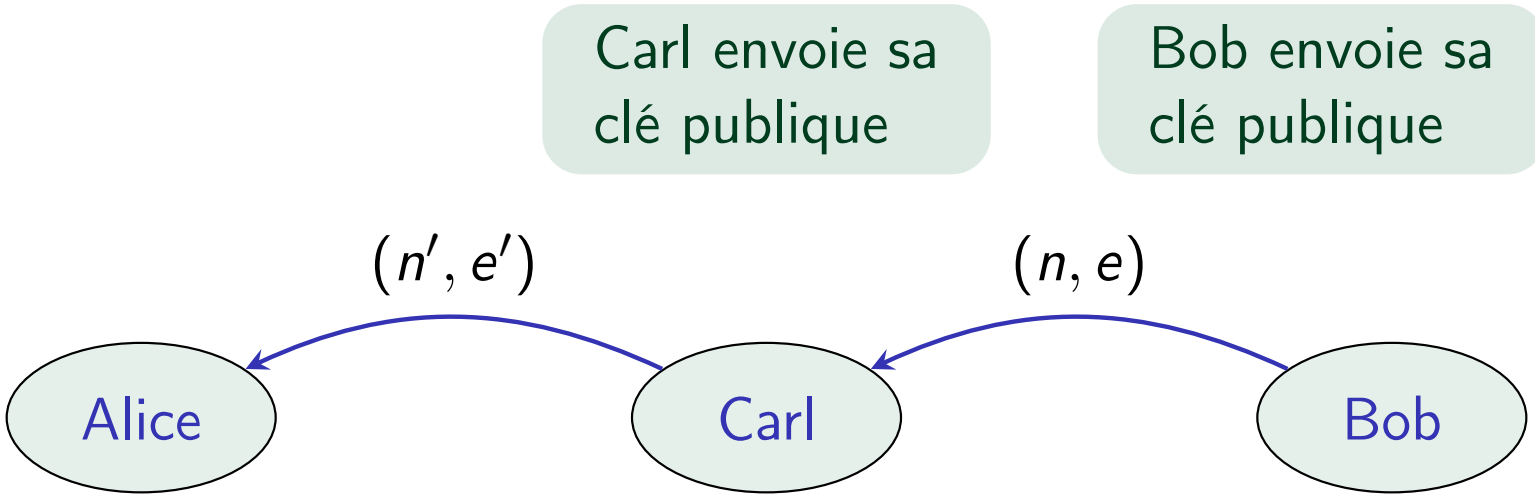
Attaque "man in the middle"

Carl intercepte la clé publique

Bob envoie sa clé publique



Attaque "man in the middle"

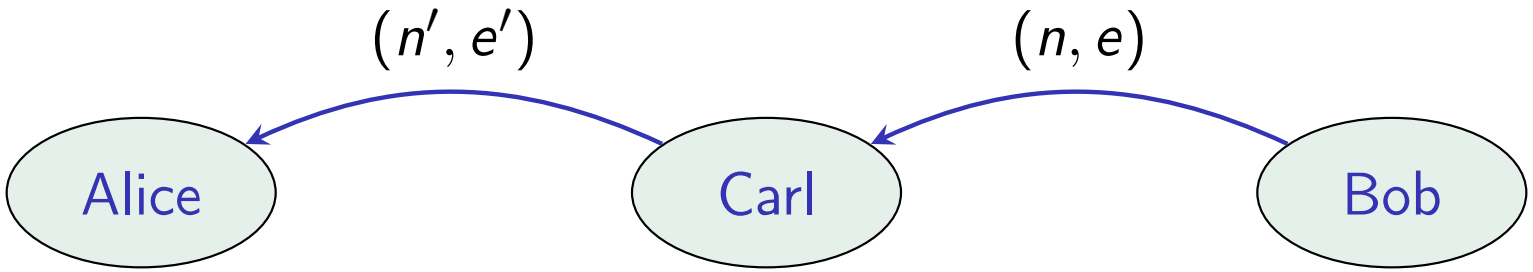


Attaque "man in the middle"

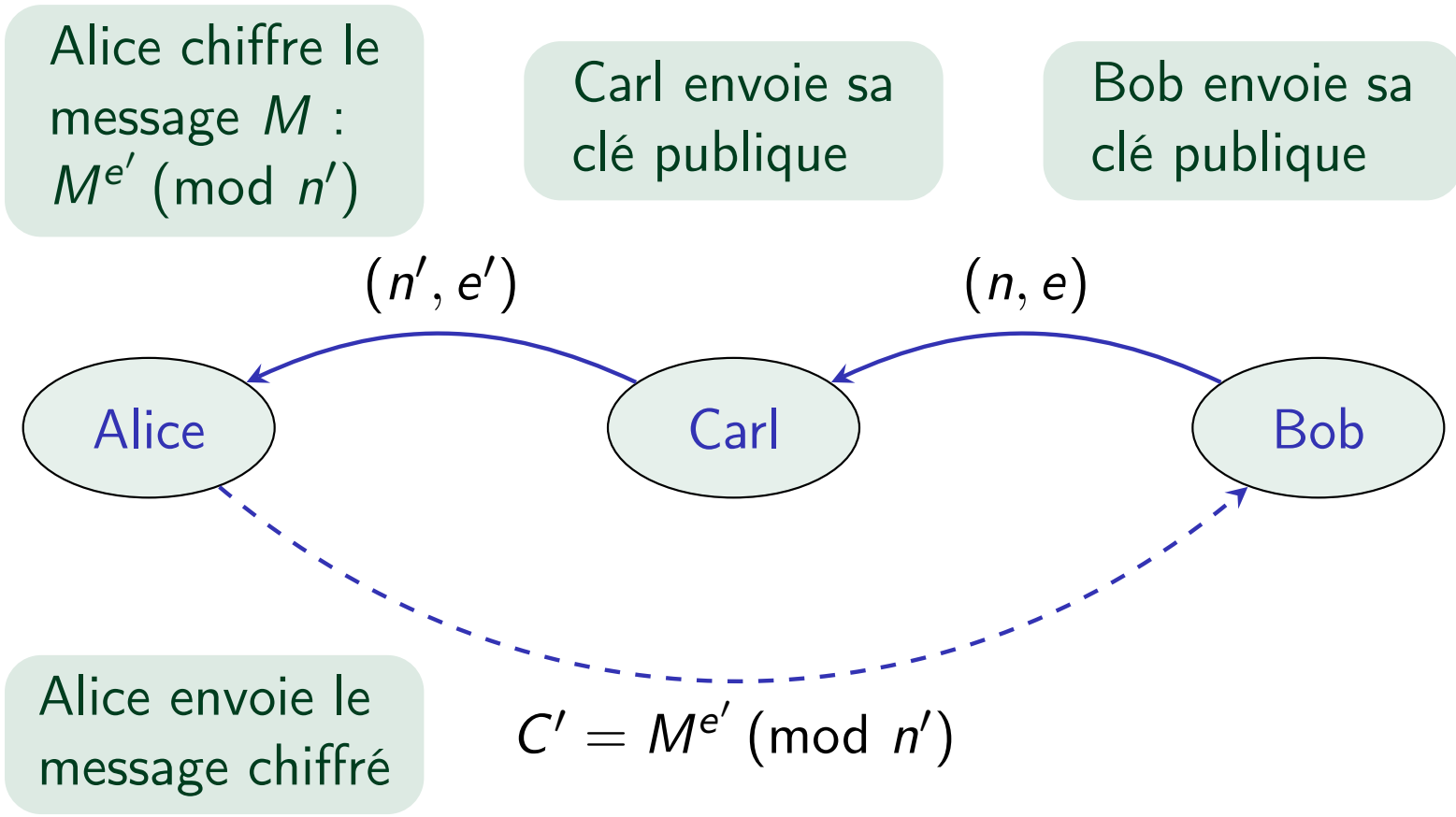
Alice chiffre le message M : $M^{e'} \pmod{n'}$

Carl envoie sa clé publique

Bob envoie sa clé publique



Attaque "man in the middle"

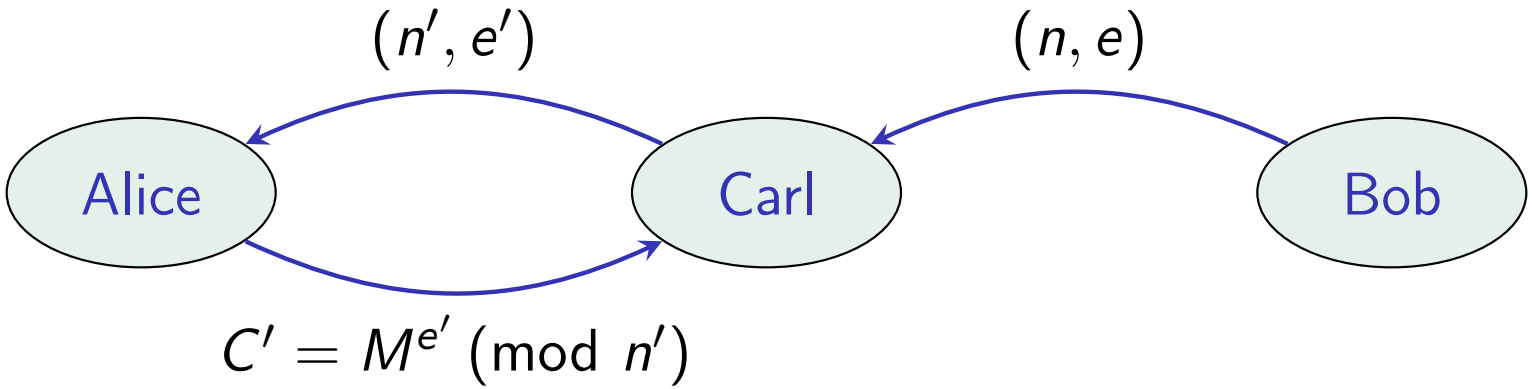


Attaque "man in the middle"

Alice chiffre le message M : $M^{e'} \pmod{n'}$

Carl envoie sa clé publique

Bob envoie sa clé publique



Alice envoie le message chiffré

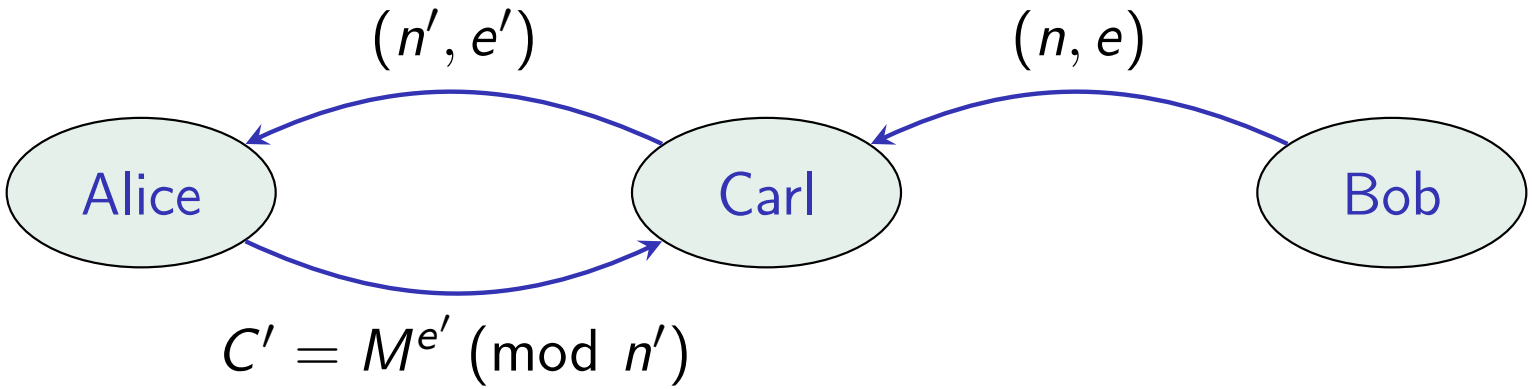
Carl intercepte le message

Attaque "man in the middle"

Alice chiffre le message M : $M^{e'} \pmod{n'}$

Carl envoie sa clé publique

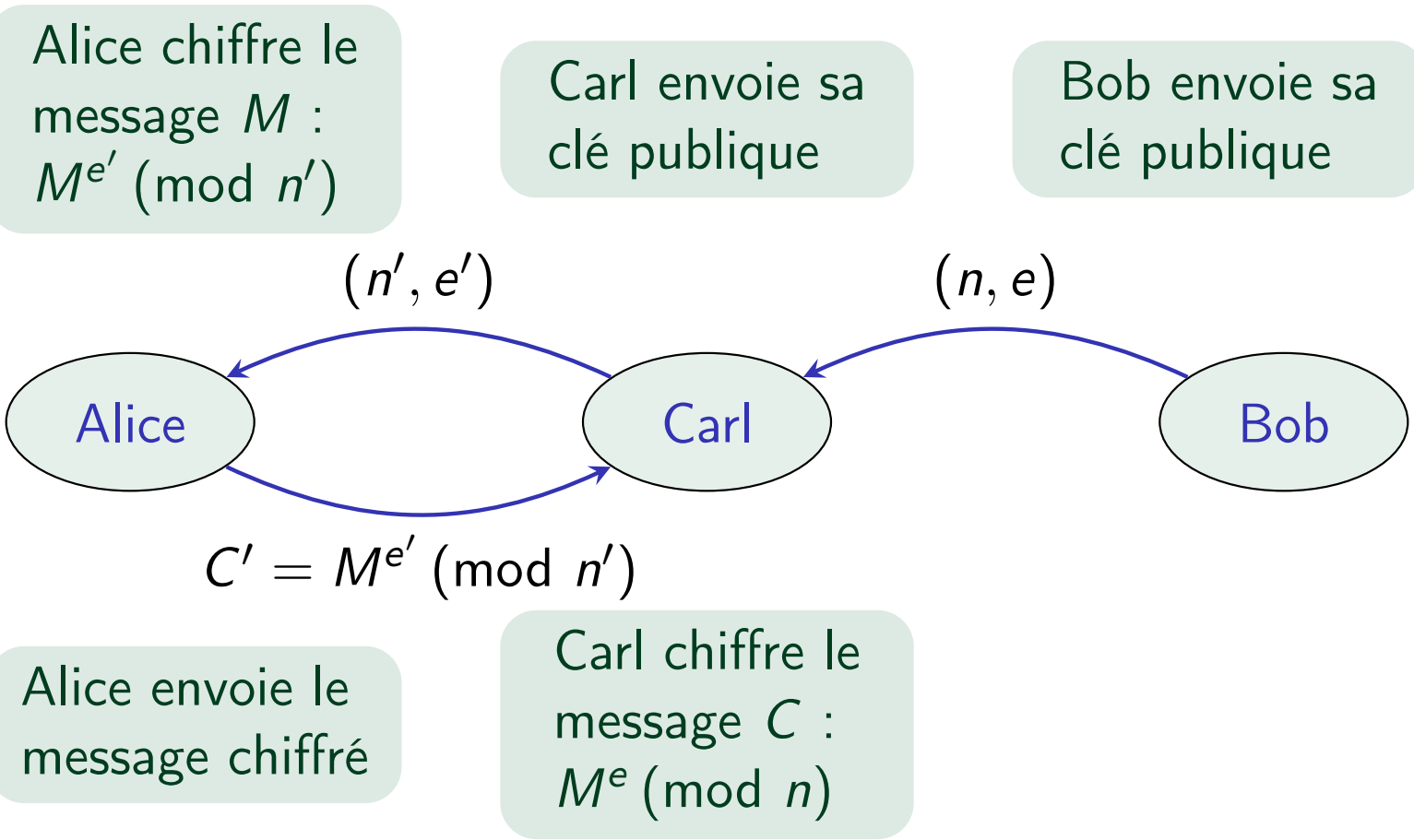
Bob envoie sa clé publique



Alice envoie le message chiffré

Carl déchiffre le message C' : $C'^{d'} \pmod{n'}$

Attaque "man in the middle"

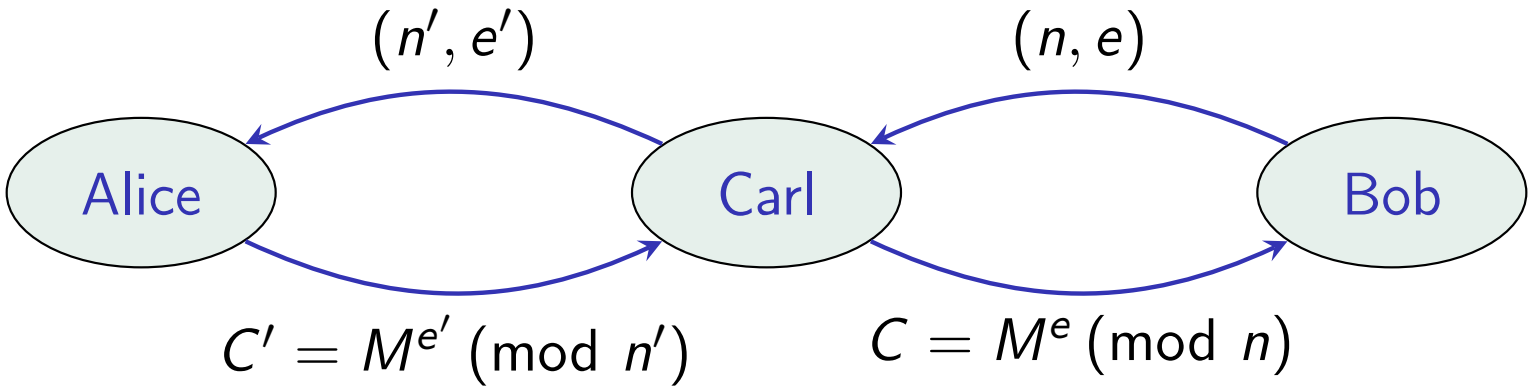


Attaque "man in the middle"

Alice chiffre le message M : $M^{e'} \pmod{n'}$

Carl envoie sa clé publique

Bob envoie sa clé publique



Alice envoie le message chiffré

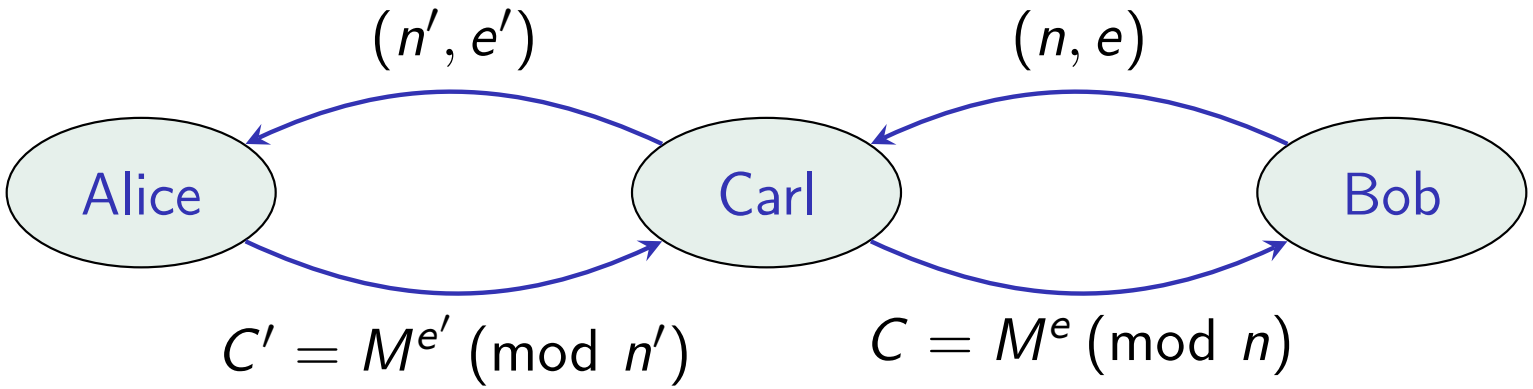
Carl envoie le message chiffré

Attaque "man in the middle"

Alice chiffre le message M : $M^{e'} \pmod{n'}$

Carl envoie sa clé publique

Bob envoie sa clé publique



Alice envoie le message chiffré

Carl envoie le message chiffré

Bob déchiffre le message C : $C^{d} \pmod{n}$

Considérations pratiques

- Jusqu'ici, pour simplifier l'exposé, on a considéré que le message M était un simple entier. Bien évidemment, ce n'est pas réaliste en pratique.
- Le message M est en réalité une suite d'octets : un texte codé en *ASCII* ou en *UTF-8*, un paquet réseau,...
- On pourrait avoir la tentation de chiffrer le message octet par octet (et le déchiffrer pareillement). Ce serait une très mauvaise idée car un attaquant n'aurait à effectuer au plus que 256 tests pour retrouver M ($X^e = M^e \pmod{n}, X \in \{0, \ldots, 255\}$). Il faut donc que le message M soit suffisamment long.
- Un autre problème réside dans le fait que, dans le cas d'un paquet réseau (un datagramme *IP* par exemple), les données à chiffrer sont formatées et le format est connu de tous. Cette connaissance peut être très utile à un attaquant potentiel. Une parade consisterait à brouiller les données avant de les chiffrer.

Considérations pratiques

- Jusqu'ici, pour simplifier l'exposé, on a considéré que le message M était un simple entier. Bien évidemment, ce n'est pas réaliste en pratique.
- Le message M est en réalité une suite d'octets : un texte codé en *ASCII* ou en *UTF-8*, un paquet réseau,...
- On pourrait avoir la tentation de chiffrer le message octet par octet (et le déchiffrer pareillement). Ce serait une très mauvaise idée car un attaquant n'aurait à effectuer au plus que 256 tests pour retrouver M ($X^e = M^e \pmod{n}, X \in \{0, \ldots, 255\}$). Il faut donc que le message M soit suffisamment long.
- Un autre problème réside dans le fait que, dans le cas d'un paquet réseau (un datagramme *IP* par exemple), les données à chiffrer sont formatées et le format est connu de tous. Cette connaissance peut être très utile à un attaquant potentiel. Une parade consisterait à brouiller les données avant de les chiffrer.

Considérations pratiques

- Jusqu'ici, pour simplifier l'exposé, on a considéré que le message M était un simple entier. Bien évidemment, ce n'est pas réaliste en pratique.
- Le message M est en réalité une suite d'octets : un texte codé en *ASCII* ou en *UTF-8*, un paquet réseau,...
- On pourrait avoir la tentation de chiffrer le message octet par octet (et le déchiffrer pareillement). Ce serait une très mauvaise idée car un attaquant n'aurait à effectuer au plus que 256 tests pour retrouver M ($X^e = M^e \pmod{n}, X \in \{0, \ldots, 255\}$). Il faut donc que le message M soit suffisamment long.
- Un autre problème réside dans le fait que, dans le cas d'un paquet réseau (un datagramme *IP* par exemple), les données à chiffrer sont formatées et le format est connu de tous. Cette connaissance peut être très utile à un attaquant potentiel. Une parade consisterait à brouiller les données avant de les chiffrer.

Considérations pratiques

- Jusqu'ici, pour simplifier l'exposé, on a considéré que le message M était un simple entier. Bien évidemment, ce n'est pas réaliste en pratique.
- Le message M est en réalité une suite d'octets : un texte codé en *ASCII* ou en *UTF-8*, un paquet réseau,...
- On pourrait avoir la tentation de chiffrer le message octet par octet (et le déchiffrer pareillement). Ce serait une très mauvaise idée car un attaquant n'aurait à effectuer au plus que 256 tests pour retrouver M ($X^e = M^e \pmod{n}, X \in \{0, \ldots, 255\}$). Il faut donc que le message M soit suffisamment long.
- Un autre problème réside dans le fait que, dans le cas d'un paquet réseau (un datagramme *IP* par exemple), les données à chiffrer sont formatées et le format est connu de tous. Cette connaissance peut être très utile à un attaquant potentiel. Une parade consisterait à brouiller les données avant de les chiffrer.

Le protocole *OAEP*

Pour parer les attaques (et d'autres également comme l'attaque par chronométrage) vues précédemment, un *schéma de remplissage* incluant des données aléatoires est nécessaire. Pour le système RSA, le plus utilisé est le protocole OAEP (*Optimal Asymmetric Encryption Padding*) créé par *Bellare* et *Rogaway* en 1994.

Les ingrédients de OAEP sont :

1) un entier $0 < k_0 < k$, où k est la longueur en bit de la clé publique n du système RSA, tel que 2^{k_0} soit suffisamment grand,
2) deux fonctions de hachage cryptographique G et H dont les hachés ont pour longueurs en bit respectives $k - k_0$ et k_0,
3) un générateur aléatoire pour engendrer des suites de bits de longueur k_0.

Le protocole *OAEP*

Pour parer les attaques (et d'autres également comme l'attaque par chronométrage) vues précédemment, un *schéma de remplissage* incluant des données aléatoires est nécessaire. Pour le système RSA, le plus utilisé est le protocole OAEP (*Optimal Asymmetric Encryption Padding*) créé par *Bellare* et *Rogaway* en 1994.

Les ingrédients de OAEP sont :

1) un entier $0 < k_0 < k$, où k est la longueur en bit de la clé publique n du système RSA, tel que 2^{k_0} soit suffisamment grand,
2) deux fonctions de hachage cryptographique G et H dont les hachés ont pour longueurs en bit respectives $k - k_0$ et k_0,
3) un générateur aléatoire pour engendrer des suites de bits de longueur k_0.

Le protocole *OAEP*

Pour parer les attaques (et d'autres également comme l'attaque par chronométrage) vues précédemment, un *schéma de remplissage* incluant des données aléatoires est nécessaire. Pour le système RSA, le plus utilisé est le protocole OAEP (*Optimal Asymmetric Encryption Padding*) créé par *Bellare* et *Rogaway* en 1994.

Les ingrédients de OAEP sont :

1) un entier $0 < k_0 < k$, où k est la longueur en bit de la clé publique n du système RSA, tel que 2^{k_0} soit suffisamment grand,
2) deux fonctions de hachage cryptographique G et H dont les hachés ont pour longueurs en bit respectives $k - k_0$ et k_0,
3) un générateur aléatoire pour engendrer des suites de bits de longueur k_0.

Le protocole *OAEP*

Pour parer les attaques (et d'autres également comme l'attaque par chronométrage) vues précédemment, un *schéma de remplissage* incluant des données aléatoires est nécessaire. Pour le système RSA, le plus utilisé est le protocole OAEP (*Optimal Asymmetric Encryption Padding*) créé par *Bellare* et *Rogaway* en 1994.

Les ingrédients de OAEP sont :

1) un entier $0 < k_0 < k$, où k est la longueur en bit de la clé publique n du système RSA, tel que 2^{k_0} soit suffisamment grand,
2) deux fonctions de hachage cryptographique G et H dont les hachés ont pour longueurs en bit respectives $k - k_0$ et k_0,
3) un générateur aléatoire pour engendrer des suites de bits de longueur k_0.

Le protocole *OAEP*

Pour parer les attaques (et d'autres également comme l'attaque par chronométrage) vues précédemment, un *schéma de remplissage* incluant des données aléatoires est nécessaire. Pour le système RSA, le plus utilisé est le protocole OAEP (*Optimal Asymmetric Encryption Padding*) créé par *Bellare* et *Rogaway* en 1994.

Les ingrédients de OAEP sont :

1) un entier $0 < k_0 < k$, où k est la longueur en bit de la clé publique n du système RSA, tel que 2^{k_0} soit suffisamment grand,
2) deux fonctions de hachage cryptographique G et H dont les hachés ont pour longueurs en bit respectives $k - k_0$ et k_0,
3) un générateur aléatoire pour engendrer des suites de bits de longueur k_0.

Le protocole *OAEP*

Pour encoder un message M de longueur en bit $m \leq n - k_0$:

1) on complète le message M par des bits à 0 : $M \underbrace{00\ldots0}_{n-k_0-m}$,
2) on génère une suite aléatoire r de k_0 bits,
3) on calcule $X = M00\ldots0 \oplus G(r)$ ($\oplus$ désigne ici l'opérateur bit à bit *ou exclusif*),
4) on calcule $Y = r \oplus H(X)$,
5) le message encodé est alors : $OAEP(M) = X \parallel Y$ ($\parallel$ désigne la concaténation).

Pour décoder le message $OAEP(M)$:

1) on récupère tout d'abord $r = Y \oplus H(X)$,
2) on obtient ensuite le message original $M00\ldots0 = X \oplus G(r)$ (le décodeur connaît la longueur m du message original).

Le protocole *OAEP*

Pour encoder un message M de longueur en bit $m \leq n - k_0$:

1) on complète le message M par des bits à 0 : $M \underbrace{00\ldots0}_{n-k_0-m}$,
2) on génère une suite aléatoire r de k_0 bits,
3) on calcule $X = M00\ldots0 \oplus G(r)$ ($\oplus$ désigne ici l'opérateur bit à bit *ou exclusif*),
4) on calcule $Y = r \oplus H(X)$,
5) le message encodé est alors : $OAEP(M) = X \parallel Y$ ($\parallel$ désigne la concaténation).

Pour décoder le message $OAEP(M)$:

1) on récupère tout d'abord $r = Y \oplus H(X)$,
2) on obtient ensuite le message original $M00\ldots0 = X \oplus G(r)$ (le décodeur connaît la longueur m du message original).

Le protocole *OAEP*

Pour encoder un message M de longueur en bit $m \leq n - k_0$:

1) on complète le message M par des bits à 0 : $M \underbrace{00\ldots0}_{n-k_0-m}$,
2) on génère une suite aléatoire r de k_0 bits,
3) on calcule $X = M00\ldots0 \oplus G(r)$ ($\oplus$ désigne ici l'opérateur bit à bit *ou exclusif*),
4) on calcule $Y = r \oplus H(X)$,
5) le message encodé est alors : $OAEP(M) = X \parallel Y$ ($\parallel$ désigne la concaténation).

Pour décoder le message $OAEP(M)$:

1) on récupère tout d'abord $r = Y \oplus H(X)$,
2) on obtient ensuite le message original $M00\ldots0 = X \oplus G(r)$ (le décodeur connaît la longueur m du message original).

Le protocole *OAEP*

Pour encoder un message M de longueur en bit $m \leq n - k_0$:

1) on complète le message M par des bits à 0 : $M \underbrace{00\ldots0}_{n-k_0-m}$,
2) on génère une suite aléatoire r de k_0 bits,
3) on calcule $X = M00\ldots0 \oplus G(r)$ ($\oplus$ désigne ici l'opérateur bit à bit *ou exclusif*),
4) on calcule $Y = r \oplus H(X)$,
5) le message encodé est alors : $OAEP(M) = X \parallel Y$ ($\parallel$ désigne la concaténation).

Pour décoder le message $OAEP(M)$:

1) on récupère tout d'abord $r = Y \oplus H(X)$,
2) on obtient ensuite le message original $M00\ldots0 = X \oplus G(r)$ (le décodeur connaît la longueur m du message original).

Le protocole *OAEP*

Pour encoder un message M de longueur en bit $m \leq n - k_0$:

1) on complète le message M par des bits à 0 : $M \underbrace{00\ldots0}_{n-k_0-m}$,
2) on génère une suite aléatoire r de k_0 bits,
3) on calcule $X = M00\ldots0 \oplus G(r)$ ($\oplus$ désigne ici l'opérateur bit à bit *ou exclusif*),
4) on calcule $Y = r \oplus H(X)$,
5) le message encodé est alors : $OAEP(M) = X \parallel Y$ ($\parallel$ désigne la concaténation).

Pour décoder le message $OAEP(M)$:

1) on récupère tout d'abord $r = Y \oplus H(X)$,
2) on obtient ensuite le message original $M00\ldots0 = X \oplus G(r)$ (le décodeur connaît la longueur m du message original).

Le protocole *OAEP*

Pour encoder un message M de longueur en bit $m \leq n - k_0$:

1) on complète le message M par des bits à 0 : $M \underbrace{00\ldots0}_{n-k_0-m}$,
2) on génère une suite aléatoire r de k_0 bits,
3) on calcule $X = M00\ldots0 \oplus G(r)$ ($\oplus$ désigne ici l'opérateur bit à bit *ou exclusif*),
4) on calcule $Y = r \oplus H(X)$,
5) le message encodé est alors : $OAEP(M) = X \parallel Y$ ($\parallel$ désigne la concaténation).

Pour décoder le message $OAEP(M)$:

1) on récupère tout d'abord $r = Y \oplus H(X)$,
2) on obtient ensuite le message original $M00\ldots0 = X \oplus G(r)$ (le décodeur connaît la longueur m du message original).

Le protocole *OAEP*

Pour encoder un message M de longueur en bit $m \leq n - k_0$:

1) on complète le message M par des bits à 0 : $M \underbrace{00\ldots0}_{n-k_0-m}$,
2) on génère une suite aléatoire r de k_0 bits,
3) on calcule $X = M00\ldots0 \oplus G(r)$ ($\oplus$ désigne ici l'opérateur bit à bit *ou exclusif*),
4) on calcule $Y = r \oplus H(X)$,
5) le message encodé est alors : $OAEP(M) = X \parallel Y$ ($\parallel$ désigne la concaténation).

Pour décoder le message $OAEP(M)$:

1) on récupère tout d'abord $r = Y \oplus H(X)$,
2) on obtient ensuite le message original $M00\ldots0 = X \oplus G(r)$ (le décodeur connaît la longueur m du message original).

Le protocole *OAEP*

Pour encoder un message M de longueur en bit $m \leq n - k_0$:

1) on complète le message M par des bits à 0 : $M \underbrace{00\ldots0}_{n-k_0-m}$,
2) on génère une suite aléatoire r de k_0 bits,
3) on calcule $X = M00\ldots0 \oplus G(r)$ ($\oplus$ désigne ici l'opérateur bit à bit *ou exclusif*),
4) on calcule $Y = r \oplus H(X)$,
5) le message encodé est alors : $OAEP(M) = X \parallel Y$ ($\parallel$ désigne la concaténation).

Pour décoder le message $OAEP(M)$:

1) on récupère tout d'abord $r = Y \oplus H(X)$,
2) on obtient ensuite le message original $M00\ldots0 = X \oplus G(r)$ (le décodeur connaît la longueur m du message original).

Le protocole *OAEP*

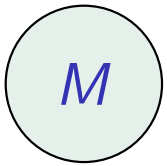
Pour encoder un message M de longueur en bit $m \leq n - k_0$:

1) on complète le message M par des bits à 0 : $M \underbrace{00\ldots0}_{n-k_0-m}$,
2) on génère une suite aléatoire r de k_0 bits,
3) on calcule $X = M00\ldots0 \oplus G(r)$ ($\oplus$ désigne ici l'opérateur bit à bit *ou exclusif*),
4) on calcule $Y = r \oplus H(X)$,
5) le message encodé est alors : $OAEP(M) = X \parallel Y$ ($\parallel$ désigne la concaténation).

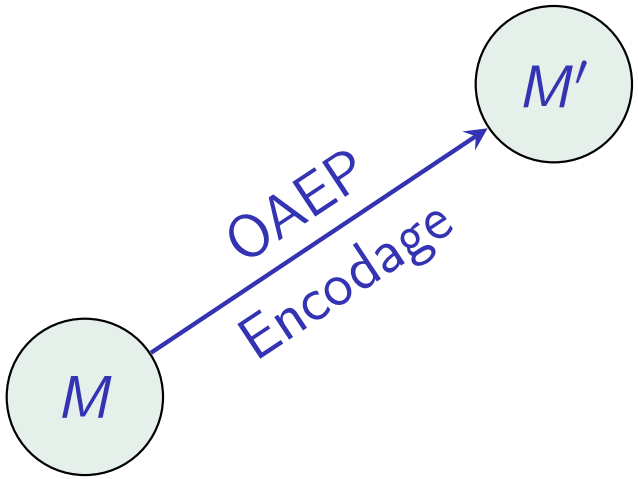
Pour décoder le message $OAEP(M)$:

1) on récupère tout d'abord $r = Y \oplus H(X)$,
2) on obtient ensuite le message original $M00\ldots0 = X \oplus G(r)$ (le décodeur connaît la longueur m du message original).

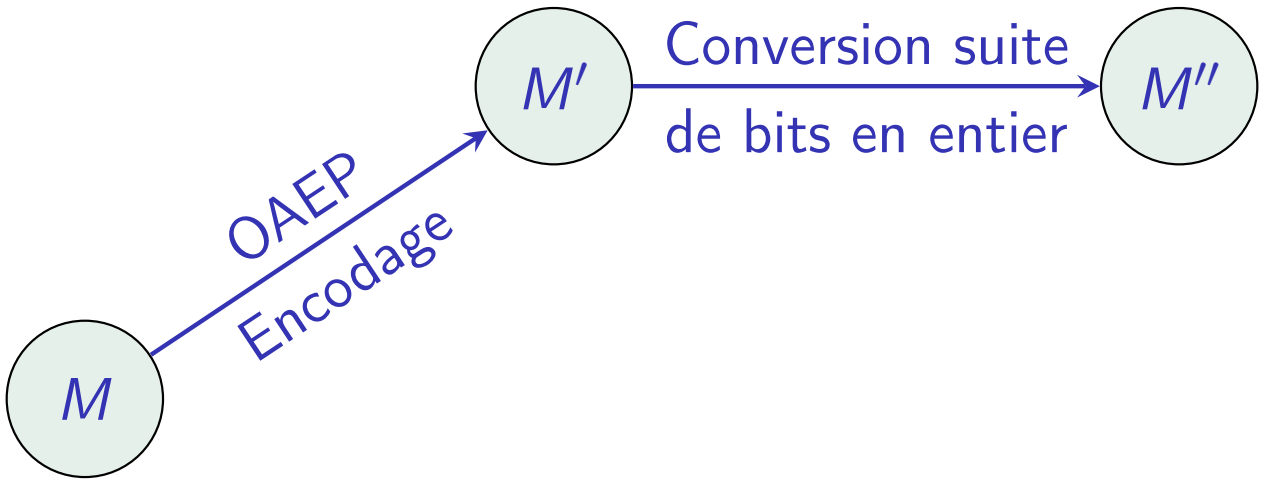
RSA-OAEP : Schéma d'ensemble



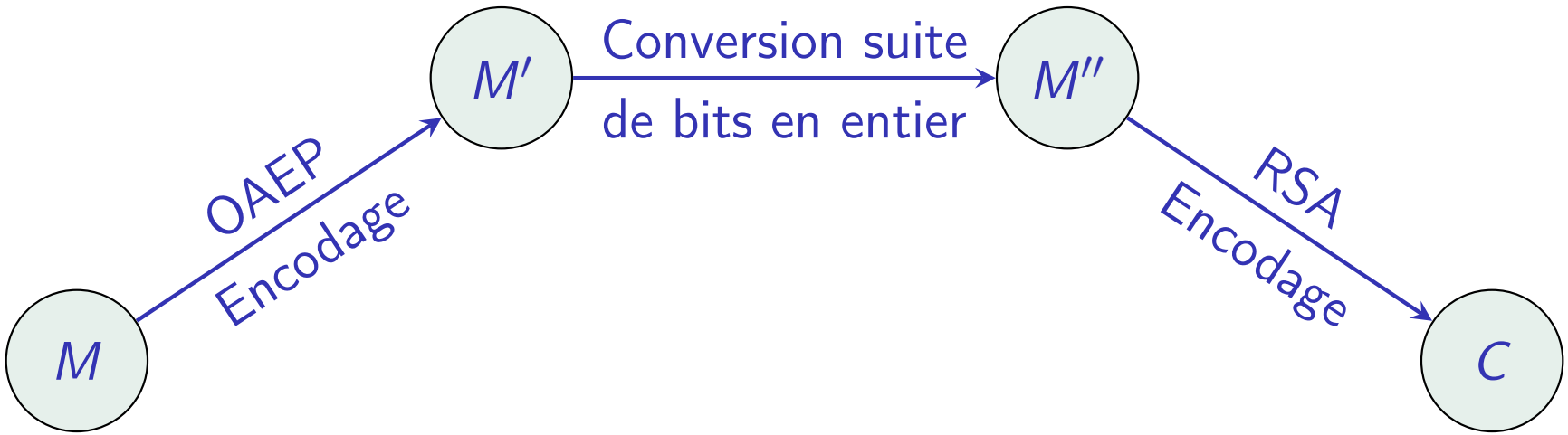
RSA-OAEP : Schéma d'ensemble



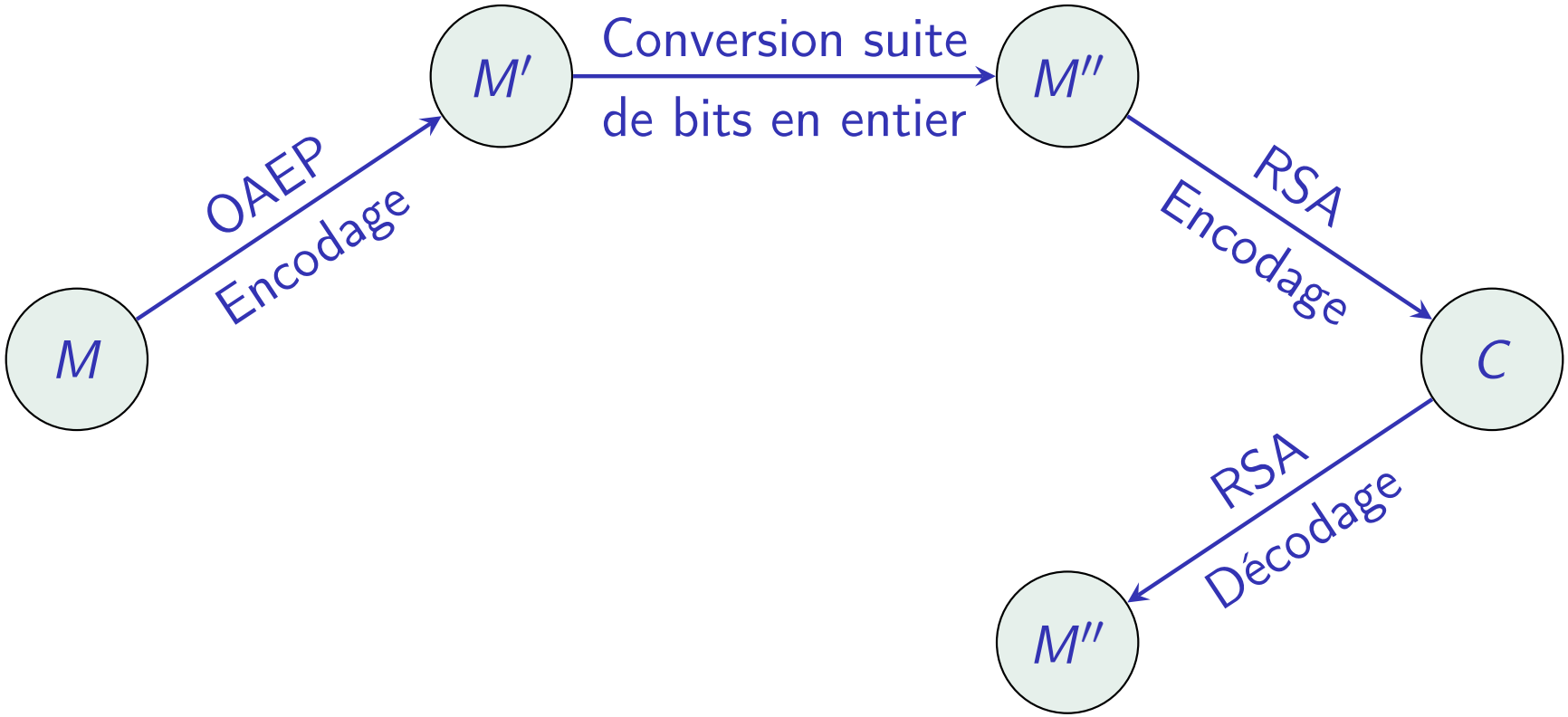
RSA-OAEP : Schéma d'ensemble



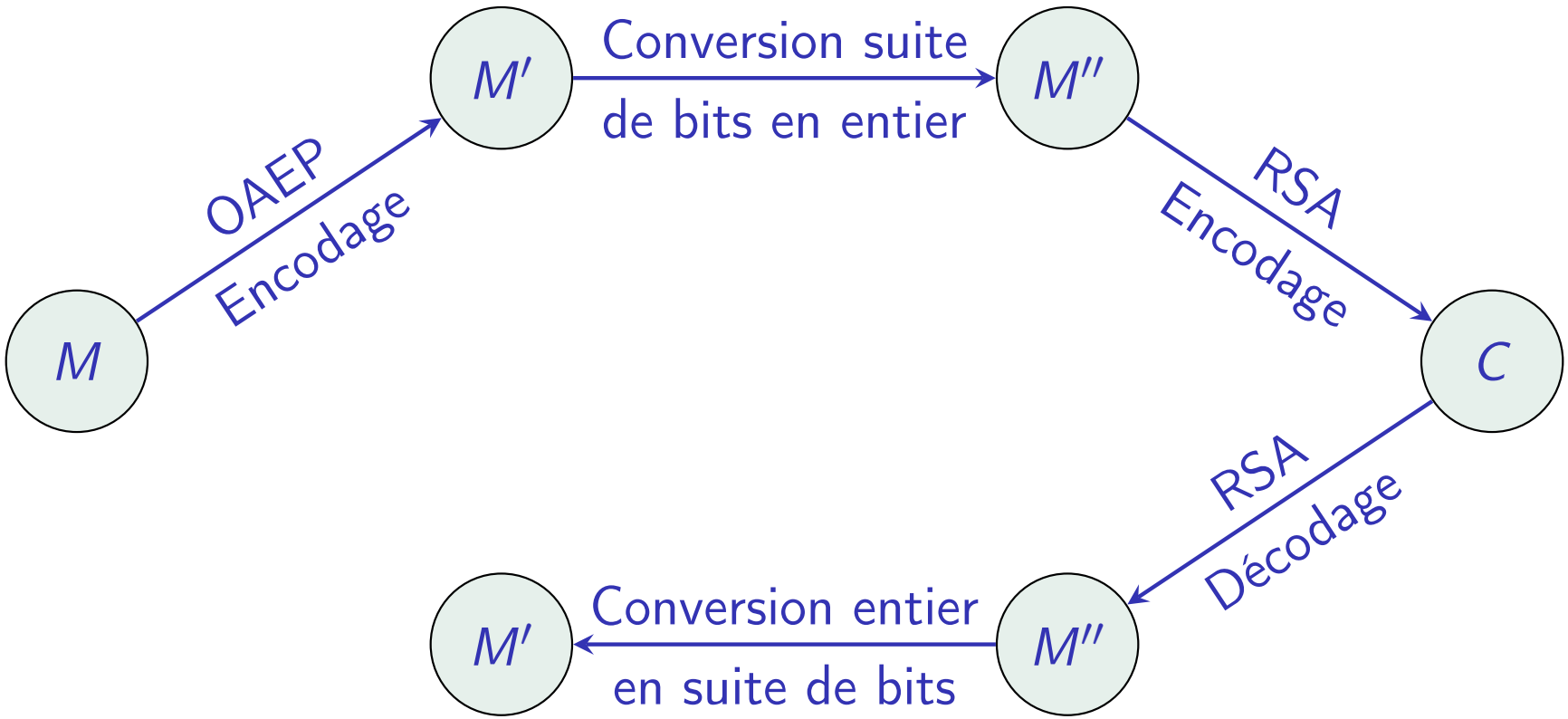
RSA-OAEP : Schéma d'ensemble



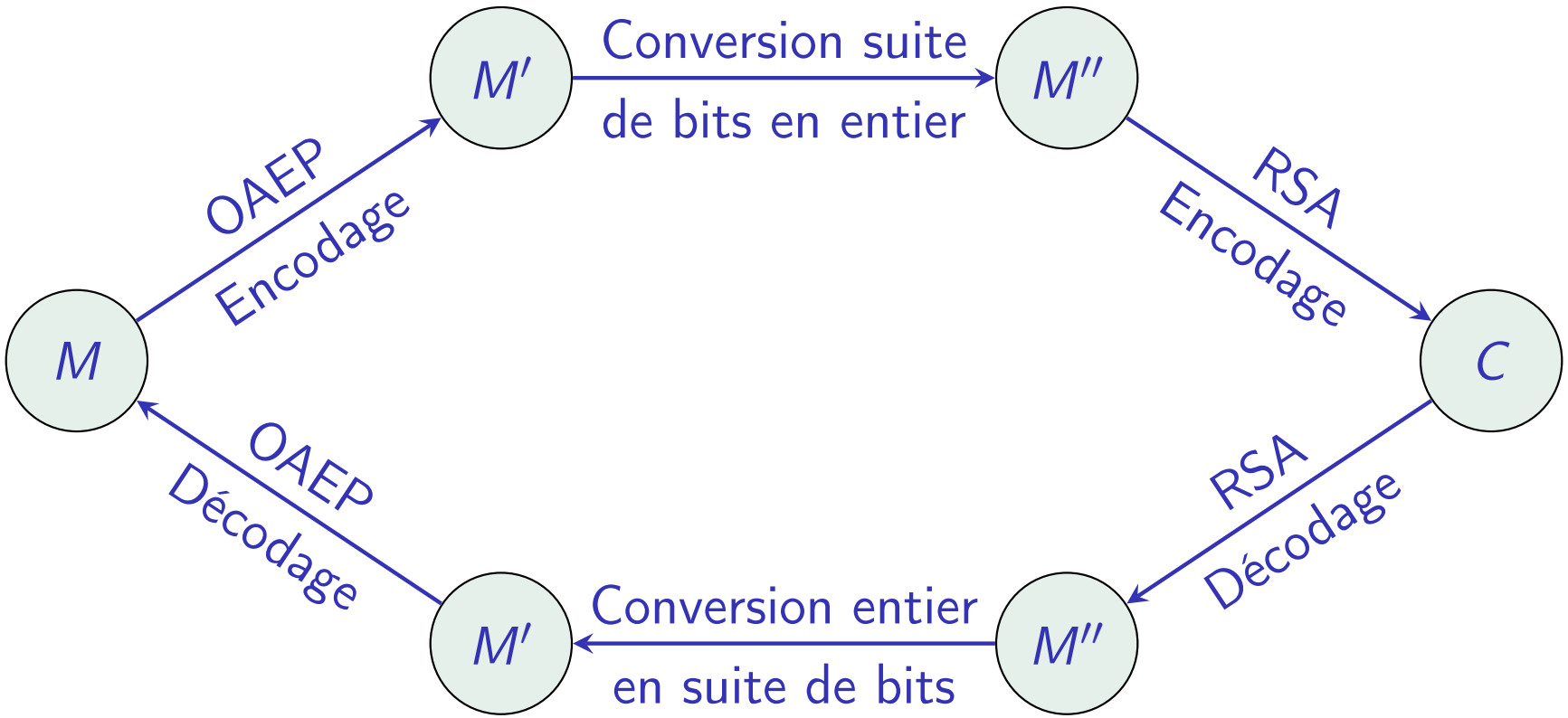
RSA-OAEP : Schéma d'ensemble



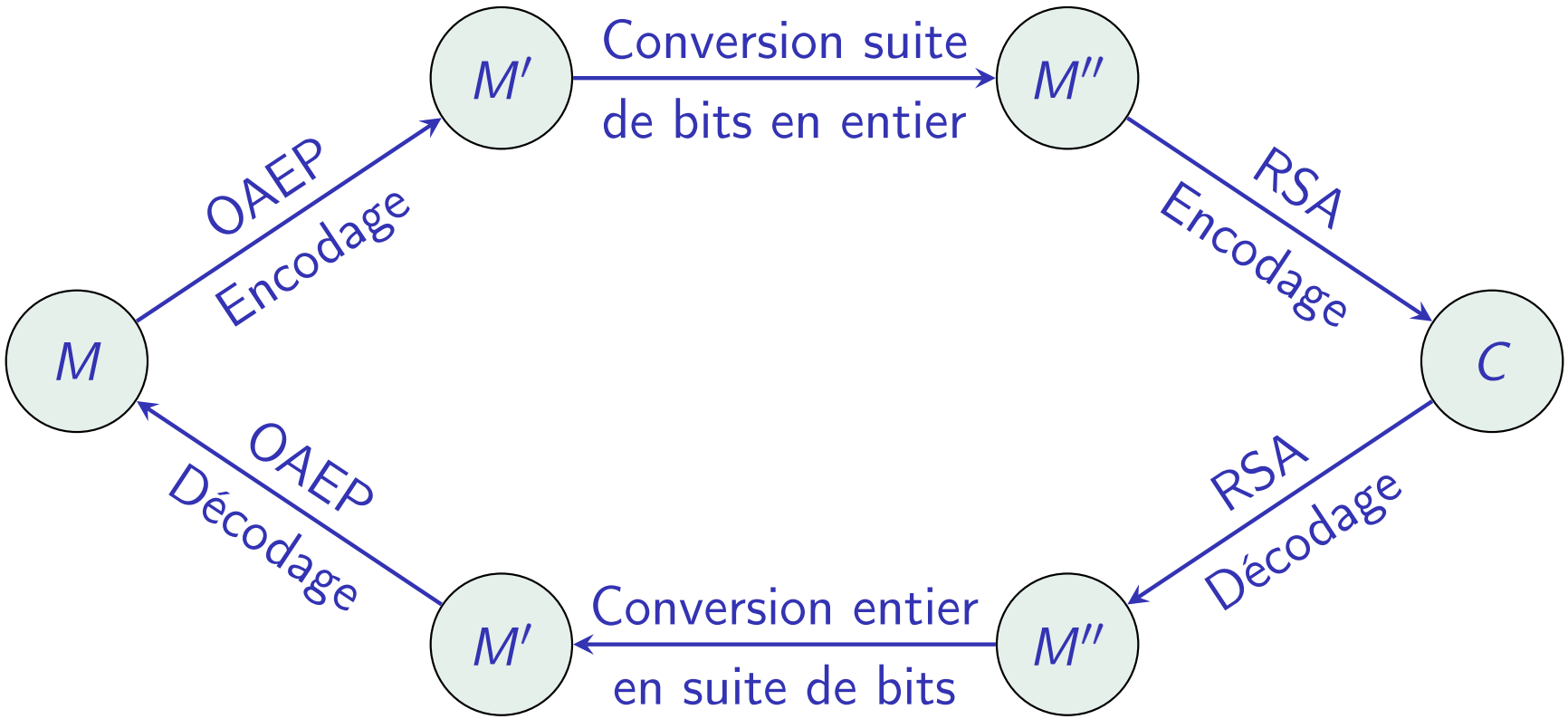
RSA-OAEP : Schéma d'ensemble



RSA-OAEP : Schéma d'ensemble



RSA-OAEP : Schéma d'ensemble



RSA-OAEP a été normalisé au sein de l'IETF : ▸ RFC-3447 (PKCS #1 v2.1).

Historique

- On considère généralement que la cryptographie moderne est née en 1976 avec la parution de l'article de *Diffie* et *Hellman* intitulé : "New directions in cryptography".
- Le problème que ces deux auteurs résolvent, et que l'on pensait insoluble jusqu'alors, est le partage d'un secret entre deux entités à travers un canal **non sécurisé**.
- Plus précisément, supposons que nos amis *Alice* et *Bob* veuillent échanger des données confidentielles à travers un canal non sécurisé, comme l'internet public par exemple. Ils devront au préalable se mettre d'accord *publiquement* sur un protocole de communication permettant un tel échange.
- Ce qui est étonnant, c'est qu'un tel protocole existe et en outre soit simple. Il est fondé sur l'**exponentielle discrète**.

Historique

- On considère généralement que la cryptographie moderne est née en 1976 avec la parution de l'article de *Diffie* et *Hellman* intitulé : "New directions in cryptography".
- Le problème que ces deux auteurs résolvent, et que l'on pensait insoluble jusqu'alors, est le partage d'un secret entre deux entités à travers un canal **non sécurisé**.
- Plus précisément, supposons que nos amis *Alice* et *Bob* veuillent échanger des données confidentielles à travers un canal non sécurisé, comme l'internet public par exemple. Ils devront au préalable se mettre d'accord *publiquement* sur un protocole de communication permettant un tel échange.
- Ce qui est étonnant, c'est qu'un tel protocole existe et en outre soit simple. Il est fondé sur l'**exponentielle discrète**.

Historique

- On considère généralement que la cryptographie moderne est née en 1976 avec la parution de l'article de *Diffie* et *Hellman* intitulé : "New directions in cryptography".
- Le problème que ces deux auteurs résolvent, et que l'on pensait insoluble jusqu'alors, est le partage d'un secret entre deux entités à travers un canal **non sécurisé**.
- Plus précisément, supposons que nos amis *Alice* et *Bob* veuillent échanger des données confidentielles à travers un canal non sécurisé, comme l'internet public par exemple. Ils devront au préalable se mettre d'accord *publiquement* sur un protocole de communication permettant un tel échange.
- Ce qui est étonnant, c'est qu'un tel protocole existe et en outre soit simple. Il est fondé sur l'**exponentielle discrète**.

Historique

- On considère généralement que la cryptographie moderne est née en 1976 avec la parution de l'article de *Diffie* et *Hellman* intitulé : "New directions in cryptography".
- Le problème que ces deux auteurs résolvent, et que l'on pensait insoluble jusqu'alors, est le partage d'un secret entre deux entités à travers un canal **non sécurisé**.
- Plus précisément, supposons que nos amis *Alice* et *Bob* veuillent échanger des données confidentielles à travers un canal non sécurisé, comme l'internet public par exemple. Ils devront au préalable se mettre d'accord *publiquement* sur un protocole de communication permettant un tel échange.
- Ce qui est étonnant, c'est qu'un tel protocole existe et en outre soit simple. Il est fondé sur l'**exponentielle discrète**.

Schéma du protocole

Schéma du protocole

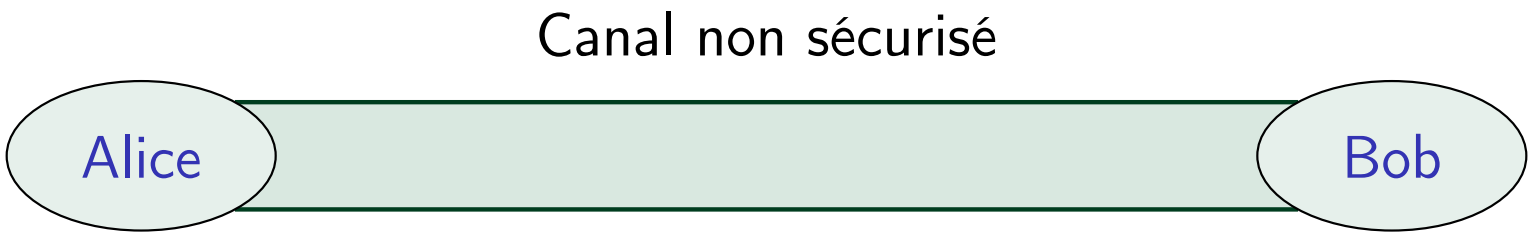


Schéma du protocole

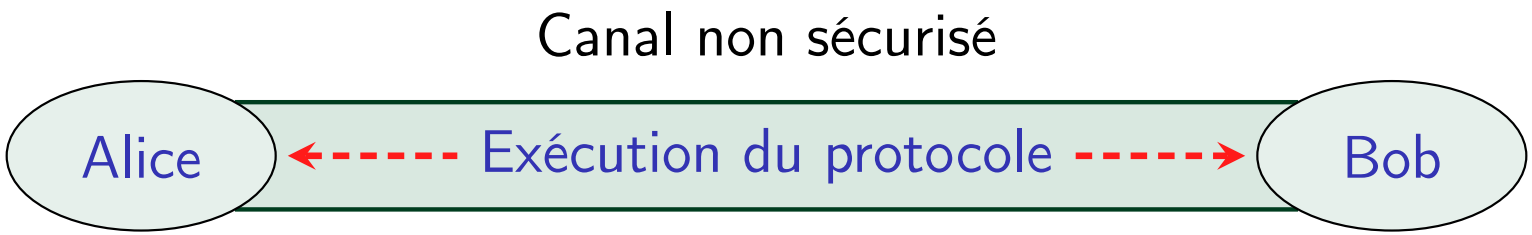
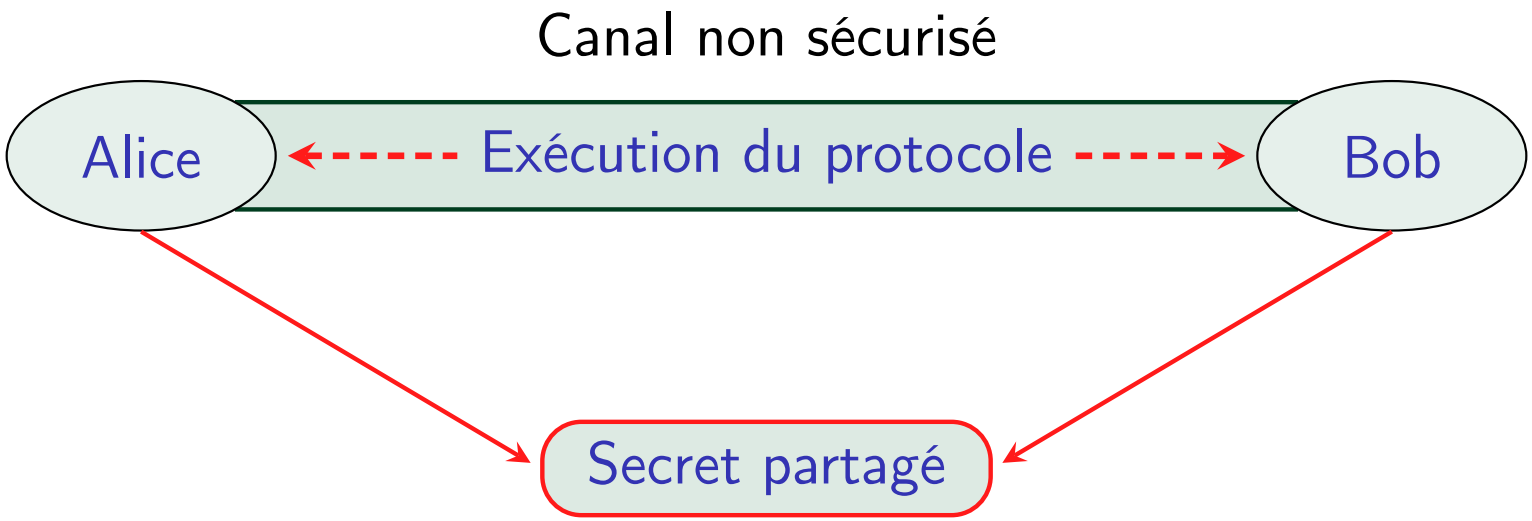


Schéma du protocole



La notion de groupe

Nous aurons besoin d'une nouvelle structure mathématique : le groupe.

Définition (Groupe)

Soient G un ensemble et $\perp$ une application de $G \times G \to G$. Alors, si $\perp(g,h)$ est noté $g \perp h$, le couple $(G,\perp)$ est un groupe si :

(i) $\forall g,g',g'' \in G,\ (g \perp g') \perp g'' = g \perp (g' \perp g'')$ *(associativité),*

(ii) $\exists e \in G\, \forall g \in G,\ g \perp e = e \perp g = g$ *(élément neutre),*

(iii) $\forall g \in G\, \exists g^{\perp},\ g \perp g^{\perp} = g^{\perp} \perp g = e$ *(élément symétrique).*

Si de plus $(G,\perp)$ satisfait :

(iv) $\forall g,g' \in G,\ g \perp g' = g' \perp g$ *(commutativité),*

$(G,\perp)$ est dit commutatif ou abélien.

Note : tous les groupes que nous rencontrerons seront commutatifs.

La notion de groupe

Nous aurons besoin d'une nouvelle structure mathématique : le groupe.

Définition (Groupe)

Soient G un ensemble et $\perp$ une application de $G \times G \to G$. Alors, si $\perp(g,h)$ est noté $g \perp h$, le couple $(G,\perp)$ est un groupe si :

(i) $\forall g,g',g'' \in G,\ (g \perp g') \perp g'' = g \perp (g' \perp g'')$ *(associativité),*

(ii) $\exists e \in G\, \forall g \in G,\ g \perp e = e \perp g = g$ *(élément neutre),*

(iii) $\forall g \in G\, \exists g^{\perp},\ g \perp g^{\perp} = g^{\perp} \perp g = e$ *(élément symétrique).*

Si de plus $(G,\perp)$ satisfait :

(iv) $\forall g,g' \in G,\ g \perp g' = g' \perp g$ *(commutativité),*

$(G,\perp)$ est dit commutatif ou abélien.

Note : tous les groupes que nous rencontrerons seront commutatifs.

La notion de groupe

Nous aurons besoin d'une nouvelle structure mathématique : le groupe.

Définition (Groupe)

Soient G un ensemble et $\perp$ une application de $G \times G \to G$. Alors, si $\perp(g,h)$ est noté $g \perp h$, le couple $(G,\perp)$ est un groupe si :

(i) $\forall g,g',g'' \in G,\ (g \perp g') \perp g'' = g \perp (g' \perp g'')$ *(associativité),*

(ii) $\exists e \in G\, \forall g \in G,\ g \perp e = e \perp g = g$ *(élément neutre),*

(iii) $\forall g \in G\, \exists g^{\perp},\ g \perp g^{\perp} = g^{\perp} \perp g = e$ *(élément symétrique).*

Si de plus $(G,\perp)$ satisfait :

(iv) $\forall g,g' \in G,\ g \perp g' = g' \perp g$ *(commutativité),*

$(G,\perp)$ est dit commutatif ou abélien.

Note : tous les groupes que nous rencontrerons seront commutatifs.

La notion de groupe

Nous aurons besoin d'une nouvelle structure mathématique : le groupe.

Définition (Groupe)

Soient G un ensemble et $\perp$ une application de $G \times G \to G$. Alors, si $\perp(g,h)$ est noté $g \perp h$, le couple $(G,\perp)$ est un groupe si :

(i) $\forall g,g',g'' \in G,\ (g \perp g') \perp g'' = g \perp (g' \perp g'')$ *(associativité),*

(ii) $\exists e \in G\, \forall g \in G,\ g \perp e = e \perp g = g$ *(élément neutre),*

(iii) $\forall g \in G\, \exists g^{\perp},\ g \perp g^{\perp} = g^{\perp} \perp g = e$ *(élément symétrique).*

Si de plus $(G,\perp)$ satisfait :

(iv) $\forall g,g' \in G,\ g \perp g' = g' \perp g$ *(commutativité),*

$(G,\perp)$ est dit commutatif ou abélien.

Note : tous les groupes que nous rencontrerons seront commutatifs.

La notion de groupe

Nous aurons besoin d'une nouvelle structure mathématique : le groupe.

Définition (Groupe)

Soient G un ensemble et $\perp$ une application de $G \times G \to G$. Alors, si $\perp(g, h)$ est noté $g \perp h$, le couple $(G, \perp)$ est un groupe si :

(i) $\forall g, g', g'' \in G,\ (g \perp g') \perp g'' = g \perp (g' \perp g'')$ *(associativité),*

(ii) $\exists e \in G\, \forall g \in G,\ g \perp e = e \perp g = g$ *(élément neutre),*

(iii) $\forall g \in G\, \exists g^{\perp},\ g \perp g^{\perp} = g^{\perp} \perp g = e$ *(élément symétrique).*

Si de plus $(G, \perp)$ satisfait :

(iv) $\forall g, g' \in G,\ g \perp g' = g' \perp g$ *(commutativité),*

$(G, \perp)$ est dit commutatif ou abélien.

Note : tous les groupes que nous rencontrerons seront commutatifs.

La notion de groupe

Nous aurons besoin d'une nouvelle structure mathématique : le groupe.

Définition (Groupe)

Soient G un ensemble et $\perp$ une application de $G \times G \to G$. Alors, si $\perp(g, h)$ est noté $g \perp h$, le couple $(G, \perp)$ est un groupe si :

(i) $\forall g, g', g'' \in G,\ (g \perp g') \perp g'' = g \perp (g' \perp g'')$ *(associativité),*

(ii) $\exists e \in G\, \forall g \in G,\ g \perp e = e \perp g = g$ *(élément neutre),*

(iii) $\forall g \in G\, \exists g^{\perp},\ g \perp g^{\perp} = g^{\perp} \perp g = e$ *(élément symétrique).*

Si de plus $(G, \perp)$ satisfait :

(iv) $\forall g, g' \in G,\ g \perp g' = g' \perp g$ *(commutativité),*

$(G, \perp)$ est dit commutatif ou abélien.

Note : tous les groupes que nous rencontrerons seront commutatifs.

La notion de groupe

Nous aurons besoin d'une nouvelle structure mathématique : le groupe.

Définition (Groupe)

Soient G un ensemble et $\perp$ une application de $G \times G \to G$. Alors, si $\perp(g, h)$ est noté $g \perp h$, le couple $(G, \perp)$ est un groupe si :

(i) $\forall g, g', g'' \in G,\ (g \perp g') \perp g'' = g \perp (g' \perp g'')$ *(associativité),*

(ii) $\exists e \in G\, \forall g \in G,\ g \perp e = e \perp g = g$ *(élément neutre),*

(iii) $\forall g \in G\, \exists g^{\perp},\ g \perp g^{\perp} = g^{\perp} \perp g = e$ *(élément symétrique).*

Si de plus $(G, \perp)$ satisfait :

(iv) $\forall g, g' \in G,\ g \perp g' = g' \perp g$ *(commutativité),*

$(G, \perp)$ est dit commutatif ou abélien.

Note : tous les groupes que nous rencontrerons seront commutatifs.

La notion de groupe

En pratique, sauf à des fins pédagogiques, on n'utilise pas la notation $(G, \perp, e, g^{\perp})$ mais plutôt :

- la notation **additive** $(G, +, 0, -g)$, généralement réservée aux groupes commutatifs. En particulier, la *puissance* d'un élément g s'écrit : $ng = \underbrace{g + \cdots + g}_{n}$.
- la notation **multiplicative** $(G, \cdot, 1, g^{-1})$. Ici, la puissance de g s'écrit : $g^n = \underbrace{g \times \cdots \times g}_{n}$.

Nous utiliserons dans un premier temps la notation multiplicative.

Définition (Ordre d'un élément)

*Soit $(G, \cdot)$ un groupe et soit $g \in G$. Si pour tout entier $n > 0$, $g^n \neq 1$, on dit que g est d'***ordre infini***. Sinon, on appelle* **ordre** *de g le plus petit entier $n > 0$ tel que $g^n = 1$.*

La notion de groupe

En pratique, sauf à des fins pédagogiques, on n'utilise pas la notation $(G, \perp, e, g^{\perp})$ mais plutôt :

- la notation **additive** $(G, +, 0, -g)$, généralement réservée aux groupes commutatifs. En particulier, la *puissance* d'un élément g s'écrit : $ng = \underbrace{g + \cdots + g}_{n}$.
- la notation **multiplicative** $(G, \cdot, 1, g^{-1})$. Ici, la puissance de g s'écrit : $g^n = \underbrace{g \times \cdots \times g}_{n}$.

Nous utiliserons dans un premier temps la notation multiplicative.

Définition (Ordre d'un élément)

*Soit $(G, \cdot)$ un groupe et soit $g \in G$. Si pour tout entier $n > 0$, $g^n \neq 1$, on dit que g est d'***ordre infini***. Sinon, on appelle* **ordre** *de g le plus petit entier $n > 0$ tel que $g^n = 1$.*

La notion de groupe

En pratique, sauf à des fins pédagogiques, on n'utilise pas la notation $(G, \perp, e, g^{\perp})$ mais plutôt :

- la notation **additive** $(G, +, 0, -g)$, généralement réservée aux groupes commutatifs. En particulier, la *puissance* d'un élément g s'écrit : $ng = \underbrace{g + \cdots + g}_{n}$.
- la notation **multiplicative** $(G, \cdot, 1, g^{-1})$. Ici, la puissance de g s'écrit : $g^n = \underbrace{g \times \cdots \times g}_{n}$.

Nous utiliserons dans un premier temps la notation multiplicative.

Définition (Ordre d'un élément)

*Soit $(G, \cdot)$ un groupe et soit $g \in G$. Si pour tout entier $n > 0$, $g^n \neq 1$, on dit que g est d'***ordre infini***. Sinon, on appelle* **ordre** *de g le plus petit entier $n > 0$ tel que $g^n = 1$.*

La notion de groupe

En pratique, sauf à des fins pédagogiques, on n'utilise pas la notation $(G, \perp, e, g^{\perp})$ mais plutôt :

- la notation **additive** $(G, +, 0, -g)$, généralement réservée aux groupes commutatifs. En particulier, la *puissance* d'un élément g s'écrit : $ng = \underbrace{g + \cdots + g}_{n}$.
- la notation **multiplicative** $(G, \cdot, 1, g^{-1})$. Ici, la puissance de g s'écrit : $g^n = \underbrace{g \times \cdots \times g}_{n}$.

Nous utiliserons dans un premier temps la notation multiplicative.

Définition (Ordre d'un élément)

*Soit $(G, \cdot)$ un groupe et soit $g \in G$. Si pour tout entier $n > 0$, $g^n \neq 1$, on dit que g est d'***ordre infini***. Sinon, on appelle* **ordre** *de g le plus petit entier $n > 0$ tel que $g^n = 1$.*

La notion de groupe

En pratique, sauf à des fins pédagogiques, on n'utilise pas la notation $(G, \perp, e, g^{\perp})$ mais plutôt :

- la notation **additive** $(G, +, 0, -g)$, généralement réservée aux groupes commutatifs. En particulier, la *puissance* d'un élément g s'écrit : $ng = \underbrace{g + \cdots + g}_{n}$.
- la notation **multiplicative** $(G, \cdot, 1, g^{-1})$. Ici, la puissance de g s'écrit : $g^n = \underbrace{g \times \cdots \times g}_{n}$.

Nous utiliserons dans un premier temps la notation multiplicative.

Définition (Ordre d'un élément)

*Soit $(G, \cdot)$ un groupe et soit $g \in G$. Si pour tout entier $n > 0$, $g^n \neq 1$, on dit que g est d'***ordre infini***. Sinon, on appelle* **ordre** *de g le plus petit entier $n > 0$ tel que $g^n = 1$.*

Exemples de groupes

- Le groupe additif des entiers relatifs $(\mathbb{Z}, +)$.
- Le groupe additif des réels $(\mathbb{R}, +)$.
- Le groupe multiplicatif des réels $(\mathbb{R}^*, \cdot)$.
- Le groupe $(\mathfrak{S}_n, \circ)$ des permutations de l'ensemble $\{1, \ldots, n\}$. C'est un groupe non commutatif et fini $(\#(G) = n!)$.
- En géométrie ou en cristallographie, les groupes de symétries.
- Le groupe fini $(\mathbb{Z}/n\mathbb{Z}, +) = \{\overline{0}, \ldots, \overline{n-1}\}$ des entiers modulo n où $\overline{k} = \{j \in \mathbb{Z} \mid j \equiv k \pmod{n}\}$, $k = 0, \ldots, n-1$. L'addition est définie ainsi : $\overline{k} + \overline{l} = \overline{k+l}$.
- Le groupe fini $((\mathbb{Z}/p\mathbb{Z})^*, \cdot) = \{\overline{1}, \ldots, \overline{p-1}\}$ des entiers modulo p et $\neq \overline{0}$ avec p **premier**. La multiplication est définie ainsi : $\overline{k} \cdot \overline{l} = \overline{kl}$. Attention : si p n'est pas premier, ce n'est pas un groupe. Par exemple si $p = 4$, $\overline{2}$ n'a pas d'élément symétrique car $\overline{2} \cdot \overline{2} = \overline{4} = \overline{0}$.
- Et bien d'autres encore...

Exemples de groupes

- Le groupe additif des entiers relatifs $(\mathbb{Z}, +)$.
- Le groupe additif des réels $(\mathbb{R}, +)$.
- Le groupe multiplicatif des réels $(\mathbb{R}^*, \cdot)$.
- Le groupe $(\mathfrak{S}_n, \circ)$ des permutations de l'ensemble $\{1, \ldots, n\}$. C'est un groupe non commutatif et fini $(\#(G) = n!)$.
- En géométrie ou en cristallographie, les groupes de symétries.
- Le groupe fini $(\mathbb{Z}/n\mathbb{Z}, +) = \{\overline{0}, \ldots, \overline{n-1}\}$ des entiers modulo n où $\overline{k} = \{j \in \mathbb{Z} \mid j \equiv k \pmod{n}\}$, $k = 0, \ldots, n-1$. L'addition est définie ainsi : $\overline{k} + \overline{l} = \overline{k+l}$.
- Le groupe fini $((\mathbb{Z}/p\mathbb{Z})^*, \cdot) = \{\overline{1}, \ldots, \overline{p-1}\}$ des entiers modulo p et $\neq \overline{0}$ avec p **premier**. La multiplication est définie ainsi : $\overline{k} \cdot \overline{l} = \overline{kl}$. Attention : si p n'est pas premier, ce n'est pas un groupe. Par exemple si $p = 4$, $\overline{2}$ n'a pas d'élément symétrique car $\overline{2} \cdot \overline{2} = \overline{4} = \overline{0}$.
- Et bien d'autres encore...

Exemples de groupes

- Le groupe additif des entiers relatifs $(\mathbb{Z}, +)$.
- Le groupe additif des réels $(\mathbb{R}, +)$.
- Le groupe multiplicatif des réels $(\mathbb{R}^*, \cdot)$.
- Le groupe $(\mathfrak{S}_n, \circ)$ des permutations de l'ensemble $\{1, \ldots, n\}$. C'est un groupe non commutatif et fini $(\#(G) = n!)$.
- En géométrie ou en cristallographie, les groupes de symétries.
- Le groupe fini $(\mathbb{Z}/n\mathbb{Z}, +) = \{\overline{0}, \ldots, \overline{n-1}\}$ des entiers modulo n où $\overline{k} = \{j \in \mathbb{Z} \mid j \equiv k \pmod{n}\}$, $k = 0, \ldots, n-1$. L'addition est définie ainsi : $\overline{k} + \overline{l} = \overline{k+l}$.
- Le groupe fini $((\mathbb{Z}/p\mathbb{Z})^*, \cdot) = \{\overline{1}, \ldots, \overline{p-1}\}$ des entiers modulo p et $\neq \overline{0}$ avec p **premier**. La multiplication est définie ainsi : $\overline{k} \cdot \overline{l} = \overline{kl}$. Attention : si p n'est pas premier, ce n'est pas un groupe. Par exemple si $p = 4$, $\overline{2}$ n'a pas d'élément symétrique car $\overline{2} \cdot \overline{2} = \overline{4} = \overline{0}$.
- Et bien d'autres encore...

Exemples de groupes

- Le groupe additif des entiers relatifs $(\mathbb{Z}, +)$.
- Le groupe additif des réels $(\mathbb{R}, +)$.
- Le groupe multiplicatif des réels $(\mathbb{R}^*, \cdot)$.
- Le groupe $(\mathfrak{S}_n, \circ)$ des permutations de l'ensemble $\{1, \ldots, n\}$. C'est un groupe non commutatif et fini $(\#(G) = n!)$.
- En géométrie ou en cristallographie, les groupes de symétries.
- Le groupe fini $(\mathbb{Z}/n\mathbb{Z}, +) = \{\overline{0}, \ldots, \overline{n-1}\}$ des entiers modulo n où $\overline{k} = \{j \in \mathbb{Z} \mid j \equiv k \pmod{n}\}$, $k = 0, \ldots, n-1$. L'addition est définie ainsi : $\overline{k} + \overline{l} = \overline{k+l}$.
- Le groupe fini $((\mathbb{Z}/p\mathbb{Z})^*, \cdot) = \{\overline{1}, \ldots, \overline{p-1}\}$ des entiers modulo p et $\neq \overline{0}$ avec p **premier**. La multiplication est définie ainsi : $\overline{k} \cdot \overline{l} = \overline{kl}$. Attention : si p n'est pas premier, ce n'est pas un groupe. Par exemple si $p = 4$, $\overline{2}$ n'a pas d'élément symétrique car $\overline{2} \cdot \overline{2} = \overline{4} = \overline{0}$.
- Et bien d'autres encore...

Exemples de groupes

- Le groupe additif des entiers relatifs $(\mathbb{Z}, +)$.
- Le groupe additif des réels $(\mathbb{R}, +)$.
- Le groupe multiplicatif des réels $(\mathbb{R}^*, \cdot)$.
- Le groupe $(\mathfrak{S}_n, \circ)$ des permutations de l'ensemble $\{1, \ldots, n\}$. C'est un groupe non commutatif et fini $(\#(G) = n!)$.
- En géométrie ou en cristallographie, les groupes de symétries.
- Le groupe fini $(\mathbb{Z}/n\mathbb{Z}, +) = \{\overline{0}, \ldots, \overline{n-1}\}$ des entiers modulo n où $\overline{k} = \{j \in \mathbb{Z} \mid j \equiv k \pmod{n}\}$, $k = 0, \ldots, n-1$. L'addition est définie ainsi : $\overline{k} + \overline{l} = \overline{k+l}$.
- Le groupe fini $((\mathbb{Z}/p\mathbb{Z})^*, \cdot) = \{\overline{1}, \ldots, \overline{p-1}\}$ des entiers modulo p et $\neq \overline{0}$ avec p **premier**. La multiplication est définie ainsi : $\overline{k} \cdot \overline{l} = \overline{kl}$. Attention : si p n'est pas premier, ce n'est pas un groupe. Par exemple si $p = 4$, $\overline{2}$ n'a pas d'élément symétrique car $\overline{2} \cdot \overline{2} = \overline{4} = \overline{0}$.
- Et bien d'autres encore...

Exemples de groupes

- Le groupe additif des entiers relatifs $(\mathbb{Z}, +)$.
- Le groupe additif des réels $(\mathbb{R}, +)$.
- Le groupe multiplicatif des réels $(\mathbb{R}^*, \cdot)$.
- Le groupe $(\mathfrak{S}_n, \circ)$ des permutations de l'ensemble $\{1, \ldots, n\}$. C'est un groupe non commutatif et fini $(\#(G) = n!)$.
- En géométrie ou en cristallographie, les groupes de symétries.
- Le groupe fini $(\mathbb{Z}/n\mathbb{Z}, +) = \{\overline{0}, \ldots, \overline{n-1}\}$ des entiers modulo n où $\overline{k} = \{j \in \mathbb{Z} \mid j \equiv k \pmod{n}\}$, $k = 0, \ldots, n-1$. L'addition est définie ainsi : $\overline{k} + \overline{l} = \overline{k+l}$.
- Le groupe fini $((\mathbb{Z}/p\mathbb{Z})^*, \cdot) = \{\overline{1}, \ldots, \overline{p-1}\}$ des entiers modulo p et $\neq \overline{0}$ avec p **premier**. La multiplication est définie ainsi : $\overline{k} \cdot \overline{l} = \overline{kl}$. Attention : si p n'est pas premier, ce n'est pas un groupe. Par exemple si $p = 4$, $\overline{2}$ n'a pas d'élément symétrique car $\overline{2} \cdot \overline{2} = \overline{4} = \overline{0}$.
- Et bien d'autres encore...

Exemples de groupes

- Le groupe additif des entiers relatifs $(\mathbb{Z}, +)$.
- Le groupe additif des réels $(\mathbb{R}, +)$.
- Le groupe multiplicatif des réels $(\mathbb{R}^*, \cdot)$.
- Le groupe $(\mathfrak{S}_n, \circ)$ des permutations de l'ensemble $\{1, \ldots, n\}$. C'est un groupe non commutatif et fini $(\#(G) = n!)$.
- En géométrie ou en cristallographie, les groupes de symétries.
- Le groupe fini $(\mathbb{Z}/n\mathbb{Z}, +) = \{\overline{0}, \ldots, \overline{n-1}\}$ des entiers modulo n où $\overline{k} = \{j \in \mathbb{Z} \mid j \equiv k \pmod{n}\}$, $k = 0, \ldots, n-1$. L'addition est définie ainsi : $\overline{k} + \overline{l} = \overline{k+l}$.
- Le groupe fini $((\mathbb{Z}/p\mathbb{Z})^*, \cdot) = \{\overline{1}, \ldots, \overline{p-1}\}$ des entiers modulo p et $\neq \overline{0}$ avec p **premier**. La multiplication est définie ainsi : $\overline{k} \cdot \overline{l} = \overline{kl}$. Attention : si p n'est pas premier, ce n'est pas un groupe. Par exemple si $p = 4$, $\overline{2}$ n'a pas d'élément symétrique car $\overline{2} \cdot \overline{2} = \overline{4} = \overline{0}$.
- Et bien d'autres encore...

Exemples de groupes

- Le groupe additif des entiers relatifs $(\mathbb{Z}, +)$.
- Le groupe additif des réels $(\mathbb{R}, +)$.
- Le groupe multiplicatif des réels $(\mathbb{R}^*, \cdot)$.
- Le groupe $(\mathfrak{S}_n, \circ)$ des permutations de l'ensemble $\{1, \ldots, n\}$. C'est un groupe non commutatif et fini $(\#(G) = n!)$.
- En géométrie ou en cristallographie, les groupes de symétries.
- Le groupe fini $(\mathbb{Z}/n\mathbb{Z}, +) = \{\overline{0}, \ldots, \overline{n-1}\}$ des entiers modulo n où $\overline{k} = \{j \in \mathbb{Z} \mid j \equiv k \pmod{n}\}$, $k = 0, \ldots, n-1$. L'addition est définie ainsi : $\overline{k} + \overline{l} = \overline{k+l}$.
- Le groupe fini $((\mathbb{Z}/p\mathbb{Z})^*, \cdot) = \{\overline{1}, \ldots, \overline{p-1}\}$ des entiers modulo p et $\neq \overline{0}$ avec p **premier**. La multiplication est définie ainsi : $\overline{k} \cdot \overline{l} = \overline{kl}$.
  Attention : si p n'est pas premier, ce n'est pas un groupe. Par exemple si $p = 4$, $\overline{2}$ n'a pas d'élément symétrique car $\overline{2} \cdot \overline{2} = \overline{4} = \overline{0}$.
- Et bien d'autres encore...

Tables d'addition et de multiplication de $\mathbb{Z}/2\mathbb{Z}$

$+$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

Addition

$\cdot$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{0}$
$\overline{1}$	$\overline{0}$	$\overline{1}$

Multiplication

$+$	pair	impair
pair	pair	impair
impair	impair	pair

Interprétation en terme de parité

$\cdot$	pair	impair
pair	pair	pair
impair	pair	impair

Interprétation en terme de parité

Tables d'addition et de multiplication de $\mathbb{Z}/2\mathbb{Z}$

$+$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

Addition

$\cdot$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{0}$
$\overline{1}$	$\overline{0}$	$\overline{1}$

Multiplication

$+$	pair	impair
pair	pair	impair
impair	impair	pair

Interprétation en terme de parité

$\cdot$	pair	impair
pair	pair	pair
impair	pair	impair

Interprétation en terme de parité

Tables d'addition et de multiplication de $\mathbb{Z}/2\mathbb{Z}$

$+$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

Addition

$\cdot$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{0}$
$\overline{1}$	$\overline{0}$	$\overline{1}$

Multiplication

$+$	pair	impair
pair	pair	impair
impair	impair	pair

Interprétation en terme de parité

$\cdot$	pair	impair
pair	pair	pair
impair	pair	impair

Interprétation en terme de parité

Tables d'addition et de multiplication de $\mathbb{Z}/2\mathbb{Z}$

$+$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

Addition

$\cdot$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{0}$
$\overline{1}$	$\overline{0}$	$\overline{1}$

Multiplication

$+$	pair	impair
pair	pair	impair
impair	impair	pair

Interprétation en terme de parité

$\cdot$	pair	impair
pair	pair	pair
impair	pair	impair

Interprétation en terme de parité

Le groupe $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

L'entier 5 étant premier, $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$ est donc un groupe (fini). Voici sa table :

$\cdot$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{1}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{2}$	$\overline{2}$	$\overline{4}$	$\overline{1}$	$\overline{3}$
$\overline{3}$	$\overline{3}$	$\overline{1}$	$\overline{4}$	$\overline{2}$
$\overline{4}$	$\overline{4}$	$\overline{3}$	$\overline{2}$	$\overline{1}$

Table de $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

- $\overline{1}$ est l'élément neutre.
- $\overline{2}$ et $\overline{3}$ sont inverses l'un de l'autre. $\overline{1}$ et $\overline{4}$ sont leur propre inverse.
- Le groupe est commutatif (symétrie par rapport à la diagonale).
- Le groupe est cyclique : $(\mathbb{Z}/5\mathbb{Z})^* = <\overline{2}> = \{\overline{1}, \overline{2}, \overline{2}^2, \overline{2}^3\}$
- Les groupes $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, p *premier*, sont les premiers groupes à avoir été utilisés pour le protocole de Diffie-Hellman.

Le groupe $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

L'entier 5 étant premier, $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$ est donc un groupe (fini). Voici sa table :

$\cdot$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{1}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{2}$	$\overline{2}$	$\overline{4}$	$\overline{1}$	$\overline{3}$
$\overline{3}$	$\overline{3}$	$\overline{1}$	$\overline{4}$	$\overline{2}$
$\overline{4}$	$\overline{4}$	$\overline{3}$	$\overline{2}$	$\overline{1}$

Table de $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

- $\overline{1}$ est l'élément neutre.
- $\overline{2}$ et $\overline{3}$ sont inverses l'un de l'autre. $\overline{1}$ et $\overline{4}$ sont leur propre inverse.
- Le groupe est commutatif (symétrie par rapport à la diagonale).
- Le groupe est cyclique : $(\mathbb{Z}/5\mathbb{Z})^* = <\overline{2}> = \{\overline{1}, \overline{2}, \overline{2}^2, \overline{2}^3\}$
- Les groupes $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, *p premier*, sont les premiers groupes à avoir été utilisés pour le protocole de Diffie-Hellman.

Le groupe $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

L'entier 5 étant premier, $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$ est donc un groupe (fini). Voici sa table :

$\cdot$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{1}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{2}$	$\overline{2}$	$\overline{4}$	$\overline{1}$	$\overline{3}$
$\overline{3}$	$\overline{3}$	$\overline{1}$	$\overline{4}$	$\overline{2}$
$\overline{4}$	$\overline{4}$	$\overline{3}$	$\overline{2}$	$\overline{1}$

Table de $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

- $\overline{1}$ est l'élémént neutre.
- $\overline{2}$ et $\overline{3}$ sont inverses l'un de l'autre. $\overline{1}$ et $\overline{4}$ sont leur propre inverse.
- Le groupe est commutatif (symétrie par rapport à la diagonale).
- Le groupe est cyclique : $(\mathbb{Z}/5\mathbb{Z})^* = <\overline{2}> = \{\overline{1}, \overline{2}, \overline{2}^2, \overline{2}^3\}$
- Les groupes $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, *p premier*, sont les premiers groupes à avoir été utilisés pour le protocole de Diffie-Hellman.

Le groupe $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

L'entier 5 étant premier, $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$ est donc un groupe (fini). Voici sa table :

$\cdot$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{1}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{2}$	$\overline{2}$	$\overline{4}$	$\overline{1}$	$\overline{3}$
$\overline{3}$	$\overline{3}$	$\overline{1}$	$\overline{4}$	$\overline{2}$
$\overline{4}$	$\overline{4}$	$\overline{3}$	$\overline{2}$	$\overline{1}$

Table de $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

- $\overline{1}$ est l'élémént neutre.
- $\overline{2}$ et $\overline{3}$ sont inverses l'un de l'autre. $\overline{1}$ et $\overline{4}$ sont leur propre inverse.
- Le groupe est commutatif (symétrie par rapport à la diagonale).
- Le groupe est cyclique : $(\mathbb{Z}/5\mathbb{Z})^* = <\overline{2}> = \{\overline{1}, \overline{2}, \overline{2}^2, \overline{2}^3\}$
- Les groupes $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, *p premier*, sont les premiers groupes à avoir été utilisés pour le protocole de Diffie-Hellman.

Le groupe $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

L'entier 5 étant premier, $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$ est donc un groupe (fini). Voici sa table :

$\cdot$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{1}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{2}$	$\overline{2}$	$\overline{4}$	$\overline{1}$	$\overline{3}$
$\overline{3}$	$\overline{3}$	$\overline{1}$	$\overline{4}$	$\overline{2}$
$\overline{4}$	$\overline{4}$	$\overline{3}$	$\overline{2}$	$\overline{1}$

Table de $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

- $\overline{1}$ est l'élémént neutre.
- $\overline{2}$ et $\overline{3}$ sont inverses l'un de l'autre. $\overline{1}$ et $\overline{4}$ sont leur propre inverse.
- Le groupe est commutatif (symétrie par rapport à la diagonale).
- Le groupe est cyclique : $(\mathbb{Z}/5\mathbb{Z})^* = <\overline{2}> = \{\overline{1}, \overline{2}, \overline{2}^2, \overline{2}^3\}$
- Les groupes $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, *p premier*, sont les premiers groupes à avoir été utilisés pour le protocole de Diffie-Hellman.

Le groupe $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

L'entier 5 étant premier, $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$ est donc un groupe (fini). Voici sa table :

$\cdot$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{1}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{2}$	$\overline{2}$	$\overline{4}$	$\overline{1}$	$\overline{3}$
$\overline{3}$	$\overline{3}$	$\overline{1}$	$\overline{4}$	$\overline{2}$
$\overline{4}$	$\overline{4}$	$\overline{3}$	$\overline{2}$	$\overline{1}$

Table de $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

- $\overline{1}$ est l'élémént neutre.
- $\overline{2}$ et $\overline{3}$ sont inverses l'un de l'autre. $\overline{1}$ et $\overline{4}$ sont leur propre inverse.
- Le groupe est commutatif (symétrie par rapport à la diagonale).
- Le groupe est cyclique : $(\mathbb{Z}/5\mathbb{Z})^* = <\overline{2}> = \{\overline{1}, \overline{2}, \overline{2}^2, \overline{2}^3\}$
- Les groupes $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, *p premier*, sont les premiers groupes à avoir été utilisés pour le protocole de Diffie-Hellman.

Le groupe $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

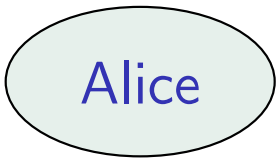
L'entier 5 étant premier, $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$ est donc un groupe (fini). Voici sa table :

$\cdot$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{1}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$
$\overline{2}$	$\overline{2}$	$\overline{4}$	$\overline{1}$	$\overline{3}$
$\overline{3}$	$\overline{3}$	$\overline{1}$	$\overline{4}$	$\overline{2}$
$\overline{4}$	$\overline{4}$	$\overline{3}$	$\overline{2}$	$\overline{1}$

Table de $((\mathbb{Z}/5\mathbb{Z})^*, \cdot)$

- $\overline{1}$ est l'élémént neutre.
- $\overline{2}$ et $\overline{3}$ sont inverses l'un de l'autre. $\overline{1}$ et $\overline{4}$ sont leur propre inverse.
- Le groupe est commutatif (symétrie par rapport à la diagonale).
- Le groupe est cyclique : $(\mathbb{Z}/5\mathbb{Z})^* = <\overline{2}> = \{\overline{1}, \overline{2}, \overline{2}^2, \overline{2}^3\}$
- Les groupes $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, *p premier*, sont les premiers groupes à avoir été utilisés pour le protocole de Diffie-Hellman.

Schéma du protocole - Exponentiation discrète



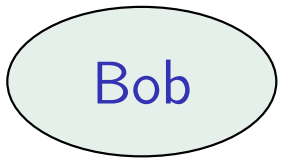


Schéma du protocole - Exponentiation discrète

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice

Bob

Schéma du protocole - Exponentiation discrète

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice choisit
$\alpha \in]0, n[$ secret

Alice

Bob

Schéma du protocole - Exponentiation discrète

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice choisit
$\alpha \in]0, n[$ secret

Alice

Bob choisit
$\beta \in]0, n[$ secret

Bob

Schéma du protocole - Exponentiation discrète

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice choisit
$\alpha \in]0, n[$ secret

Bob choisit
$\beta \in]0, n[$ secret

Alice

Bob

Alice calcule g^{α}

Schéma du protocole - Exponentiation discrète

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice choisit $\alpha \in]0, n[$ secret

Alice

Alice calcule g^{α}

Bob choisit $\beta \in]0, n[$ secret

Bob

Bob calcule g^{β}

Schéma du protocole - Exponentiation discrète

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice choisit $\alpha \in]0, n[$ secret

Bob choisit $\beta \in]0, n[$ secret

Alice calcule g^{α}

Bob calcule g^{β}

Schéma du protocole - Exponentiation discrète

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

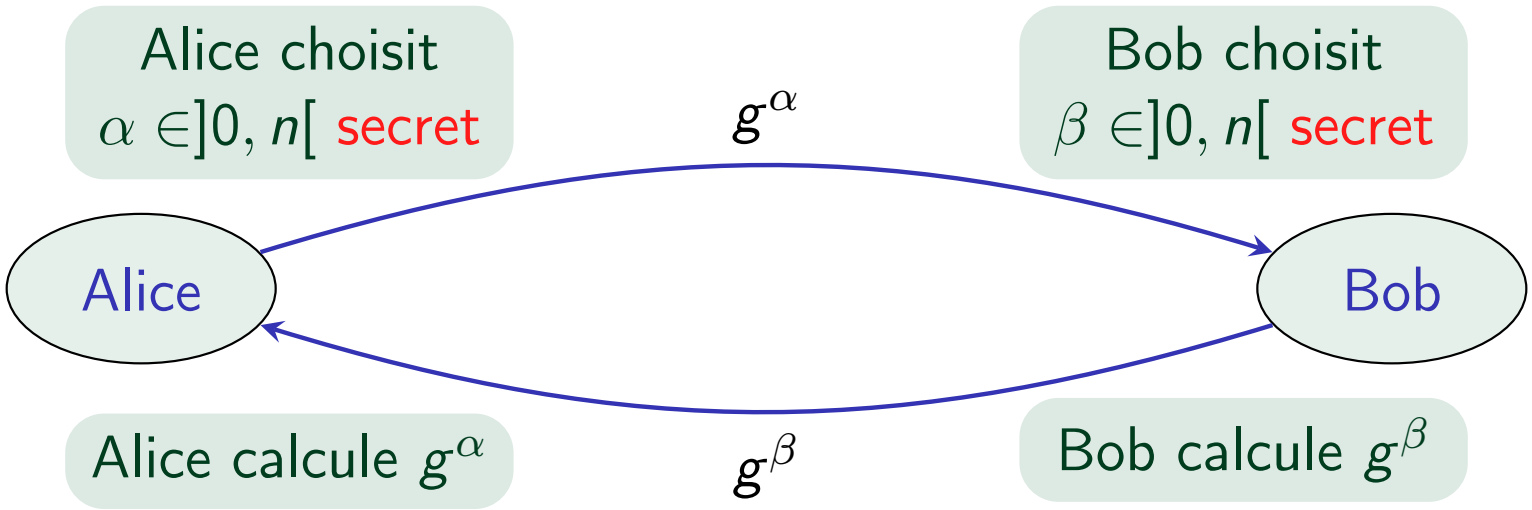


Schéma du protocole - Exponentiation discrète

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

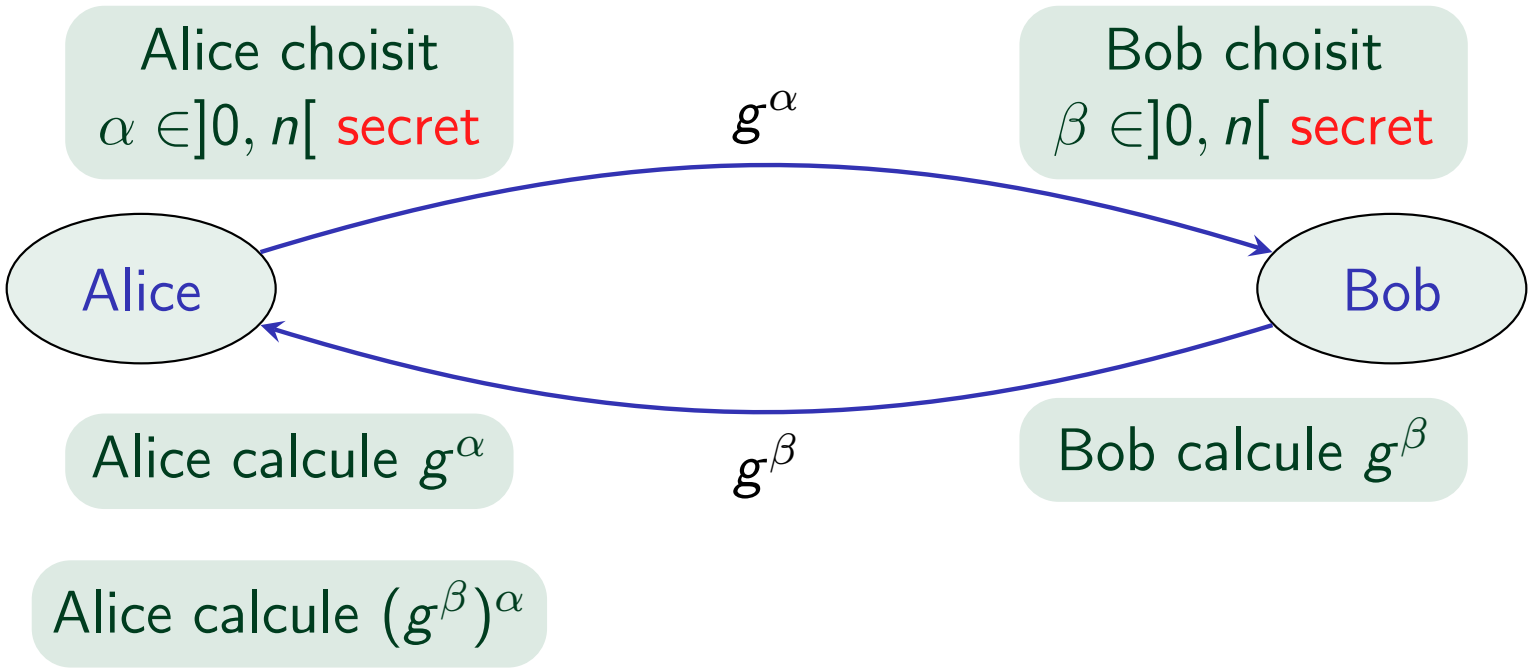


Schéma du protocole - Exponentiation discrète

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

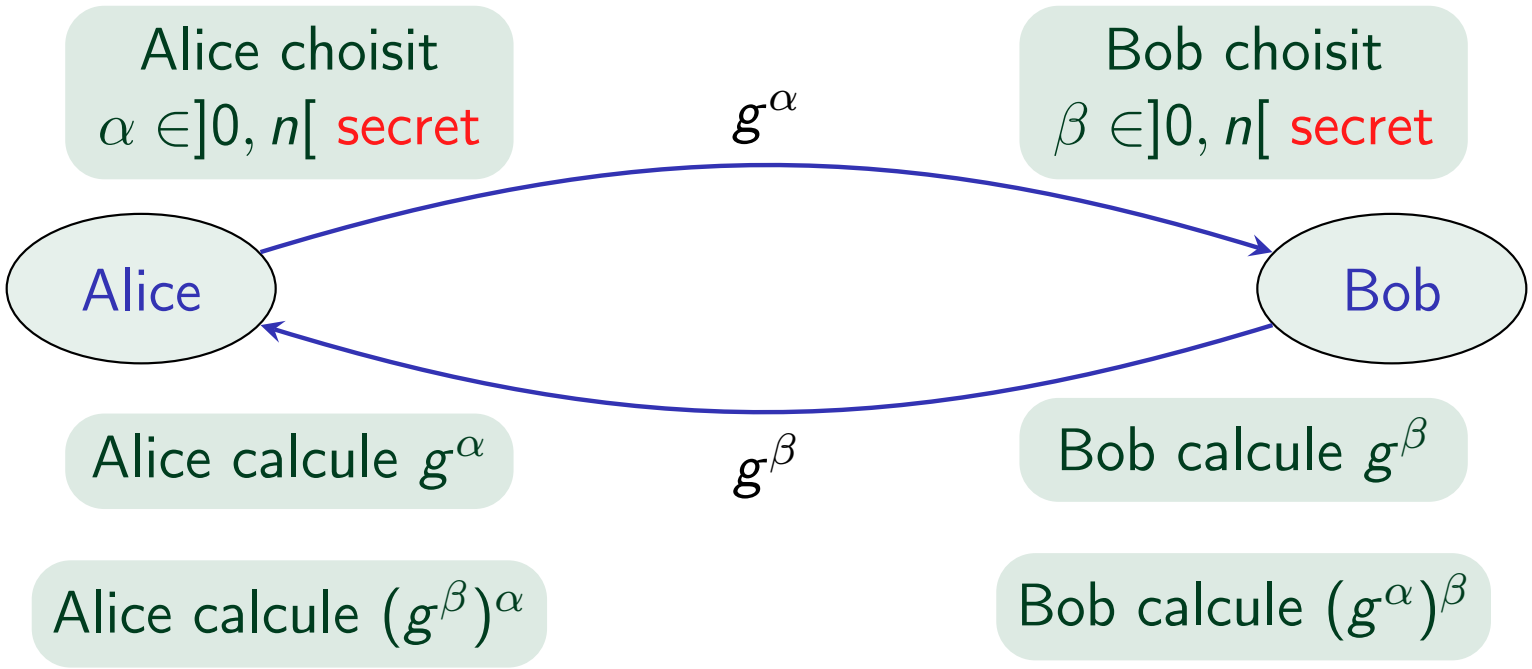


Schéma du protocole - Exponentiation discrète

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

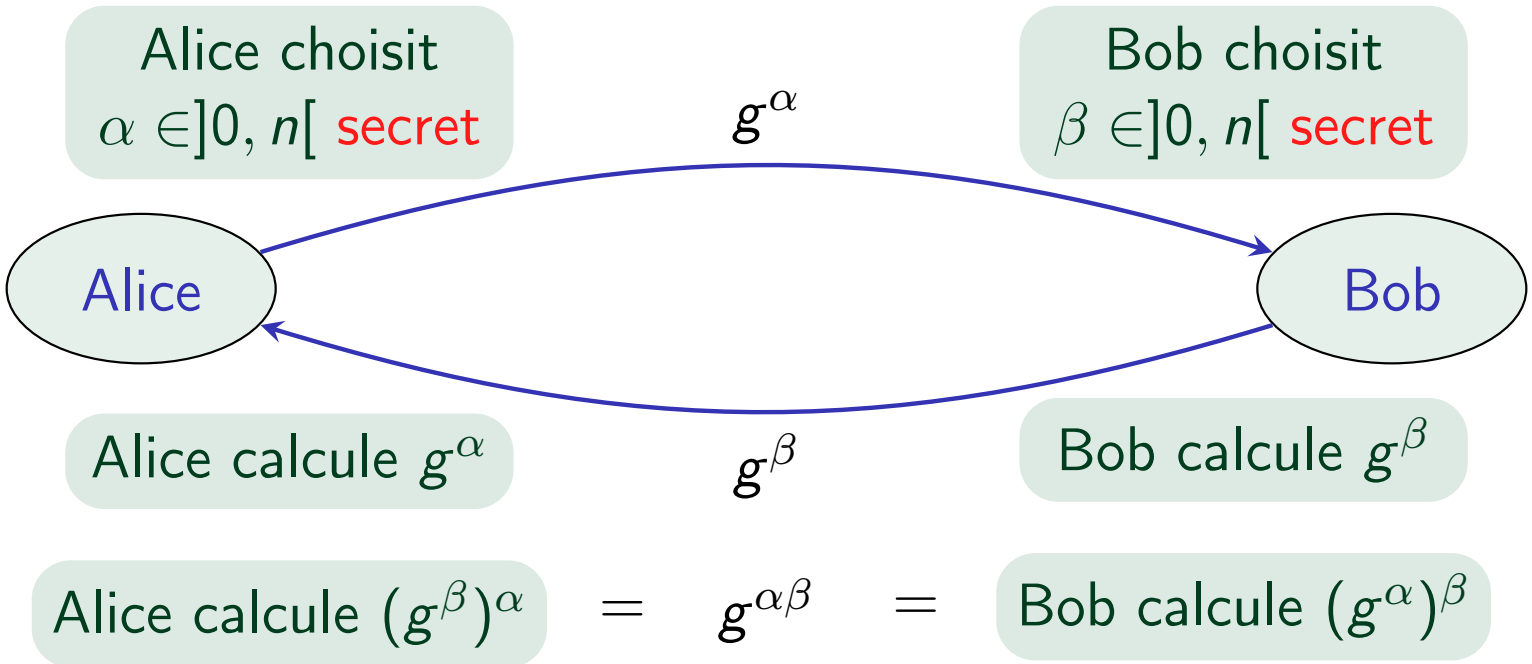
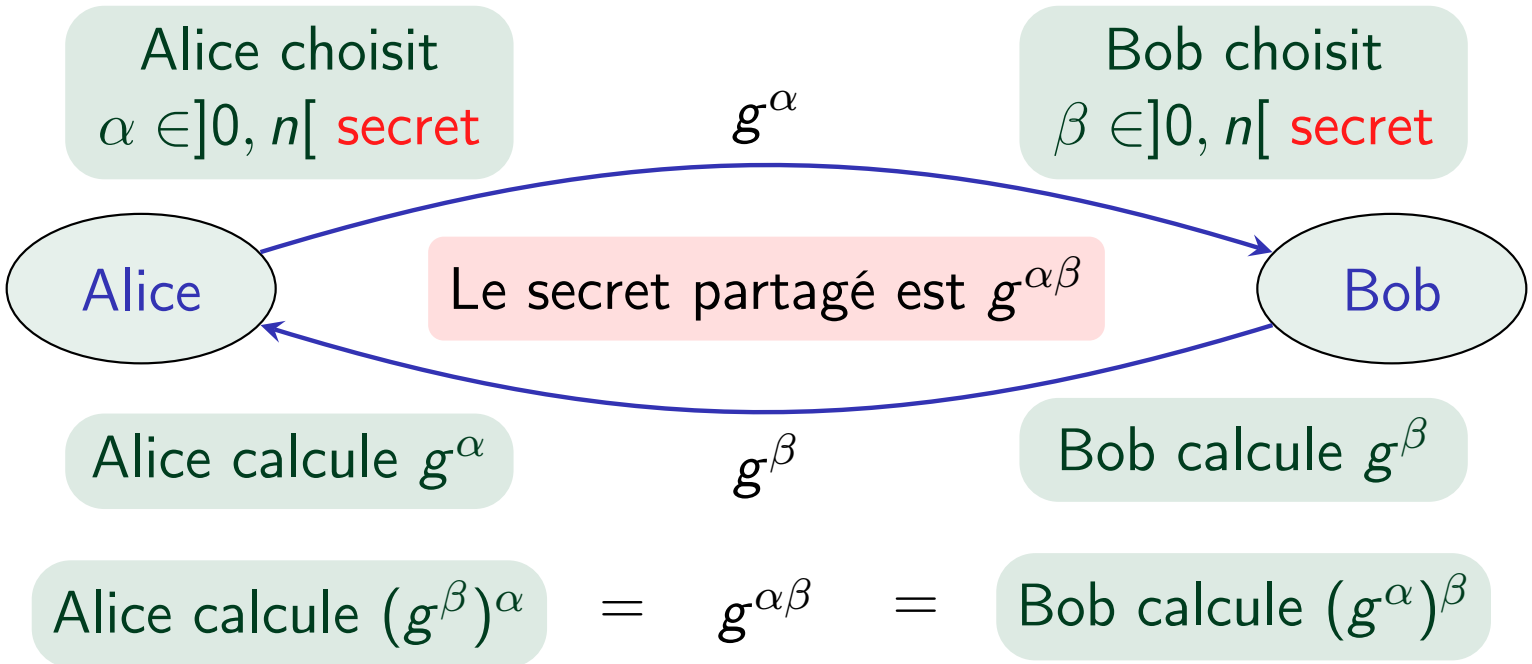


Schéma du protocole - Exponentiation discrète

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n



Robustesse du protocole

Définition (problème du logarithme discret)

Soit h un élément de $<g>=\{1, g, g^2, \cdots, g^k, \cdots\}$, le sous-groupe engendré par $g \in G$. Le problème du logarithme discret (en abrégé PLD) s'énonce ainsi : trouver un entier n tel que $h = g^n$.
Le plus petit entier n tel que $h = g^n$ s'appelle le logarithme en base g de h et est noté $\log_g(h)$.

La robustesse du protocole de Diffie-Hellman repose sur la difficulté de la résolution de ce problème pour des groupes bien choisis. Précisément :

- Un attaquant espionnant le canal non sécurisé ne peut prendre connaissance que de g^α (et aussi de g^β). Il connaît aussi le groupe $(G, \cdot)$ et $g \in G$ d'ordre n qui sont publics.
- Pour connaître α, il doit donc calculer $\log_g(h)$ où $h = g^\alpha$.
- On peut choisir $(G, \cdot)$ de façon à ce que ce calcul soit **impossible** à effectuer en un temps raisonnable.

Robustesse du protocole

Définition (problème du logarithme discret)

Soit h un élément de $<g>=\{1, g, g^2, \cdots, g^k, \cdots\}$, le sous-groupe engendré par $g \in G$. Le problème du logarithme discret (en abrégé PLD) s'énonce ainsi : trouver un entier n tel que $h = g^n$.
Le plus petit entier n tel que $h = g^n$ s'appelle le logarithme en base g de h et est noté $\log_g(h)$.

La robustesse du protocole de Diffie-Hellman repose sur la difficulté de la résolution de ce problème pour des groupes bien choisis. Précisément :

- Un attaquant espionnant le canal non sécurisé ne peut prendre connaissance que de g^α (et aussi de g^β). Il connaît aussi le groupe $(G, \cdot)$ et $g \in G$ d'ordre n qui sont publics.
- Pour connaître α, il doit donc calculer $\log_g(h)$ où $h = g^\alpha$.
- On peut choisir $(G, \cdot)$ de façon à ce que ce calcul soit **impossible** à effectuer en un temps raisonnable.

Robustesse du protocole

Définition (problème du logarithme discret)

Soit h un élément de $<g>=\{1, g, g^2, \cdots, g^k, \cdots\}$, le sous-groupe engendré par $g \in G$. Le problème du logarithme discret (en abrégé PLD) s'énonce ainsi : trouver un entier n tel que $h = g^n$.
Le plus petit entier n tel que $h = g^n$ s'appelle le logarithme en base g de h et est noté $\log_g(h)$.

La robustesse du protocole de Diffie-Hellman repose sur la difficulté de la résolution de ce problème pour des groupes bien choisis. Précisément :

- Un attaquant espionnant le canal non sécurisé ne peut prendre connaissance que de g^α (et aussi de g^β). Il connaît aussi le groupe $(G, \cdot)$ et $g \in G$ d'ordre n qui sont publics.
- Pour connaître α, il doit donc calculer $\log_g(h)$ où $h = g^\alpha$.
- On peut choisir $(G, \cdot)$ de façon à ce que ce calcul soit **impossible** à effectuer en un temps raisonnable.

Robustesse du protocole

Définition (problème du logarithme discret)

Soit h un élément de $<g>=\{1, g, g^2, \cdots, g^k, \cdots\}$, le sous-groupe engendré par $g \in G$. Le problème du logarithme discret (en abrégé PLD) s'énonce ainsi : trouver un entier n tel que $h = g^n$.
Le plus petit entier n tel que $h = g^n$ s'appelle le logarithme en base g de h et est noté $\log_g(h)$.

La robustesse du protocole de Diffie-Hellman repose sur la difficulté de la résolution de ce problème pour des groupes bien choisis. Précisément :

- Un attaquant espionnant le canal non sécurisé ne peut prendre connaissance que de g^α (et aussi de g^β). Il connaît aussi le groupe $(G, \cdot)$ et $g \in G$ d'ordre n qui sont publics.
- Pour connaître α, il doit donc calculer $\log_g(h)$ où $h = g^\alpha$.
- On peut choisir $(G, \cdot)$ de façon à ce que ce calcul soit **impossible** à effectuer en un temps raisonnable.

Robustesse du protocole

Définition (problème du logarithme discret)

Soit h un élément de $<g>=\{1, g, g^2, \cdots, g^k, \cdots\}$, le sous-groupe engendré par $g \in G$. Le problème du logarithme discret (en abrégé PLD) s'énonce ainsi : trouver un entier n tel que $h = g^n$.
Le plus petit entier n tel que $h = g^n$ s'appelle le logarithme en base g de h et est noté $\log_g(h)$.

La robustesse du protocole de Diffie-Hellman repose sur la difficulté de la résolution de ce problème pour des groupes bien choisis. Précisément :

- Un attaquant espionnant le canal non sécurisé ne peut prendre connaissance que de g^α (et aussi de g^β). Il connaît aussi le groupe $(G, \cdot)$ et $g \in G$ d'ordre n qui sont publics.
- Pour connaître α, il doit donc calculer $\log_g(h)$ où $h = g^\alpha$.
- On peut choisir $(G, \cdot)$ de façon à ce que ce calcul soit **impossible** à effectuer en un temps raisonnable.

Difficulté de résolution du PLD

- Cette difficulté est extrêmement variable selon le groupe $(G, \cdot)$ considéré.
- Dans $(\mathbb{R}_+^*, \cdot)$ on dispose du logarithme népérien $\ln : \mathbb{R}_+^* \to \mathbb{R}$. $h = g^\alpha \Rightarrow \ln(h) = \ln(g^\alpha) = \alpha \ln(g) \Rightarrow \alpha = \frac{\ln(h)}{\ln(g)} = \log_g(h)$.
- Dans $(\mathbb{Z}/n\mathbb{Z}, +)$, l'algorithme d'Euclide étendu permet de résoudre facilement ce problème. Ce n'est donc pas non plus un bon candidat pour le protocole de Diffie-Hellman.
- En revanche pour le groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, *p premier*, le meilleur algorithme de résolution connu est en temps en $\mathcal{O}(e^{c\sqrt[3]{\log p (\log\log p)^2}})$.
- C'est encore plus difficile avec les courbes elliptiques construites sur le corps fini $\mathbb{F}_p = (\mathbb{Z}/p\mathbb{Z}, +, \cdot)$, *p premier*. En effet, dans ce cas, le meilleur algorithme de résolution connu dans le cas général est en temps en $\mathcal{O}(\sqrt{p})$.

Difficulté de résolution du PLD

- Cette difficulté est extrêmement variable selon le groupe $(G, \cdot)$ considéré.
- Dans $(\mathbb{R}_+^*, \cdot)$ on dispose du logarithme népérien $\ln : \mathbb{R}_+^* \to \mathbb{R}$.
  $h = g^\alpha \Rightarrow \ln(h) = \ln(g^\alpha) = \alpha \ln(g) \Rightarrow \alpha = \frac{\ln(h)}{\ln(g)} = \log_g(h)$.
- Dans $(\mathbb{Z}/n\mathbb{Z}, +)$, l'algorithme d'Euclide étendu permet de résoudre facilement ce problème. Ce n'est donc pas non plus un bon candidat pour le protocole de Diffie-Hellman.
- En revanche pour le groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, *p premier*, le meilleur algorithme de résolution connu est en temps en $\mathcal{O}(e^{c\sqrt[3]{\log p (\log\log p)^2}})$.
- C'est encore plus difficile avec les courbes elliptiques construites sur le corps fini $\mathbb{F}_p = (\mathbb{Z}/p\mathbb{Z}, +, \cdot)$, *p premier*. En effet, dans ce cas, le meilleur algorithme de résolution connu dans le cas général est en temps en $\mathcal{O}(\sqrt{p})$.

Difficulté de résolution du PLD

- Cette difficulté est extrêmement variable selon le groupe $(G, \cdot)$ considéré.
- Dans $(\mathbb{R}_+^*, \cdot)$ on dispose du logarithme népérien $\ln : \mathbb{R}_+^* \to \mathbb{R}$.
  $h = g^\alpha \Rightarrow \ln(h) = \ln(g^\alpha) = \alpha \ln(g) \Rightarrow \alpha = \frac{\ln(h)}{\ln(g)} = \log_g(h)$.
- Dans $(\mathbb{Z}/n\mathbb{Z}, +)$, l'algorithme d'Euclide étendu permet de résoudre facilement ce problème. Ce n'est donc pas non plus un bon candidat pour le protocole de Diffie-Hellman.
- En revanche pour le groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, *p premier*, le meilleur algorithme de résolution connu est en temps en $\mathcal{O}(e^{c\sqrt[3]{\log p(\log\log p)^2}})$.
- C'est encore plus difficile avec les courbes elliptiques construites sur le corps fini $\mathbb{F}_p = (\mathbb{Z}/p\mathbb{Z}, +, \cdot)$, *p premier*. En effet, dans ce cas, le meilleur algorithme de résolution connu dans le cas général est en temps en $\mathcal{O}(\sqrt{p})$.

Difficulté de résolution du PLD

- Cette difficulté est extrêmement variable selon le groupe $(G, \cdot)$ considéré.
- Dans $(\mathbb{R}_+^*, \cdot)$ on dispose du logarithme népérien $\ln : \mathbb{R}_+^* \to \mathbb{R}$.
  $h = g^\alpha \Rightarrow \ln(h) = \ln(g^\alpha) = \alpha \ln(g) \Rightarrow \alpha = \frac{\ln(h)}{\ln(g)} = \log_g(h)$.
- Dans $(\mathbb{Z}/n\mathbb{Z}, +)$, l'algorithme d'Euclide étendu permet de résoudre facilement ce problème. Ce n'est donc pas non plus un bon candidat pour le protocole de Diffie-Hellman.
- En revanche pour le groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, *p premier*, le meilleur algorithme de résolution connu est en temps en $\mathcal{O}(e^{c\sqrt[3]{\log p (\log\log p)^2}})$.
- C'est encore plus difficile avec les courbes elliptiques construites sur le corps fini $\mathbb{F}_p = (\mathbb{Z}/p\mathbb{Z}, +, \cdot)$, *p premier*. En effet, dans ce cas, le meilleur algorithme de résolution connu dans le cas général est en temps en $\mathcal{O}(\sqrt{p})$.

Difficulté de résolution du PLD

- Cette difficulté est extrêmement variable selon le groupe $(G, \cdot)$ considéré.
- Dans $(\mathbb{R}_+^*, \cdot)$ on dispose du logarithme népérien $\ln : \mathbb{R}_+^* \to \mathbb{R}$.
  $h = g^\alpha \Rightarrow \ln(h) = \ln(g^\alpha) = \alpha \ln(g) \Rightarrow \alpha = \frac{\ln(h)}{\ln(g)} = \log_g(h)$.
- Dans $(\mathbb{Z}/n\mathbb{Z}, +)$, l'algorithme d'Euclide étendu permet de résoudre facilement ce problème. Ce n'est donc pas non plus un bon candidat pour le protocole de Diffie-Hellman.
- En revanche pour le groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, *p premier*, le meilleur algorithme de résolution connu est en temps en $\mathcal{O}(e^{c\sqrt[3]{\log p (\log\log p)^2}})$.
- C'est encore plus difficile avec les courbes elliptiques construites sur le corps fini $\mathbb{F}_p = (\mathbb{Z}/p\mathbb{Z}, +, \cdot)$, *p premier*. En effet, dans ce cas, le meilleur algorithme de résolution connu dans le cas général est en temps en $\mathcal{O}(\sqrt{p})$.

Les groupes $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, p premier

Ce sont à l'heure actuelle, avec les groupes associés aux courbes elliptiques, les groupes les plus utilisés pour le protocole de Diffie-Hellman. On a le résultat suivant :

Théorème (*Gauss*)

Pour tout p premier, $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$ est un groupe cyclique. Un générateur de ce groupe est appelé **élément primitif**.

La résolution du PLD est en général difficile. En revanche, dans certains cas, elle peut être particulièrement aisée. Par exemple, dans le cas où $\#((\mathbb{Z}/p\mathbb{Z})^*) = p - 1 = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ n'a que des petits facteurs. C'est la :

Méthode de Pohlig-Hellman : pour résoudre $y = g^x$ dans $(\mathbb{Z}/p\mathbb{Z})^*$, g élément primitif, il suffit, pour chaque i, de trouver $x \pmod{p_i^{\alpha_i}}$ et ainsi d'obtenir, grâce au théorème chinois, $x \pmod{p-1}$. Cette méthode est **très efficace** si tous les $p_i^{\alpha_i}$ sont petits.

Les groupes $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, p premier

Ce sont à l'heure actuelle, avec les groupes associés aux courbes elliptiques, les groupes les plus utilisés pour le protocole de Diffie-Hellman. On a le résultat suivant :

Théorème (*Gauss*)

Pour tout p premier, $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$ *est un groupe cyclique. Un générateur de ce groupe est appelé* **élément primitif**.

La résolution du PLD est en général difficile. En revanche, dans certains cas, elle peut être particulièrement aisée. Par exemple, dans le cas où $\#((\mathbb{Z}/p\mathbb{Z})^*) = p - 1 = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ n'a que des petits facteurs. C'est la :

Méthode de Pohlig-Hellman : pour résoudre $y = g^x$ dans $(\mathbb{Z}/p\mathbb{Z})^*$, g élément primitif, il suffit, pour chaque i, de trouver $x \pmod{p_i^{\alpha_i}}$ et ainsi d'obtenir, grâce au théorème chinois, $x \pmod{p-1}$. Cette méthode est **très efficace** si tous les $p_i^{\alpha_i}$ sont petits.

Les groupes $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, p premier

Ce sont à l'heure actuelle, avec les groupes associés aux courbes elliptiques, les groupes les plus utilisés pour le protocole de Diffie-Hellman. On a le résultat suivant :

Théorème (*Gauss*)

Pour tout p premier, $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$ *est un groupe cyclique. Un générateur de ce groupe est appelé* **élément primitif**.

La résolution du PLD est en général difficile. En revanche, dans certains cas, elle peut être particulièrement aisée. Par exemple, dans le cas où $\#((\mathbb{Z}/p\mathbb{Z})^*) = p - 1 = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ n'a que des petits facteurs. C'est la :

Méthode de Pohlig-Hellman : pour résoudre $y = g^x$ dans $(\mathbb{Z}/p\mathbb{Z})^*$, g élément primitif, il suffit, pour chaque i, de trouver $x \pmod{p_i^{\alpha_i}}$ et ainsi d'obtenir, grâce au théorème chinois, $x \pmod{p-1}$. Cette méthode est **très efficace** si tous les $p_i^{\alpha_i}$ sont petits.

Les groupes $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$, p premier

Ce sont à l'heure actuelle, avec les groupes associés aux courbes elliptiques, les groupes les plus utilisés pour le protocole de Diffie-Hellman. On a le résultat suivant :

Théorème (*Gauss*)

Pour tout p premier, $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$ *est un groupe cyclique. Un générateur de ce groupe est appelé* **élément primitif**.

La résolution du PLD est en général difficile. En revanche, dans certains cas, elle peut être particulièrement aisée. Par exemple, dans le cas où $\#((\mathbb{Z}/p\mathbb{Z})^*) = p - 1 = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ n'a que des petits facteurs. C'est la :

Méthode de Pohlig-Hellman : pour résoudre $y = g^x$ dans $(\mathbb{Z}/p\mathbb{Z})^*$, g élément primitif, il suffit, pour chaque i, de trouver $x \pmod{p_i^{\alpha_i}}$ et ainsi d'obtenir, grâce au théorème chinois, $x \pmod{p-1}$. Cette méthode est **très efficace** si tous les $p_i^{\alpha_i}$ sont petits.

Le choix du groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$

- Le choix du groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$ se résume au choix de l'entier premier p. Celui-ci doit être suffisamment grand pour empêcher la résolution du PLD par force brute.
- L'entier premier p doit pouvoir résister à l'attaque de Pohlig-Hellman et donc $p-1$ ne doit pas avoir de petits facteurs premiers. Un choix efficace est de prendre pour p un *premier de Sophie Germain*, c'est à dire tel que $(p-1)/2$ soit également premier (on conjecture qu'il existe une infinité de tels nombres).
- Contrairement au système RSA, les nombre premiers considérés sont publics. On peut donc en générer, avec des tailles différentes, une fois pour toutes et ensuite les partager en les publiant au sein de la communauté internet par exemple.
- C'est la solution retenue pour le protocole IPsec (*Internet Protocol Security*) qui utilise des groupes de Diffie-Hellman "bien connus".

Le choix du groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$

- Le choix du groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$ se résume au choix de l'entier premier p. Celui-ci doit être suffisamment grand pour empêcher la résolution du PLD par force brute.
- L'entier premier p doit pouvoir résister à l'attaque de Pohlig-Hellman et donc $p-1$ ne doit pas avoir de petits facteurs premiers. Un choix efficace est de prendre pour p un *premier de Sophie Germain*, c'est à dire tel que $(p-1)/2$ soit également premier (on conjecture qu'il existe une infinité de tels nombres).
- Contrairement au système RSA, les nombre premiers considérés sont publics. On peut donc en générer, avec des tailles différentes, une fois pour toutes et ensuite les partager en les publiant au sein de la communauté internet par exemple.
- C'est la solution retenue pour le protocole IPsec (*Internet Protocol Security*) qui utilise des groupes de Diffie-Hellman "bien connus".

Le choix du groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$

- Le choix du groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$ se résume au choix de l'entier premier p. Celui-ci doit être suffisamment grand pour empêcher la résolution du PLD par force brute.
- L'entier premier p doit pouvoir résister à l'attaque de Pohlig-Hellman et donc $p-1$ ne doit pas avoir de petits facteurs premiers. Un choix efficace est de prendre pour p un *premier de Sophie Germain*, c'est à dire tel que $(p-1)/2$ soit également premier (on conjecture qu'il existe une infinité de tels nombres).
- Contrairement au système RSA, les nombre premiers considérés sont publics. On peut donc en générer, avec des tailles différentes, une fois pour toutes et ensuite les partager en les publiant au sein de la communauté internet par exemple.
- C'est la solution retenue pour le protocole IPsec (*Internet Protocol Security*) qui utilise des groupes de Diffie-Hellman "bien connus".

Le choix du groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$

- Le choix du groupe $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$ se résume au choix de l'entier premier p. Celui-ci doit être suffisamment grand pour empêcher la résolution du PLD par force brute.
- L'entier premier p doit pouvoir résister à l'attaque de Pohlig-Hellman et donc $p-1$ ne doit pas avoir de petits facteurs premiers. Un choix efficace est de prendre pour p un *premier de Sophie Germain*, c'est à dire tel que $(p-1)/2$ soit également premier (on conjecture qu'il existe une infinité de tels nombres).
- Contrairement au système RSA, les nombre premiers considérés sont publics. On peut donc en générer, avec des tailles différentes, une fois pour toutes et ensuite les partager en les publiant au sein de la communauté internet par exemple.
- C'est la solution retenue pour le protocole IPsec (*Internet Protocol Security*) qui utilise des groupes de Diffie-Hellman "bien connus".

Well-Known Group 2 - RFC 2412

Un exemple extrait de la RFC 2412 (1024 bits) :

$$
\begin{aligned}
p =& 2^{1024} - 2^{960} - 1 + 2^{64}(\lfloor 2^{894}\pi \rfloor + 129093) \\
=& 179769313486231590770839156793787453197860296048756011706444 \\
& 423684197180216158519368947833795864925541502180565485980503 \\
& 646440548199239100050792877003355816639229553136239076508735 \\
& 759914822574862575007425302077447712589550957937778424442426 \\
& 617334727629299387668709205606050270810842907692932019128194 \\
& 467627007
\end{aligned}
$$

Les bits du milieu de p proviennent du développement binaire de π (dont le calcul est facile grâce à la formule de *Bailey–Borwein–Plouffe*) afin d'écarter tout soupçon quant au fait que p aurait été **délibérément choisi comme étant faible**.

Well-Known Group 2 - RFC 2412

Un exemple extrait de la RFC 2412 (1024 bits) :

$$
\begin{aligned}
p =& 2^{1024} - 2^{960} - 1 + 2^{64}(\lfloor 2^{894}\pi \rfloor + 129093) \\
=& 179769313486231590770839156793787453197860296048756011706444 \\
& 423684197180216158519368947833795864925541502180565485980503 \\
& 646440548199239100050792877003355816639229553136239076508735 \\
& 759914822574862575007425302077447712589550957937778424442426 \\
& 617334727629299387668709205606050270810842907692932019128194 \\
& 467627007
\end{aligned}
$$

Les bits du milieu de p proviennent du développement binaire de π (dont le calcul est facile grâce à la formule de *Bailey–Borwein–Plouffe*) afin d'écarter tout soupçon quant au fait que p aurait été **délibérément choisi comme étant faible**.

Well-Known Group 2 - RFC 2412

L'écriture hexadécimale de p est :

$p =$ FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE65381
FFFFFFFF FFFFFFFF

Dans l'écriture hexadécimale de p, on remarque que les 64 premiers et derniers bits de p valent 1. Cela permet d'accélérer les algorithmes de calcul de reste (classique ou de *Montgomery*).

Well-Known Group 2 - RFC 2412

L'écriture hexadécimale de p est :

$$p = \texttt{FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1}$$
$$\texttt{29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD}$$
$$\texttt{EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245}$$
$$\texttt{E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED}$$
$$\texttt{EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE65381}$$
$$\texttt{FFFFFFFF FFFFFFFF}$$

Dans l'écriture hexadécimale de p, on remarque que les 64 premiers et derniers bits de p valent 1. Cela permet d'accélérer les algorithmes de calcul de reste (classique ou de *Montgomery*).

Well-Known Group 2 - RFC 2412

- Le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ a quatre sous-groupes : le sous-groupe trivial $\{1\}$, le sous-groupe $\{1, p-1\}$ d'ordre 2, un sous-groupe $\mathcal{C}_q$ d'ordre $q = (p-1)/2$ et lui-même. Il n'y en a pas d'autres car q est premier.
- On ne va pas choisir le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ tout entier car, si g est un élément primitif, un attaquant connaissant $g^x \pmod p$ peut savoir si x est pair ou impair selon que g^x est un carré modulo p ou pas. Or pour savoir si g^x est un carré modulo p, il suffit de calculer le *symbole de Legendre* $\left(\frac{g^x}{p}\right)$, calcul pour lequel il existe un algorithme efficace.
- On évitera bien évidemment les sous-groupes triviaux $\{1\}$ et $\{1, p-1\}$.
- On va donc utiliser le sous-groupe $\mathcal{C}_q$ et choisir 2 comme générateur de ce groupe, choix qui permet d'accélérer notablement l'exponentiation modulaire. 2 est un générateur car $\#(<2>) > 1$ et $\#(<2>) \mid q$ (théorème de *Lagrange*). Comme q est premier, on a : $\#(<2>) = q$.

Well-Known Group 2 - RFC 2412

- Le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ a quatre sous-groupes : le sous-groupe trivial $\{1\}$, le sous-groupe $\{1, p-1\}$ d'ordre 2, un sous-groupe $\mathcal{C}_q$ d'ordre $q = (p-1)/2$ et lui-même. Il n'y en a pas d'autres car q est premier.
- On ne va pas choisir le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ tout entier car, si g est un élément primitif, un attaquant connaissant $g^x \pmod p$ peut savoir si x est pair ou impair selon que g^x est un carré modulo p ou pas. Or pour savoir si g^x est un carré modulo p, il suffit de calculer le *symbole de Legendre* $\left(\frac{g^x}{p}\right)$, calcul pour lequel il existe un algorithme efficace.
- On évitera bien évidemment les sous-groupes triviaux $\{1\}$ et $\{1, p-1\}$.
- On va donc utiliser le sous-groupe $\mathcal{C}_q$ et choisir 2 comme générateur de ce groupe, choix qui permet d'accélérer notablement l'exponentiation modulaire. 2 est un générateur car $\#(<2>) > 1$ et $\#(<2>) \mid q$ (théorème de *Lagrange*). Comme q est premier, on a : $\#(<2>) = q$.

Well-Known Group 2 - RFC 2412

- Le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ a quatre sous-groupes : le sous-groupe trivial $\{1\}$, le sous-groupe $\{1, p-1\}$ d'ordre 2, un sous-groupe $\mathcal{C}_q$ d'ordre $q = (p-1)/2$ et lui-même. Il n'y en a pas d'autres car q est premier.
- On ne va pas choisir le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ tout entier car, si g est un élément primitif, un attaquant connaissant $g^x \pmod p$ peut savoir si x est pair ou impair selon que g^x est un carré modulo p ou pas. Or pour savoir si g^x est un carré modulo p, il suffit de calculer le *symbole de Legendre* $\left(\frac{g^x}{p}\right)$, calcul pour lequel il existe un algorithme efficace.
- On évitera bien évidemment les sous-groupes triviaux $\{1\}$ et $\{1, p-1\}$.
- On va donc utiliser le sous-groupe $\mathcal{C}_q$ et choisir 2 comme générateur de ce groupe, choix qui permet d'accélérer notablement l'exponentiation modulaire. 2 est un générateur car $\#(<2>) > 1$ et $\#(<2>) \mid q$ (théorème de *Lagrange*). Comme q est premier, on a : $\#(<2>) = q$.

Well-Known Group 2 - RFC 2412

- Le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ a quatre sous-groupes : le sous-groupe trivial $\{1\}$, le sous-groupe $\{1, p-1\}$ d'ordre 2, un sous-groupe $\mathcal{C}_q$ d'ordre $q = (p-1)/2$ et lui-même. Il n'y en a pas d'autres car q est premier.
- On ne va pas choisir le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ tout entier car, si g est un élément primitif, un attaquant connaissant $g^x \pmod p$ peut savoir si x est pair ou impair selon que g^x est un carré modulo p ou pas. Or pour savoir si g^x est un carré modulo p, il suffit de calculer le *symbole de Legendre* $\left(\frac{g^x}{p}\right)$, calcul pour lequel il existe un algorithme efficace.
- On évitera bien évidemment les sous-groupes triviaux $\{1\}$ et $\{1, p-1\}$.
- On va donc utiliser le sous-groupe $\mathcal{C}_q$ et choisir 2 comme générateur de ce groupe, choix qui permet d'accélérer notablement l'exponentiation modulaire. 2 est un générateur car $\#(<2>) > 1$ et $\#(<2>) \mid q$ (théorème de *Lagrange*). Comme q est premier, on a : $\#(<2>) = q$.

Well-Known Groups - RFCs 2412, 3526, 5114,...

- D'autres groupes de Diffie-Hellman ont été définis dans différentes RFCs avec des longueurs variables : $n = 2048$, 3072, 4096, 6144 et 8192 bits.
- Dans tous les cas, p est défini par la formule :

$$p = 2^n - 2^{n-64} - 1 + 2^{64}(\lfloor 2^{n-130}\pi \rfloor + k)$$

où k est le plus petit entier > 0 tel que p soit un *premier de Sophie Germain*.

- Comme précédemment, on choisit pour chaque p le sous-groupe $\mathcal{C}_q$ où $q = (p-1)/2$ et pour générateur de ce sous-groupe l'entier 2.
- Remarque importante : l'entier 2 n'engendre pas le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ tout entier mais seulement le sous-groupe $\mathcal{C}_q$. Cela permet une vérification supplémentaire à savoir : lorsque, par exemple, *Alice* reçoit de *Bob* g^y, elle peut vérifier que $g^y \in \mathcal{C}_q$.

Well-Known Groups - RFCs 2412, 3526, 5114,...

- D'autres groupes de Diffie-Hellman ont été définis dans différentes RFCs avec des longueurs variables : $n = 2048$, 3072, 4096, 6144 et 8192 bits.
- Dans tous les cas, p est défini par la formule :

$$p = 2^n - 2^{n-64} - 1 + 2^{64}(\lfloor 2^{n-130}\pi \rfloor + k)$$

où k est le plus petit entier > 0 tel que p soit un *premier de Sophie Germain*.

- Comme précédemment, on choisit pour chaque p le sous-groupe $\mathcal{C}_q$ où $q = (p-1)/2$ et pour générateur de ce sous-groupe l'entier 2.
- Remarque importante : l'entier 2 n'engendre pas le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ tout entier mais seulement le sous-groupe $\mathcal{C}_q$. Cela permet une vérification supplémentaire à savoir : lorsque, par exemple, *Alice* reçoit de *Bob* g^y, elle peut vérifier que $g^y \in \mathcal{C}_q$.

Well-Known Groups - RFCs 2412, 3526, 5114,...

- D'autres groupes de Diffie-Hellman ont été définis dans différentes RFCs avec des longueurs variables : $n = 2048$, 3072, 4096, 6144 et 8192 bits.
- Dans tous les cas, p est défini par la formule :

$$p = 2^n - 2^{n-64} - 1 + 2^{64}(\lfloor 2^{n-130}\pi \rfloor + k)$$

où k est le plus petit entier > 0 tel que p soit un *premier de Sophie Germain*.

- Comme précédemment, on choisit pour chaque p le sous-groupe $\mathcal{C}_q$ où $q = (p-1)/2$ et pour générateur de ce sous-groupe l'entier 2.
- Remarque importante : l'entier 2 n'engendre pas le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ tout entier mais seulement le sous-groupe $\mathcal{C}_q$. Cela permet une vérification supplémentaire à savoir : lorsque, par exemple, *Alice* reçoit de *Bob* g^y, elle peut vérifier que $g^y \in \mathcal{C}_q$.

Well-Known Groups - RFCs 2412, 3526, 5114,...

- D'autres groupes de Diffie-Hellman ont été définis dans différentes RFCs avec des longueurs variables : $n = 2048$, 3072, 4096, 6144 et 8192 bits.
- Dans tous les cas, p est défini par la formule :

$$p = 2^n - 2^{n-64} - 1 + 2^{64}(\lfloor 2^{n-130}\pi \rfloor + k)$$

où k est le plus petit entier > 0 tel que p soit un *premier de Sophie Germain*.

- Comme précédemment, on choisit pour chaque p le sous-groupe $\mathcal{C}_q$ où $q = (p-1)/2$ et pour générateur de ce sous-groupe l'entier 2.
- Remarque importante : l'entier 2 n'engendre pas le groupe $(\mathbb{Z}/p\mathbb{Z})^*$ tout entier mais seulement le sous-groupe $\mathcal{C}_q$. Cela permet une vérification supplémentaire à savoir : lorsque, par exemple, *Alice* reçoit de *Bob* g^y, elle peut vérifier que $g^y \in \mathcal{C}_q$.

Le schéma du protocole revisité

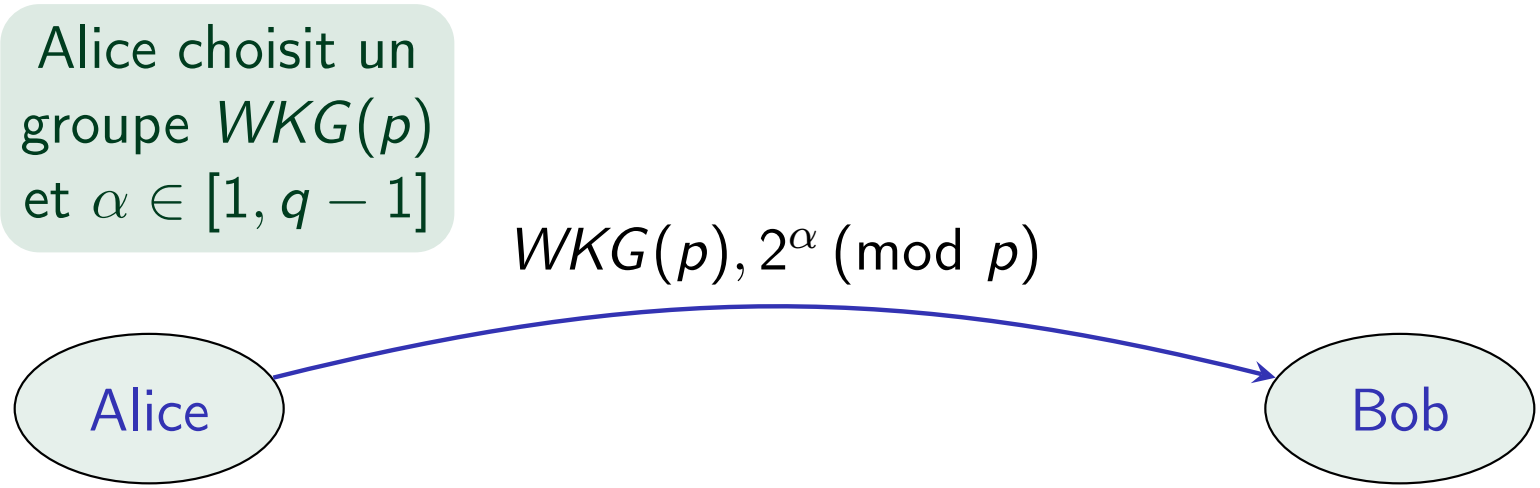
Le schéma du protocole revisité

Alice choisit un groupe $WKG(p)$ et $\alpha \in [1, q-1]$

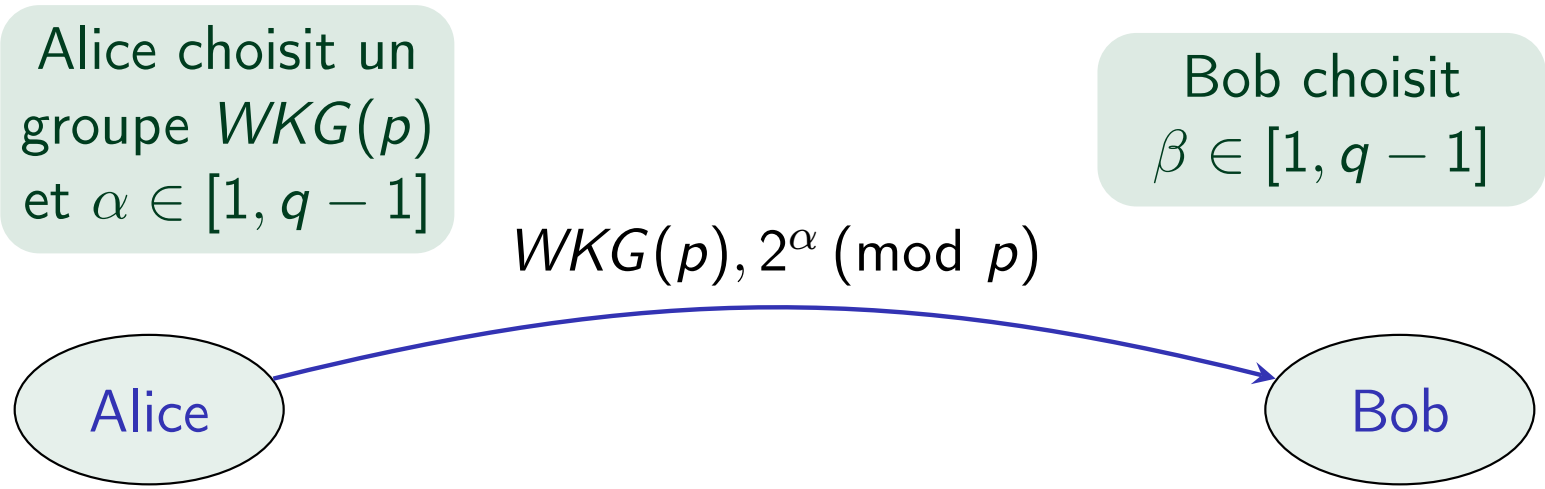
Alice

Bob

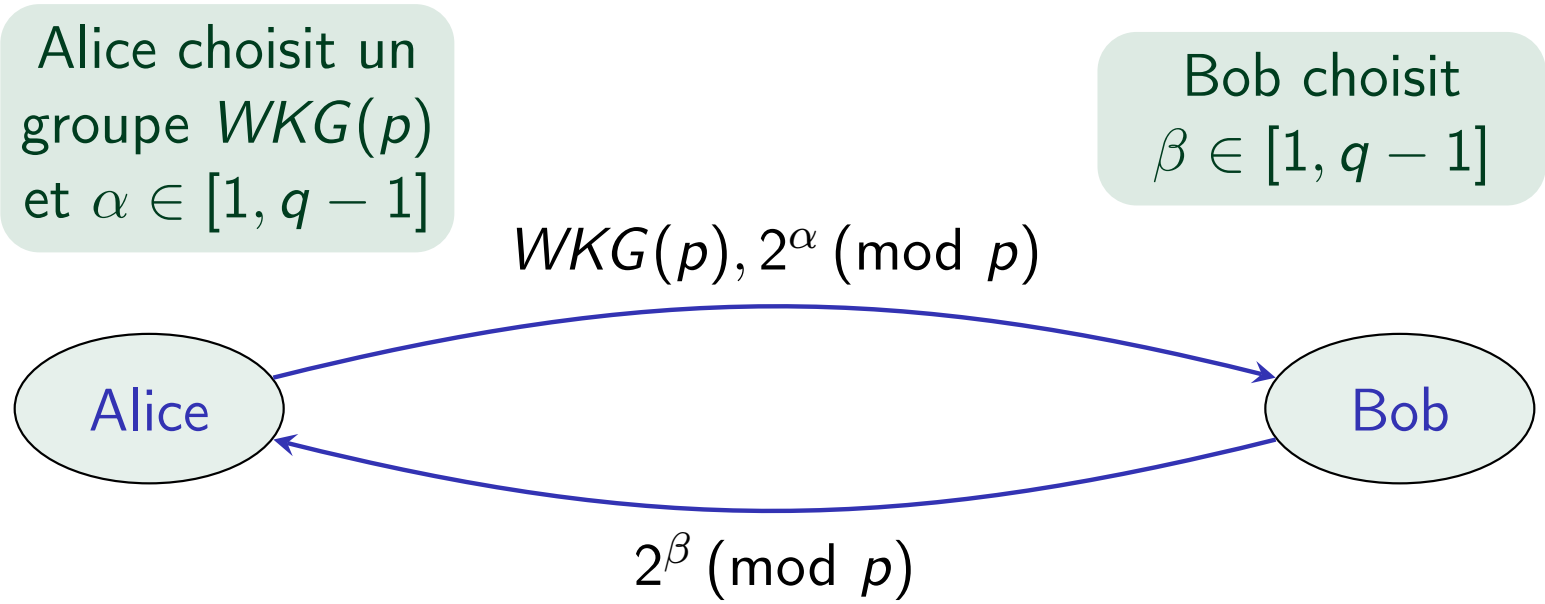
Le schéma du protocole revisité



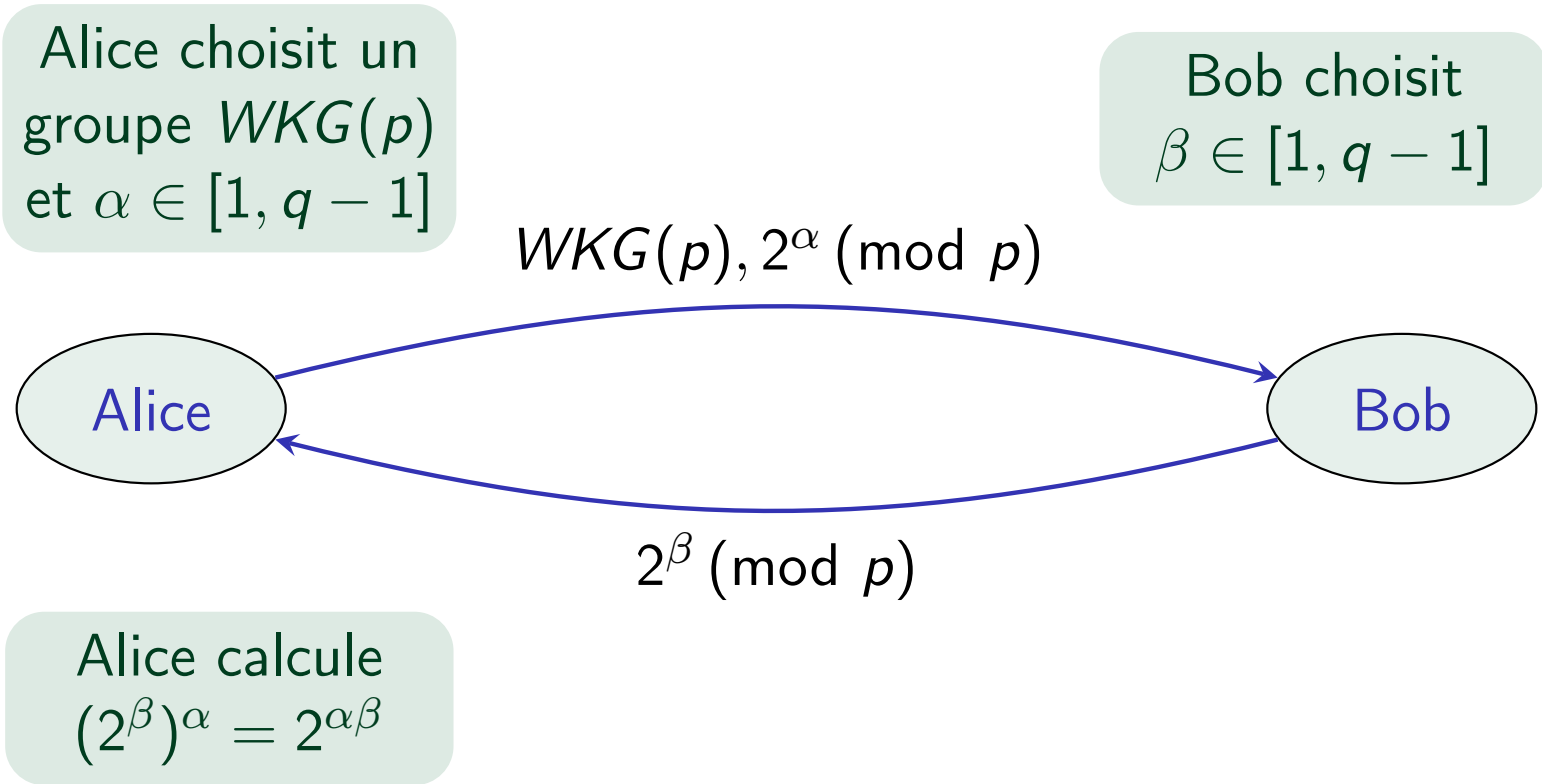
Le schéma du protocole revisité



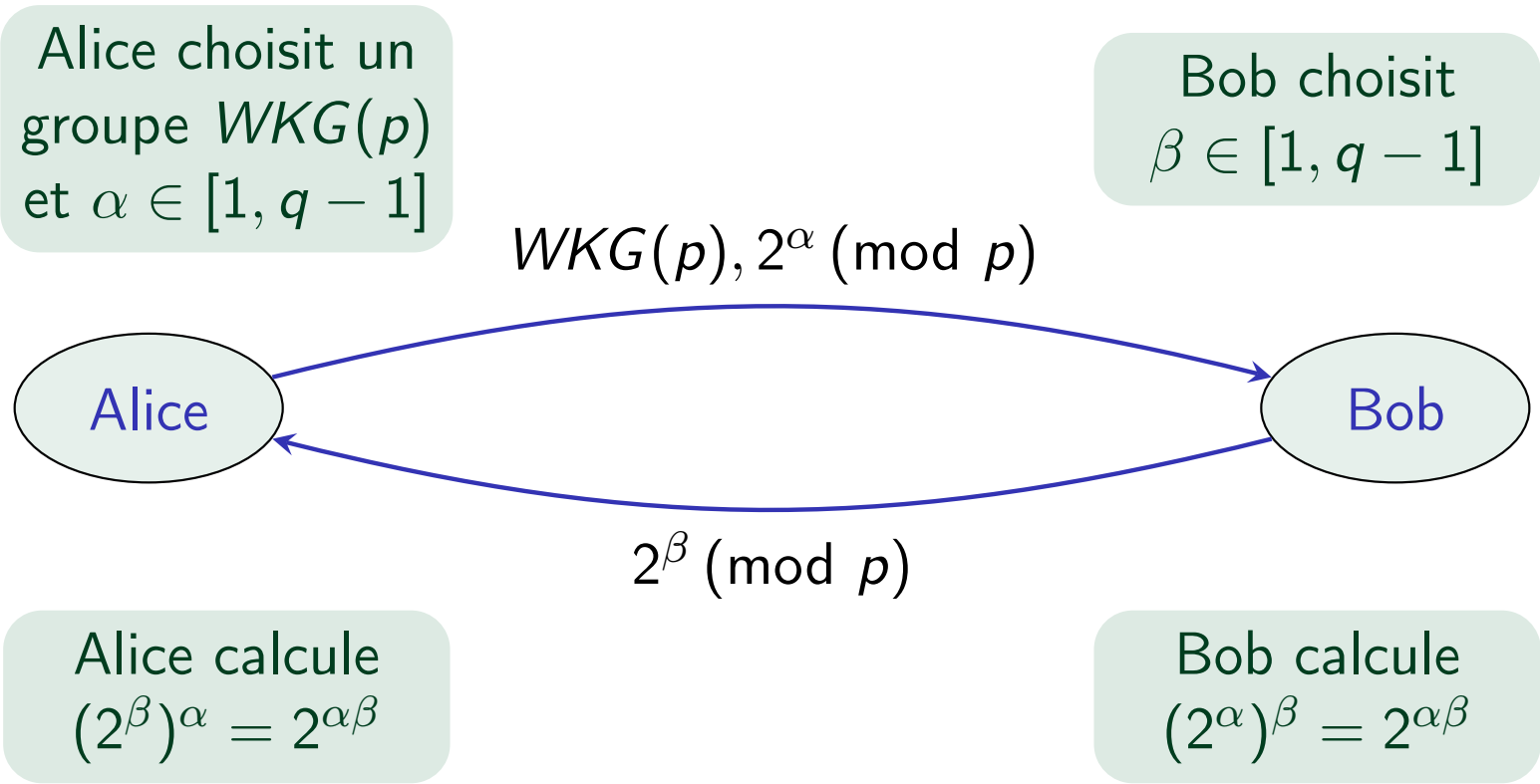
Le schéma du protocole revisité



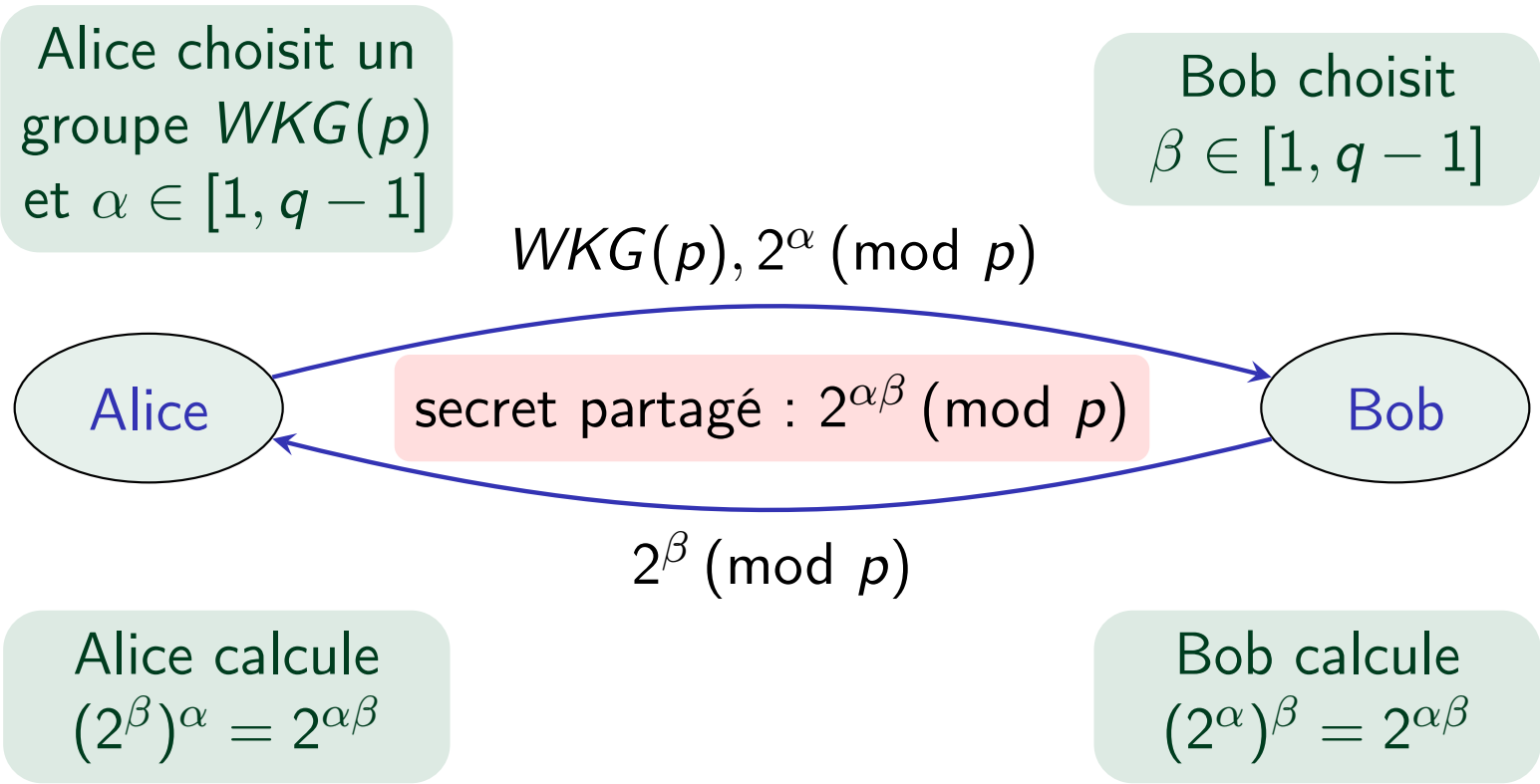
Le schéma du protocole revisité



Le schéma du protocole revisité



Le schéma du protocole revisité



L'attaque "man in the middle"

L'attaque "man in the middle"

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice

Bob

L'attaque "man in the middle"

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice choisit
$\alpha \in]0, n[$ secret

Alice

Bob

L'attaque "man in the middle"

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice choisit
$\alpha \in]0, n[$ secret

Bob choisit
$\beta \in]0, n[$ secret

Alice

Bob

L'attaque "man in the middle"

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice choisit
$\alpha \in]0, n[$ secret

Bob choisit
$\beta \in]0, n[$ secret

Alice

Bob

Alice calcule g^{α}

L'attaque "man in the middle"

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice choisit
$\alpha \in]0, n[$ secret

Bob choisit
$\beta \in]0, n[$ secret

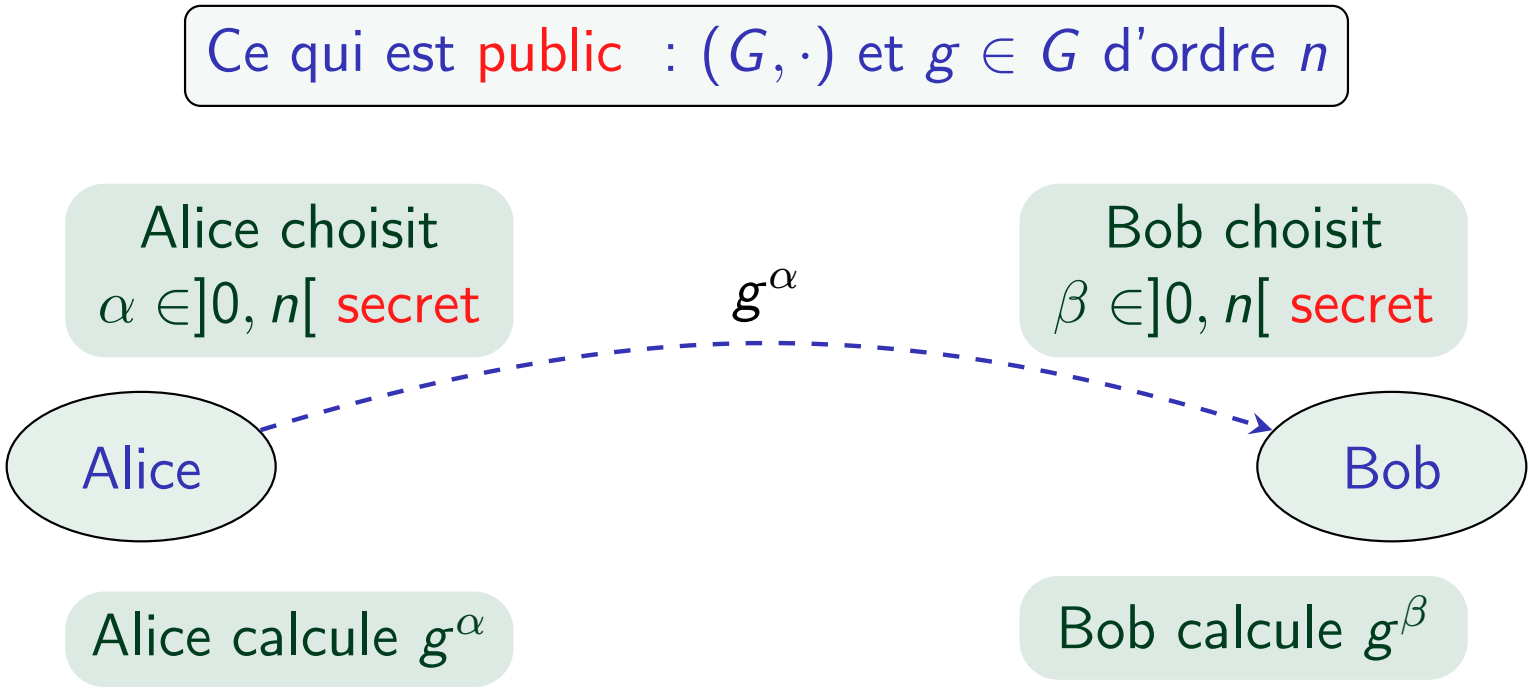
Alice

Bob

Alice calcule g^{α}

Bob calcule g^{β}

L'attaque "man in the middle"



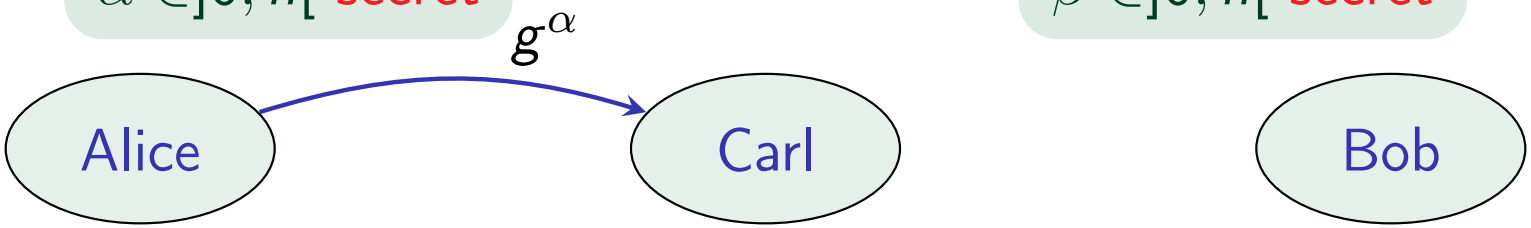
L'attaque "man in the middle"

Ce qui est public : $(G, \cdot)$ et $g \in G$ d'ordre n

Alice choisit $\alpha \in]0, n[$ secret

Carl intercepte l'envoi d'Alice

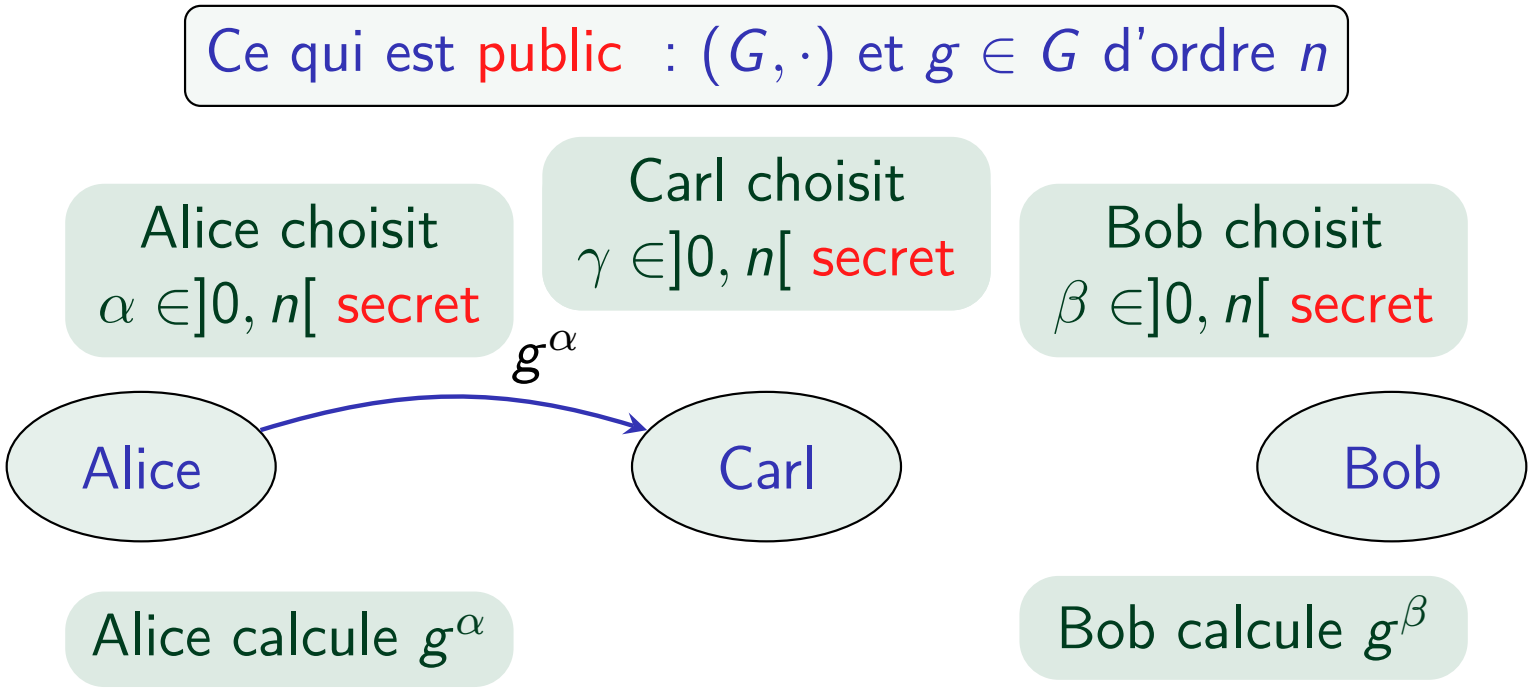
Bob choisit $\beta \in]0, n[$ secret



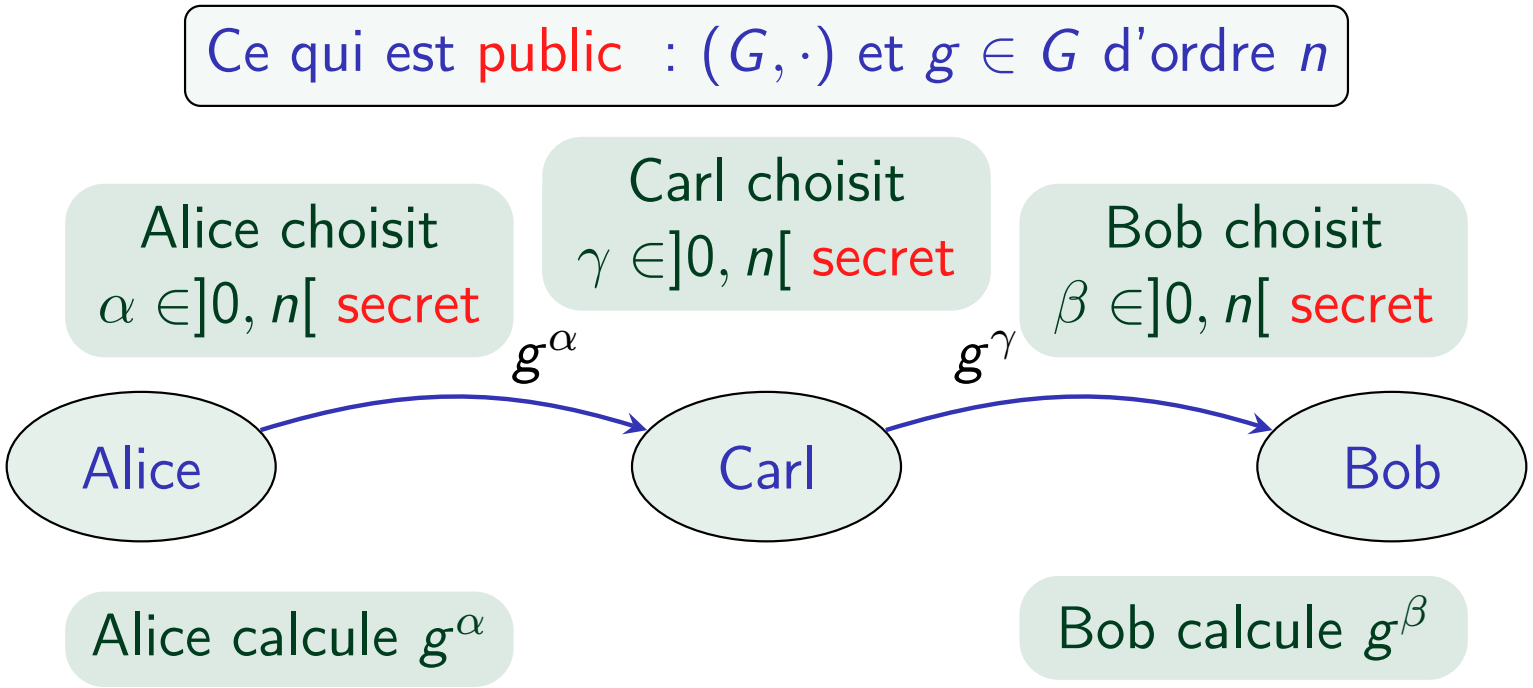
Alice calcule g^α

Bob calcule g^β

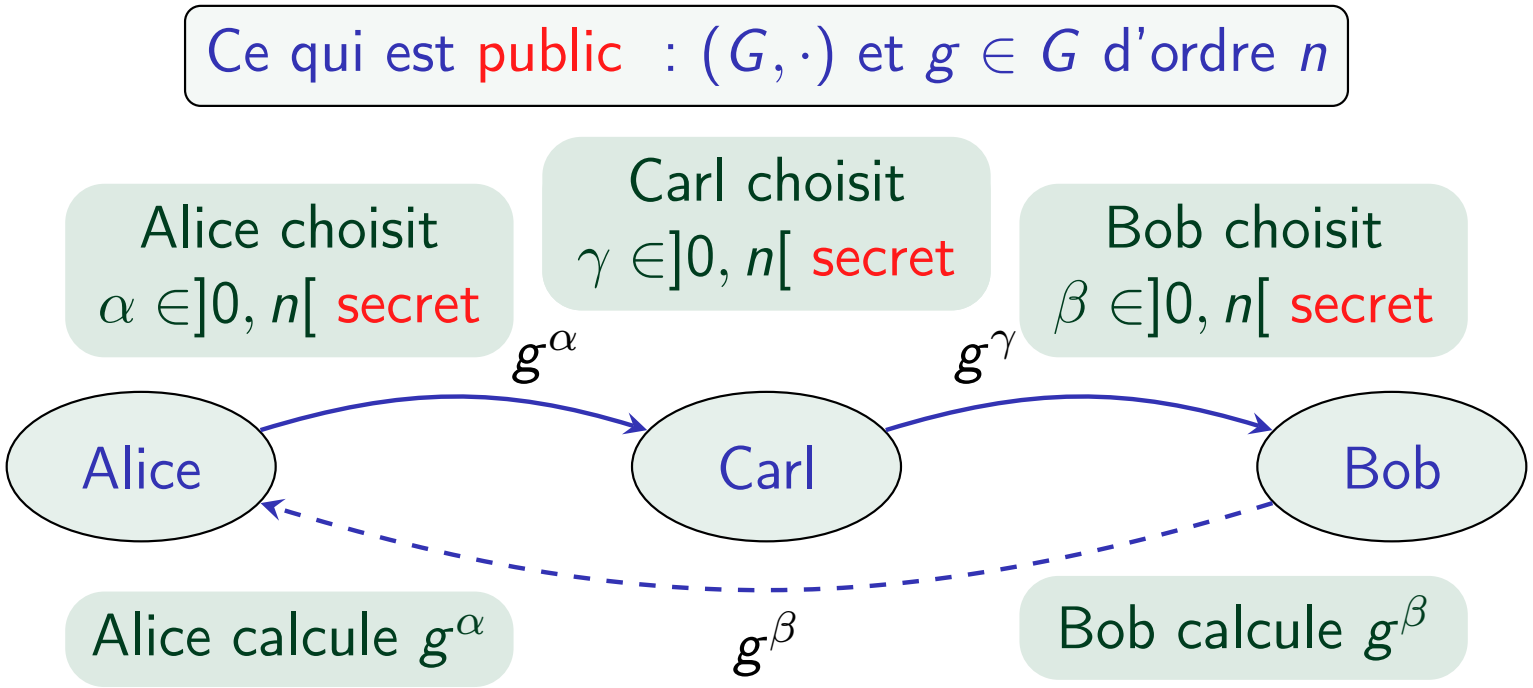
L'attaque "man in the middle"



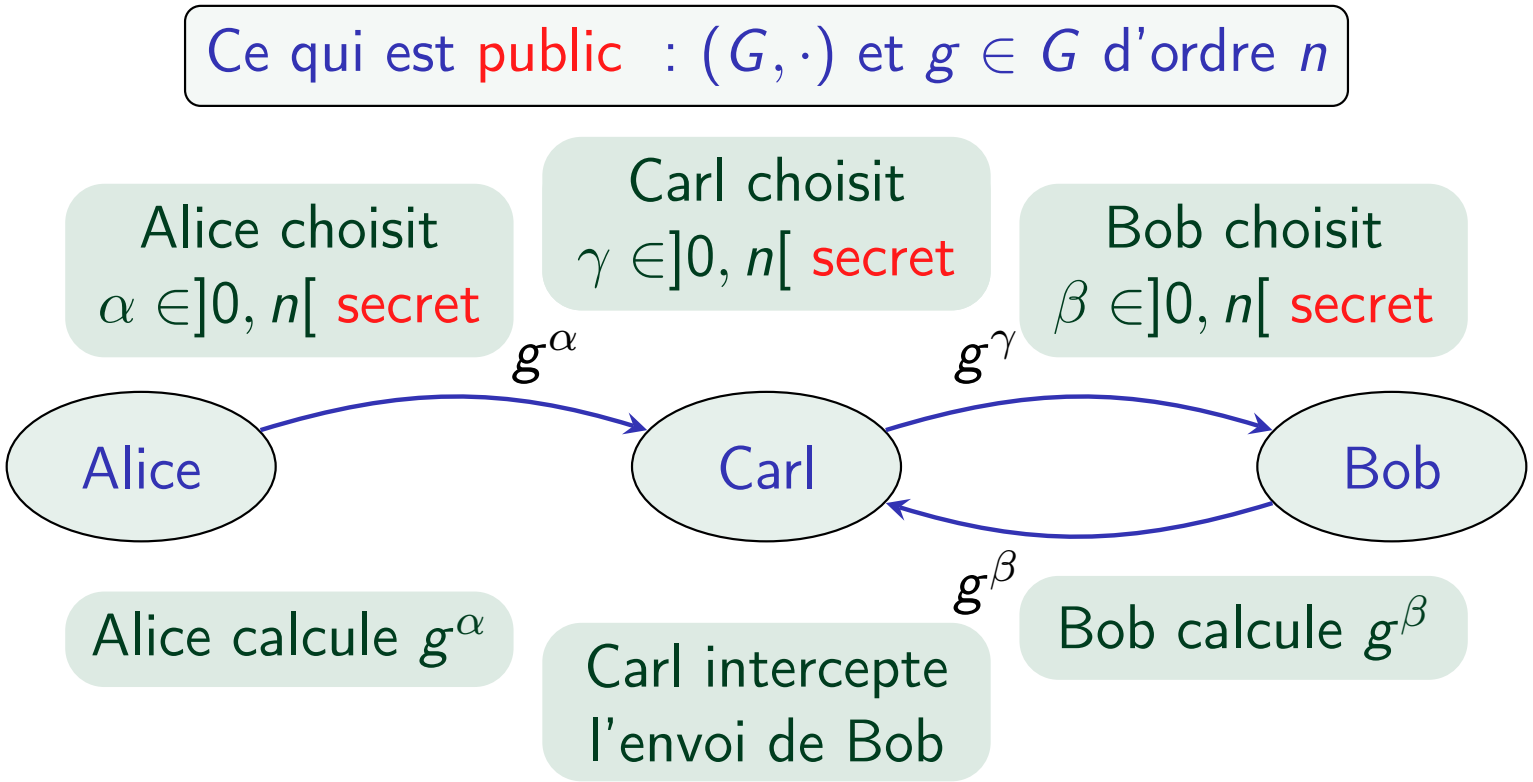
L'attaque "man in the middle"



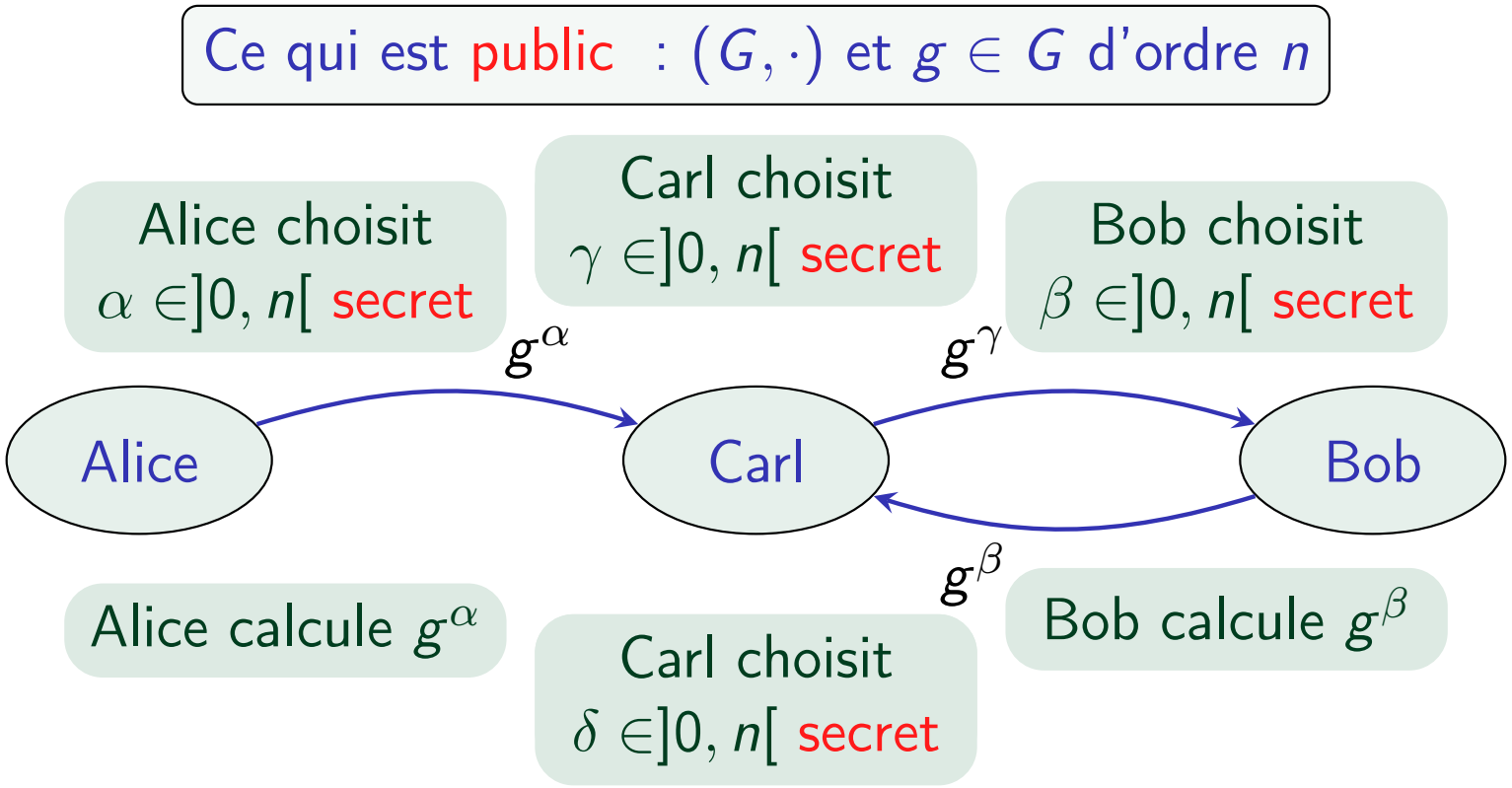
L'attaque "man in the middle"



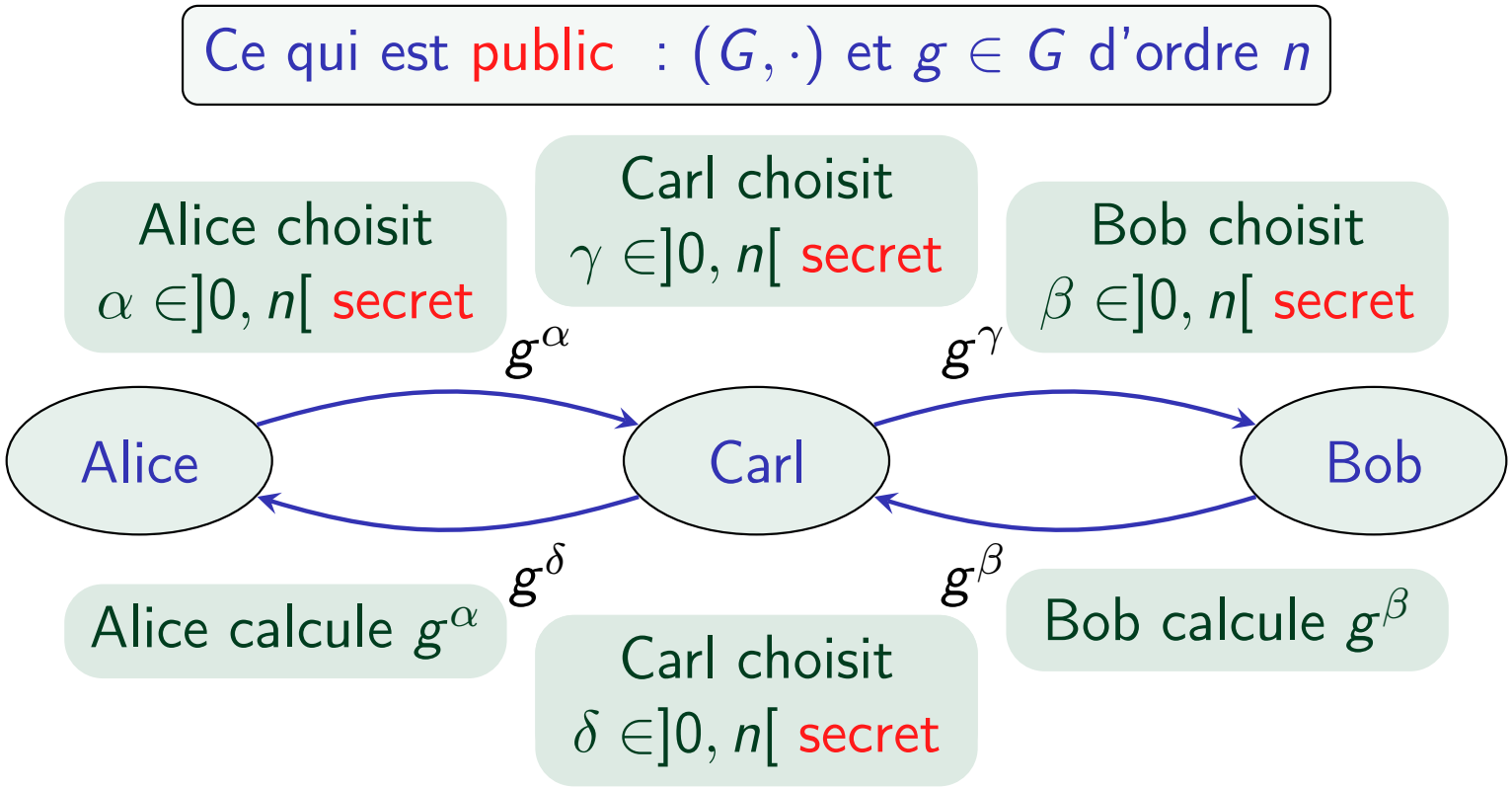
L'attaque "man in the middle"



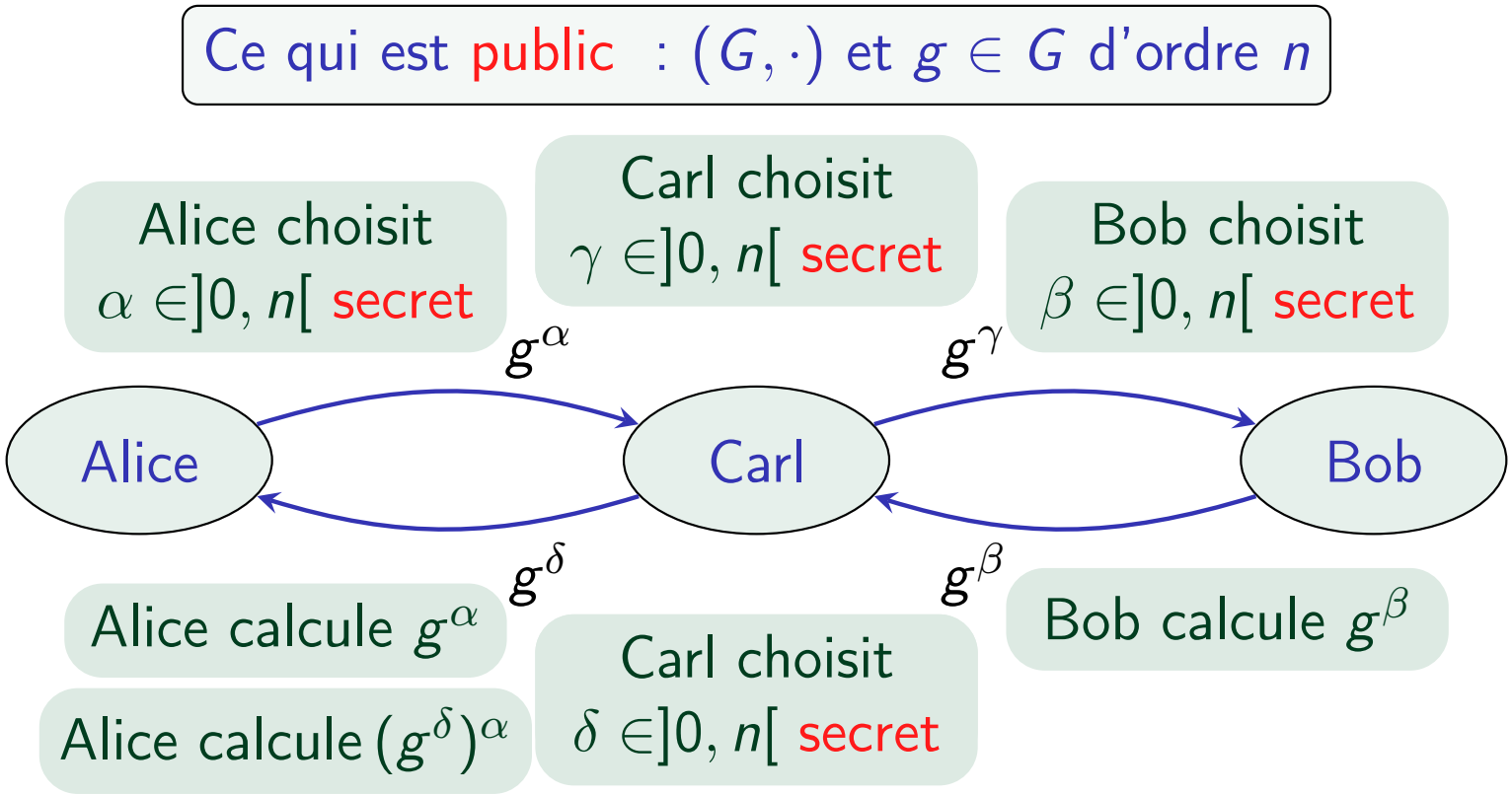
L'attaque "man in the middle"



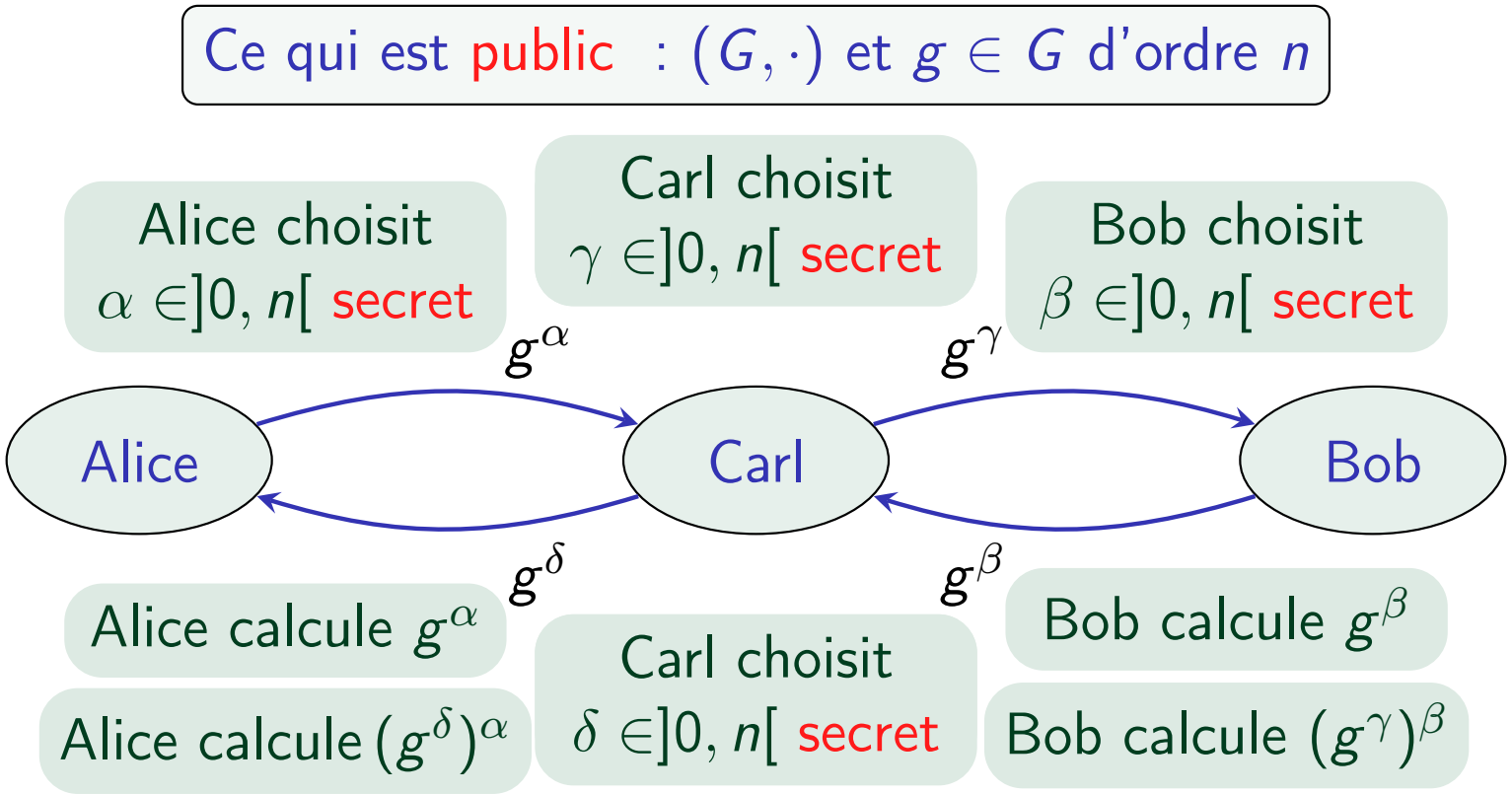
L'attaque "man in the middle"



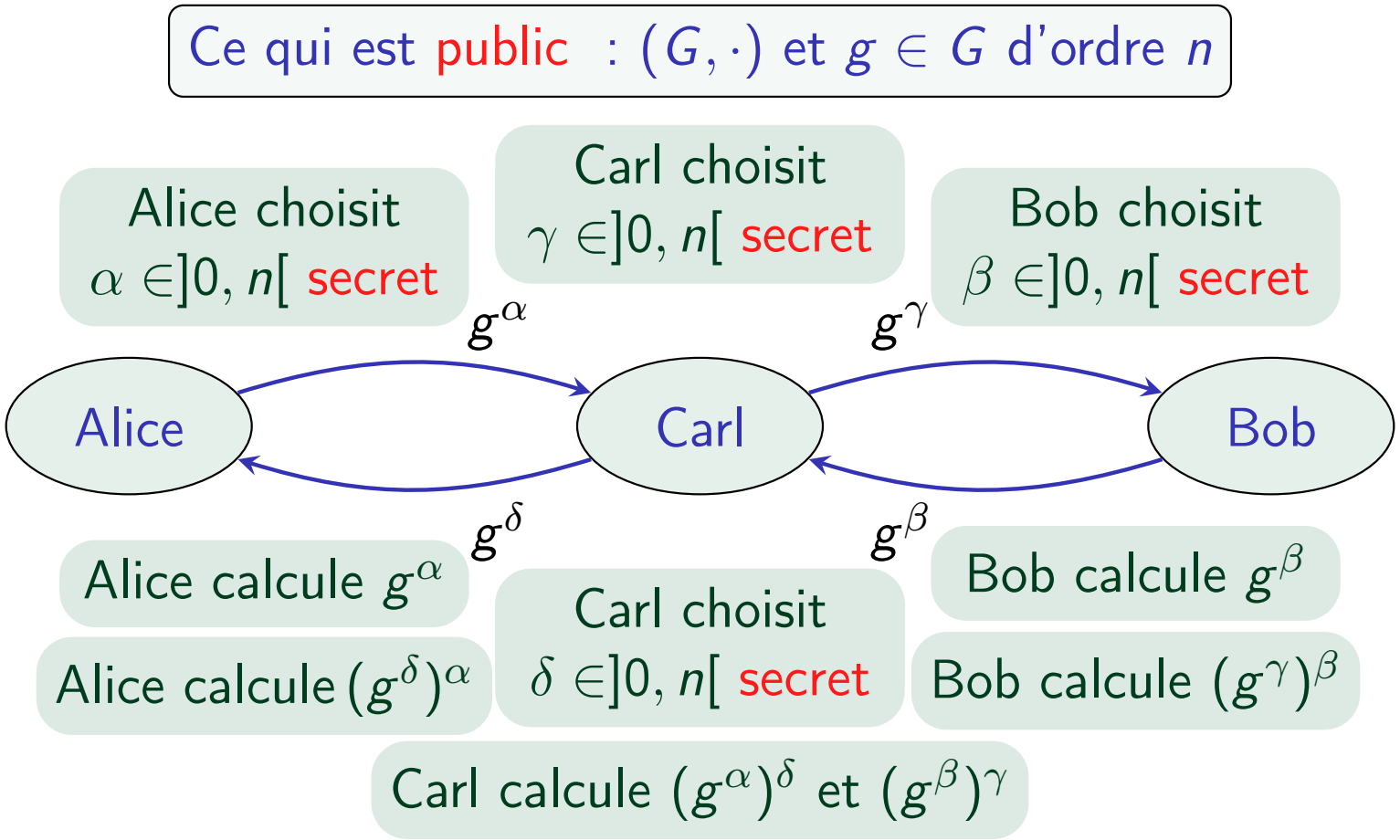
L'attaque "man in the middle"



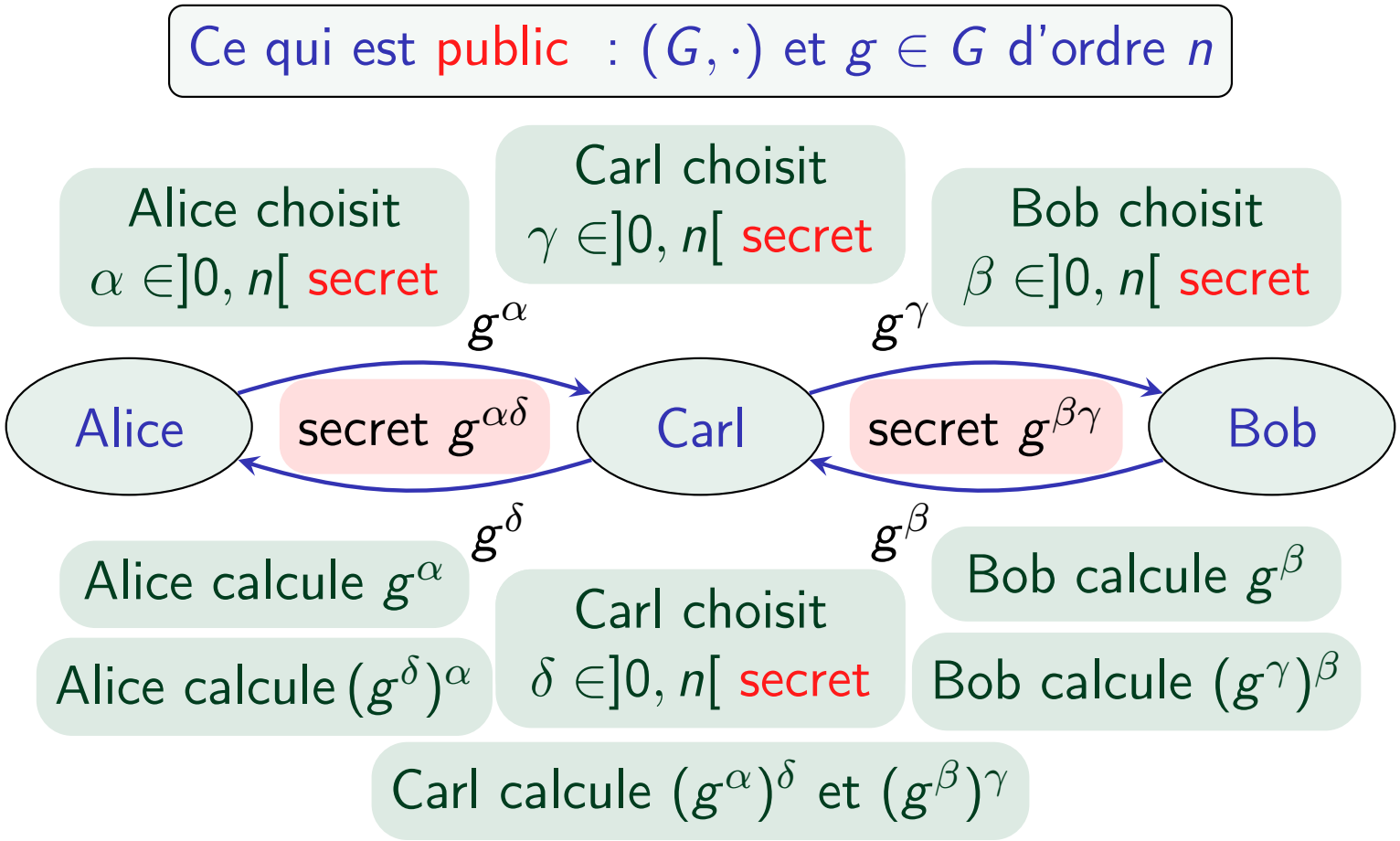
L'attaque "man in the middle"



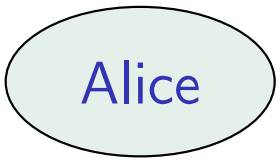
L'attaque "man in the middle"

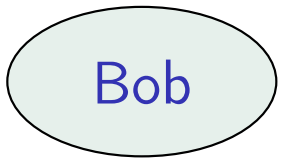


L'attaque "man in the middle"



Le schéma du protocole sécurisé avec une signature RSA





Le schéma du protocole sécurisé avec une signature RSA

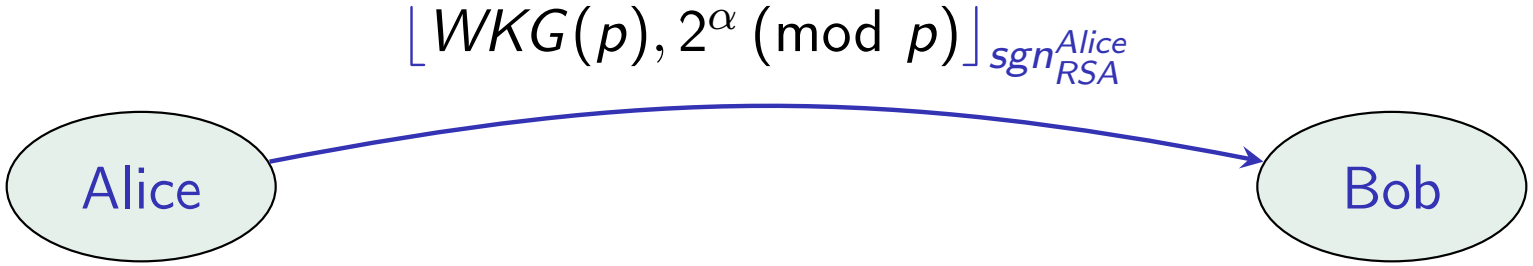
Alice choisit un groupe $WKG(p)$ et $\alpha \in [1, q-1]$

Alice

Bob

Le schéma du protocole sécurisé avec une signature RSA

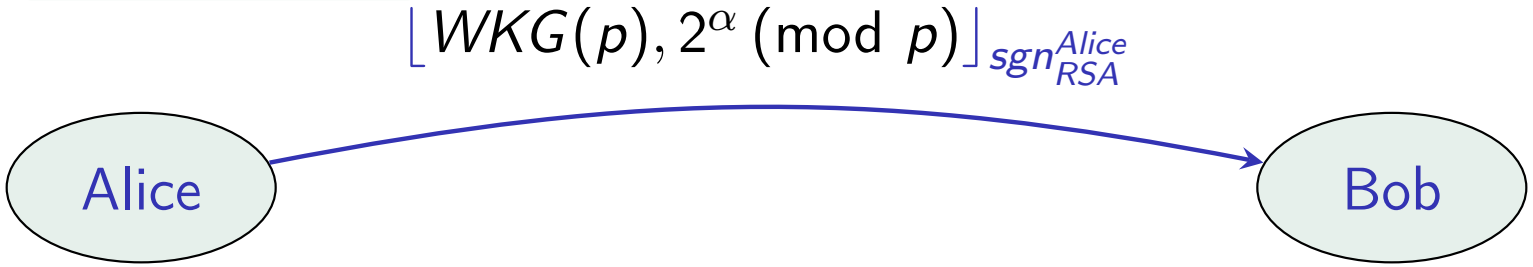
Alice choisit un groupe $WKG(p)$ et $\alpha \in [1, q-1]$



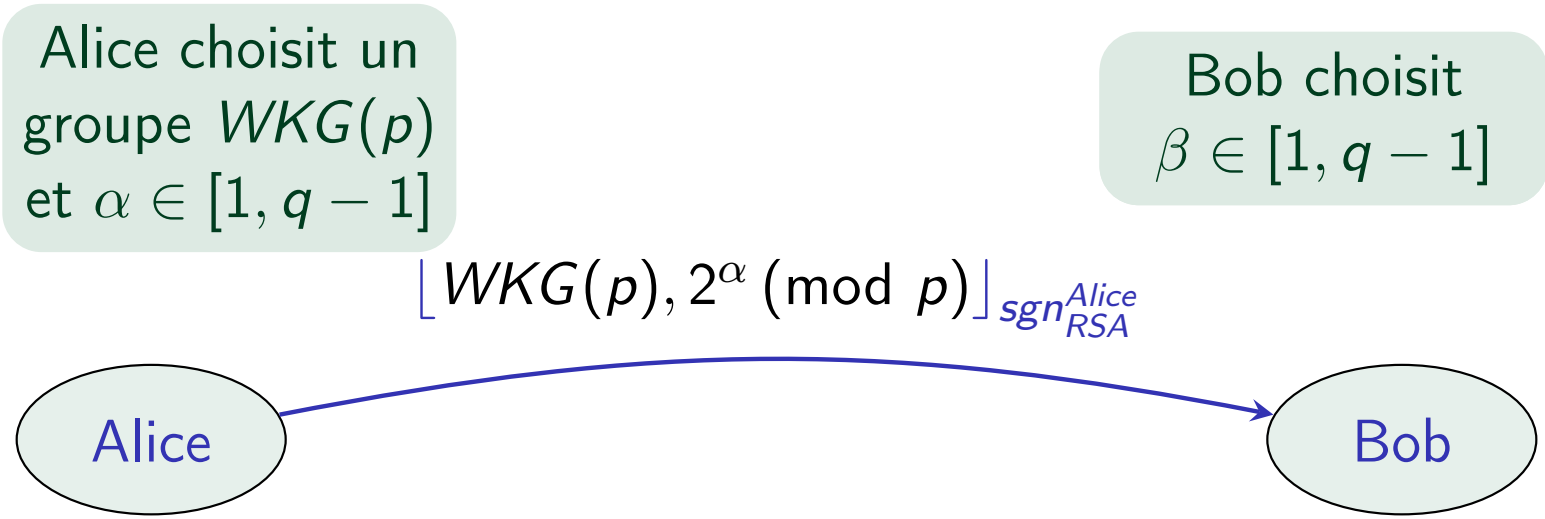
Le schéma du protocole sécurisé avec une signature RSA

Alice choisit un groupe $WKG(p)$ et $\alpha \in [1, q-1]$

Bob vérifie la signature



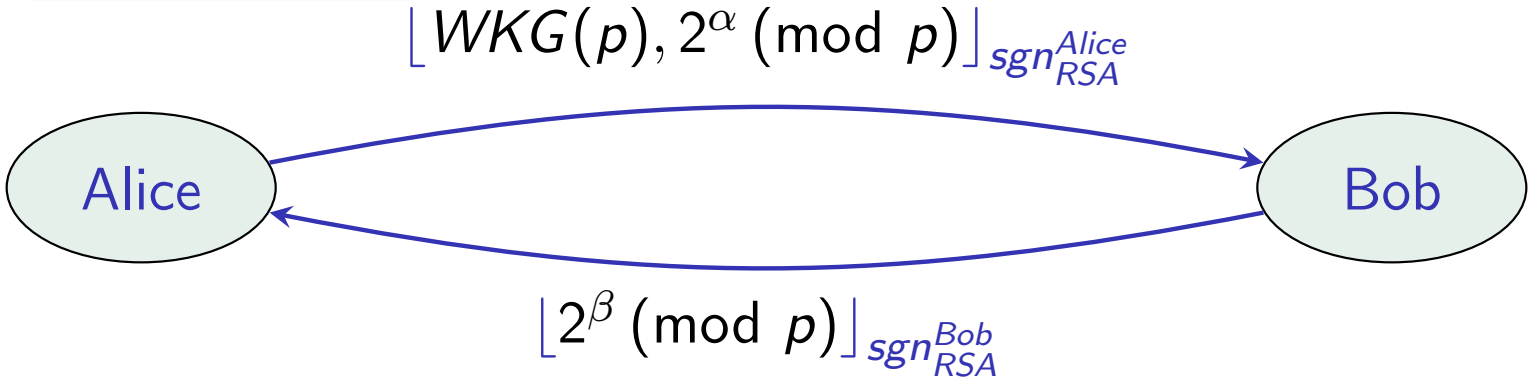
Le schéma du protocole sécurisé avec une signature RSA



Le schéma du protocole sécurisé avec une signature RSA

Alice choisit un groupe $WKG(p)$ et $\alpha \in [1, q-1]$

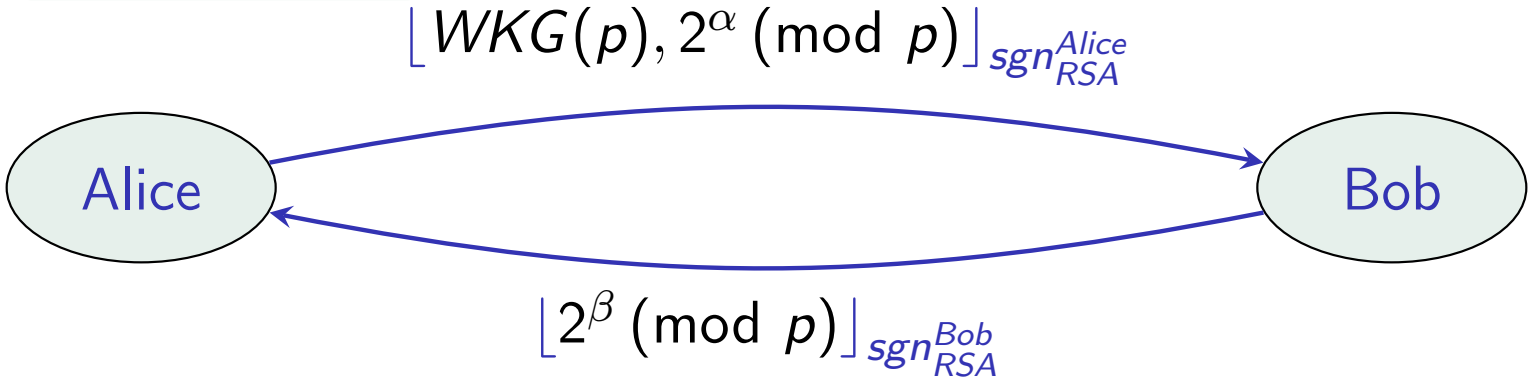
Bob choisit $\beta \in [1, q-1]$



Le schéma du protocole sécurisé avec une signature RSA

Alice choisit un groupe $WKG(p)$ et $\alpha \in [1, q-1]$

Bob choisit $\beta \in [1, q-1]$

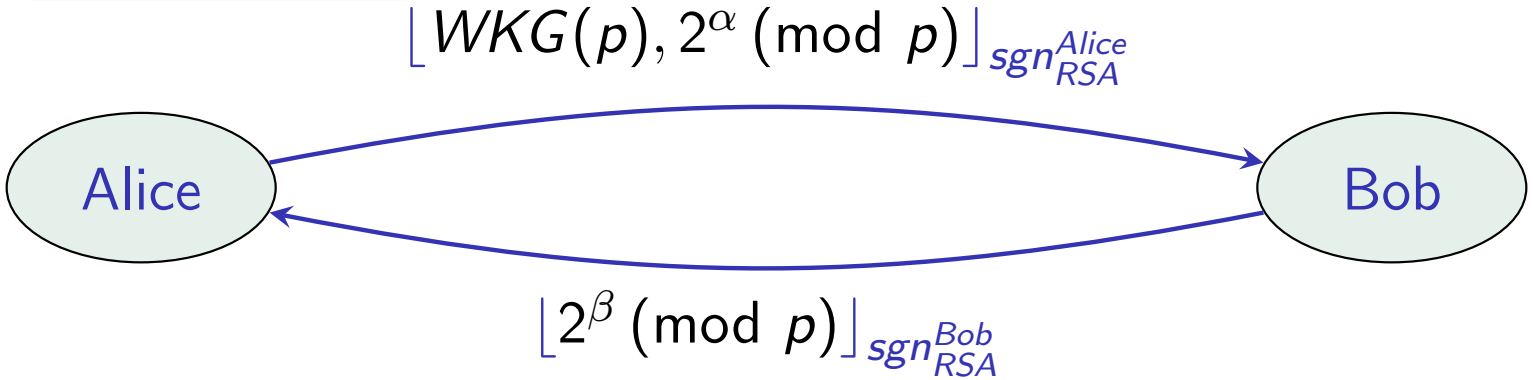


Alice vérifie la signature

Le schéma du protocole sécurisé avec une signature RSA

Alice choisit un groupe $WKG(p)$ et $\alpha \in [1, q-1]$

Bob choisit $\beta \in [1, q-1]$

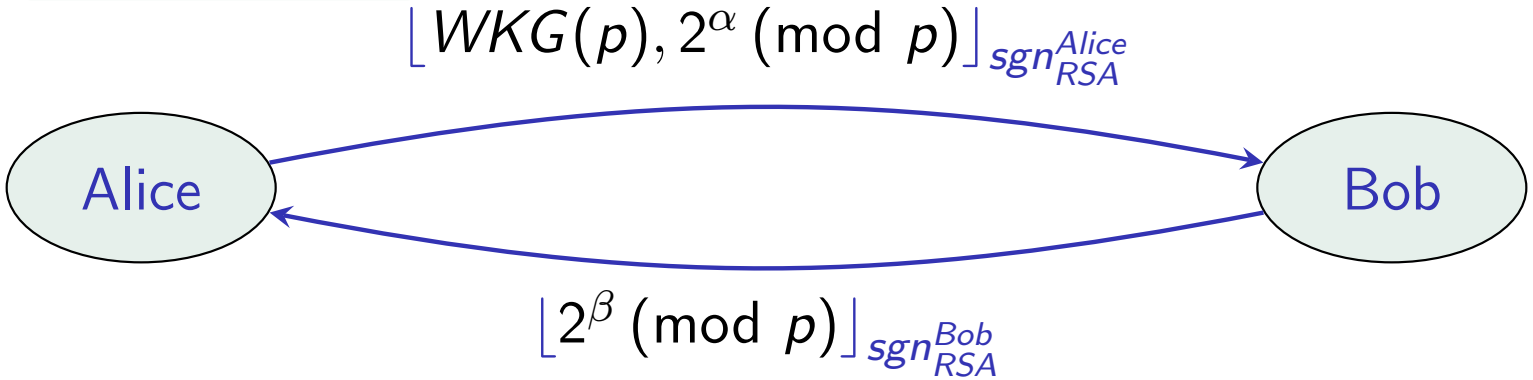


Alice calcule $(2^{\beta})^{\alpha} = 2^{\alpha\beta}$

Le schéma du protocole sécurisé avec une signature RSA

Alice choisit un groupe $WKG(p)$ et $\alpha \in [1, q-1]$

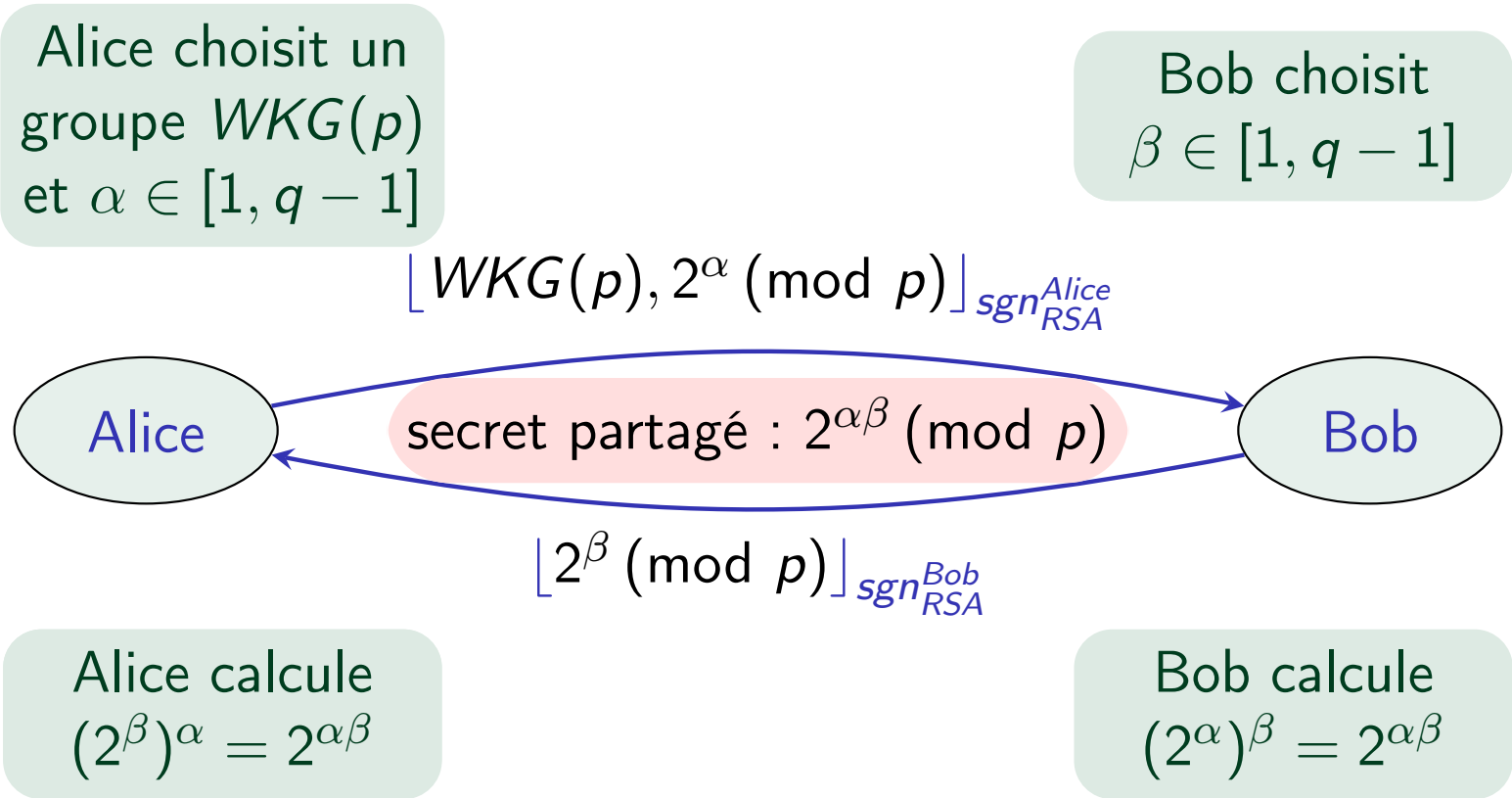
Bob choisit $\beta \in [1, q-1]$



Alice calcule $(2^{\beta})^{\alpha} = 2^{\alpha\beta}$

Bob calcule $(2^{\alpha})^{\beta} = 2^{\alpha\beta}$

Le schéma du protocole sécurisé avec une signature RSA



Quelques remarques sur le schéma précédent

- Le précédent schéma est encore trop abstrait et incomplet si l'on a en vue une implémentation concrète.
- Dans le monde internet, cette implémentation sera sous la forme d'un **client/serveur** (UDP par exemple), le serveur étant sur la machine de *Bob* vu qu'*Alice* est l'initiatrice du protocole.
- Il faut que *Bob* ait la possibilité de refuser le groupe de Diffie-Hellman proposé par *Alice*. Par exemple, *Bob* n'accepte que les $WKG(p)$ tels que $\log_2(p) \geq 2048$.
- On appliquera une fonction de hachage au secret partagé $2^{\alpha\beta} \pmod{p}$, par exemple : SHA_{256}, le secret partagé devenant alors : $\mathrm{SHA}_{256}(2^{\alpha\beta} \pmod{p})$. Ceci permet d'effacer toute trace de la construction mathématique du secret partagé initial.
- Pour contrecarrer les **attaques par rejeu**, il faudra aussi introduire un peu d'aléatoire sous forme d'un *nonce* (Number Only Once).

Quelques remarques sur le schéma précédent

- Le précédent schéma est encore trop abstrait et incomplet si l'on a en vue une implémentation concrète.
- Dans le monde internet, cette implémentation sera sous la forme d'un **client/serveur** (UDP par exemple), le serveur étant sur la machine de *Bob* vu qu'*Alice* est l'initiatrice du protocole.
- Il faut que *Bob* ait la possibilité de refuser le groupe de Diffie-Hellman proposé par *Alice*. Par exemple, *Bob* n'accepte que les $WKG(p)$ tels que $\log_2(p) \geq 2048$.
- On appliquera une fonction de hachage au secret partagé $2^{\alpha\beta} \pmod p$, par exemple : SHA_{256}, le secret partagé devenant alors : $\mathrm{SHA}_{256}(2^{\alpha\beta} \pmod p)$. Ceci permet d'effacer toute trace de la construction mathématique du secret partagé initial.
- Pour contrecarrer les **attaques par rejeu**, il faudra aussi introduire un peu d'aléatoire sous forme d'un *nonce* (Number Only Once).

Quelques remarques sur le schéma précédent

- Le précédent schéma est encore trop abstrait et incomplet si l'on a en vue une implémentation concrète.
- Dans le monde internet, cette implémentation sera sous la forme d'un **client/serveur** (UDP par exemple), le serveur étant sur la machine de *Bob* vu qu'*Alice* est l'initiatrice du protocole.
- Il faut que *Bob* ait la possibilité de refuser le groupe de Diffie-Hellman proposé par *Alice*. Par exemple, *Bob* n'accepte que les $WKG(p)$ tels que $\log_2(p) \geq 2048$.
- On appliquera une fonction de hachage au secret partagé $2^{\alpha\beta} \pmod p$, par exemple : SHA_{256}, le secret partagé devenant alors : $\mathrm{SHA}_{256}(2^{\alpha\beta} \pmod p)$. Ceci permet d'effacer toute trace de la construction mathématique du secret partagé initial.
- Pour contrecarrer les **attaques par rejeu**, il faudra aussi introduire un peu d'aléatoire sous forme d'un *nonce* (Number Only Once).

Quelques remarques sur le schéma précédent

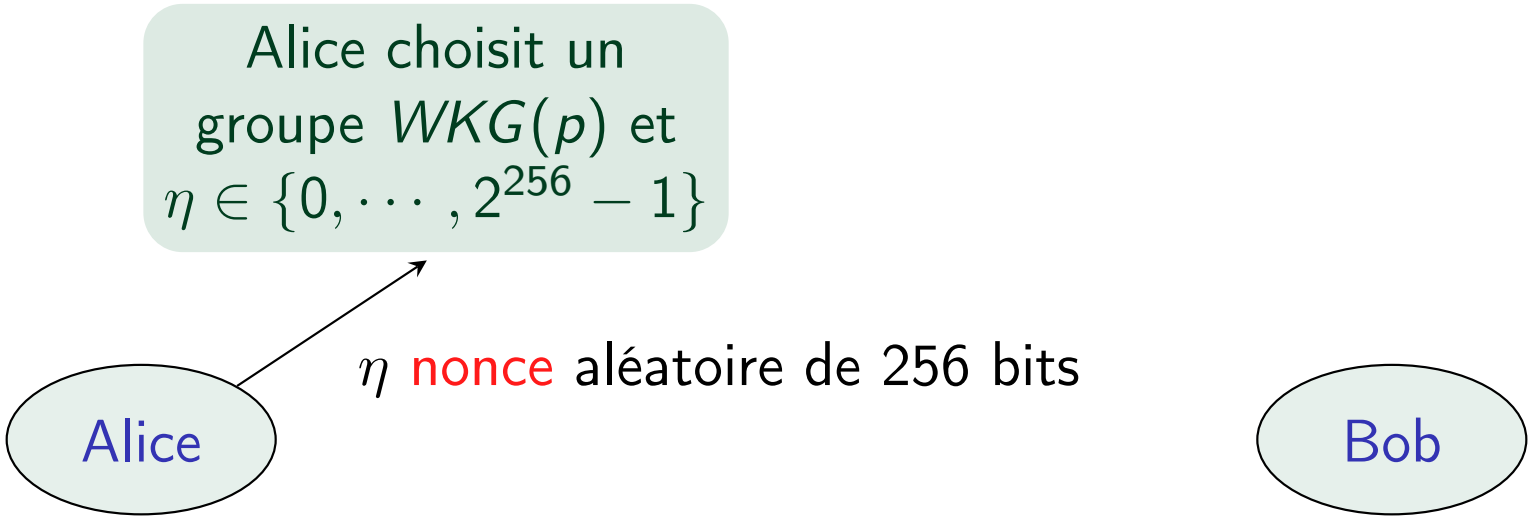
- Le précédent schéma est encore trop abstrait et incomplet si l'on a en vue une implémentation concrète.
- Dans le monde internet, cette implémentation sera sous la forme d'un **client/serveur** (UDP par exemple), le serveur étant sur la machine de *Bob* vu qu'*Alice* est l'initiatrice du protocole.
- Il faut que *Bob* ait la possibilité de refuser le groupe de Diffie-Hellman proposé par *Alice*. Par exemple, *Bob* n'accepte que les $WKG(p)$ tels que $\log_2(p) \geq 2048$.
- On appliquera une fonction de hachage au secret partagé $2^{\alpha\beta} \pmod p$, par exemple : SHA_{256}, le secret partagé devenant alors : $\mathrm{SHA}_{256}(2^{\alpha\beta} \pmod p)$. Ceci permet d'effacer toute trace de la construction mathématique du secret partagé initial.
- Pour contrecarrer les **attaques par rejeu**, il faudra aussi introduire un peu d'aléatoire sous forme d'un *nonce* (Number Only Once).

Quelques remarques sur le schéma précédent

- Le précédent schéma est encore trop abstrait et incomplet si l'on a en vue une implémentation concrète.
- Dans le monde internet, cette implémentation sera sous la forme d'un **client/serveur** (UDP par exemple), le serveur étant sur la machine de *Bob* vu qu'*Alice* est l'initiatrice du protocole.
- Il faut que *Bob* ait la possibilité de refuser le groupe de Diffie-Hellman proposé par *Alice*. Par exemple, *Bob* n'accepte que les $WKG(p)$ tels que $\log_2(p) \geq 2048$.
- On appliquera une fonction de hachage au secret partagé $2^{\alpha\beta} \pmod{p}$, par exemple : SHA_{256}, le secret partagé devenant alors : $\mathrm{SHA}_{256}(2^{\alpha\beta} \pmod{p})$. Ceci permet d'effacer toute trace de la construction mathématique du secret partagé initial.
- Pour contrecarrer les **attaques par rejeu**, il faudra aussi introduire un peu d'aléatoire sous forme d'un *nonce* (Number Only Once).

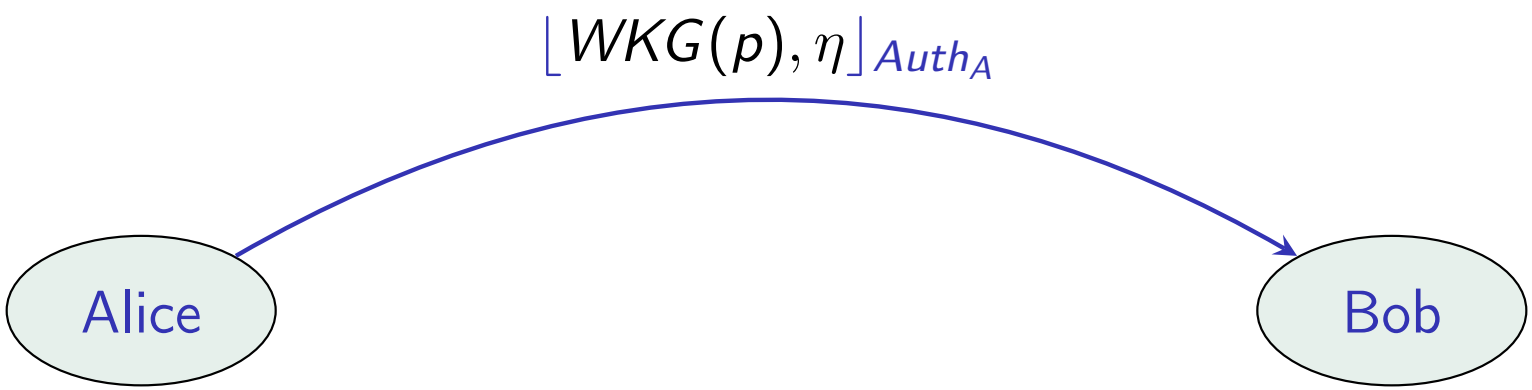
Le schéma final du protocole

Le schéma final du protocole

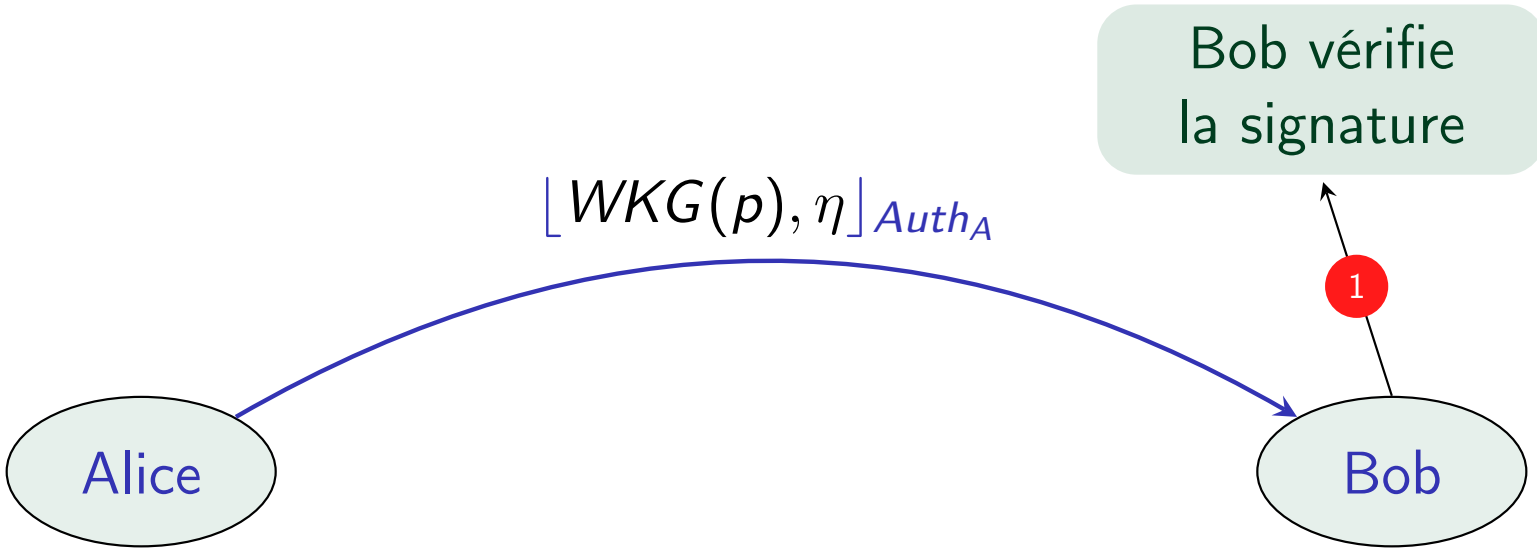


Le schéma final du protocole

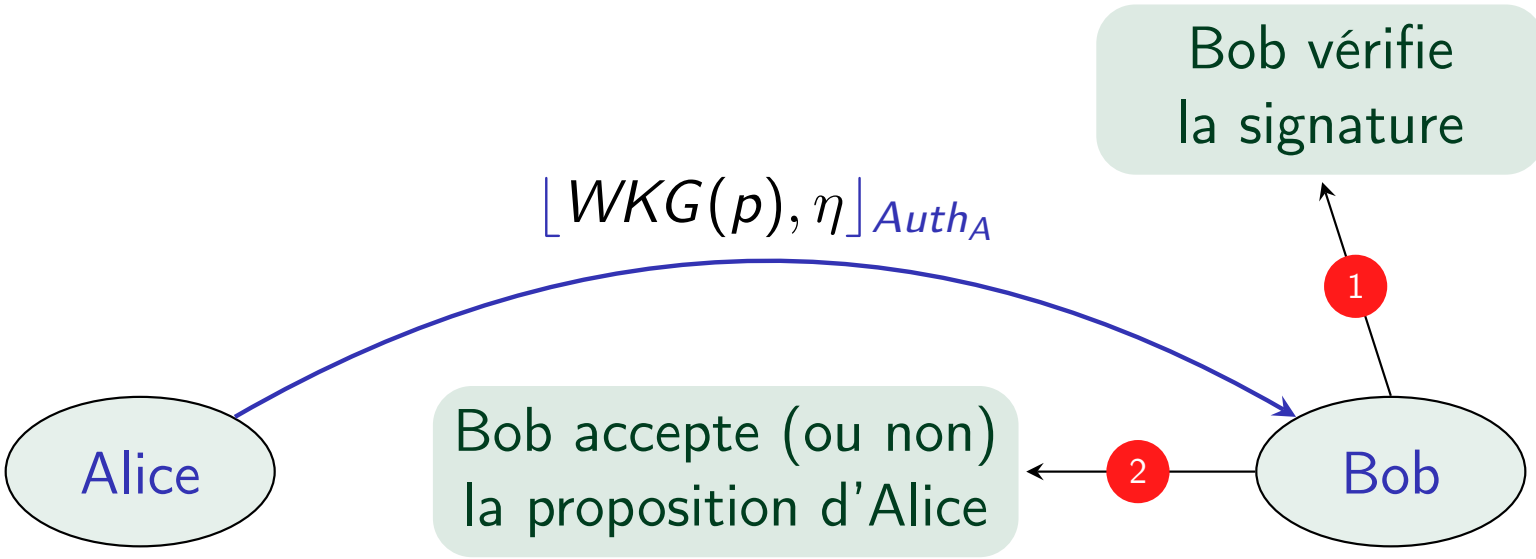
$$Auth_A = sgn_{RSA}^{Alice}(WKG(p) \parallel \eta)$$



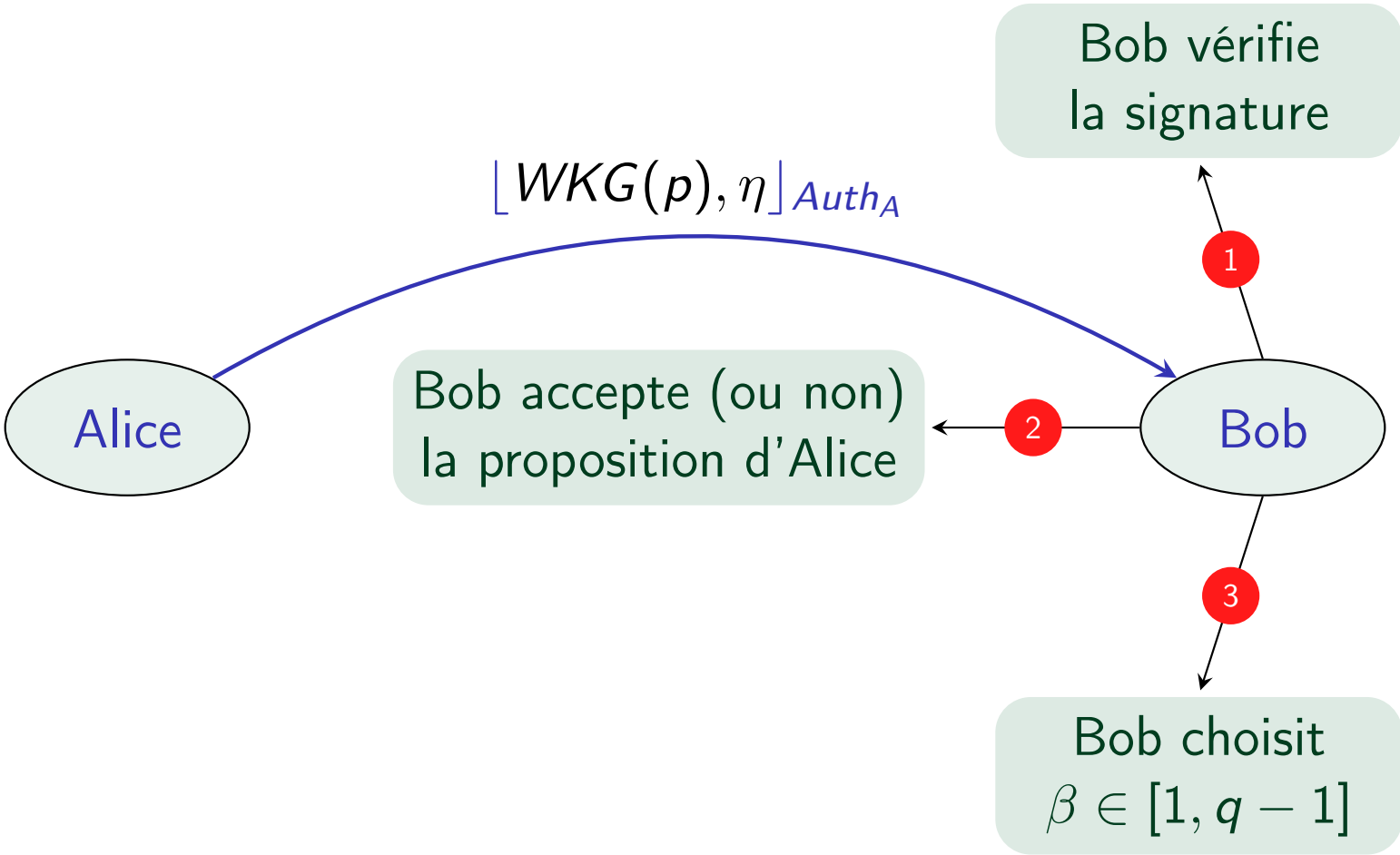
Le schéma final du protocole



Le schéma final du protocole

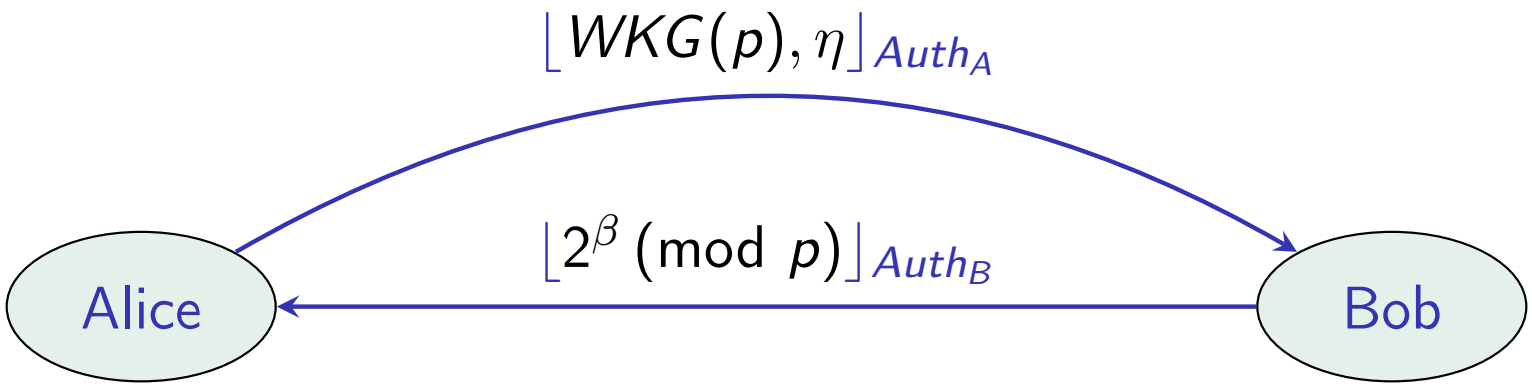


Le schéma final du protocole

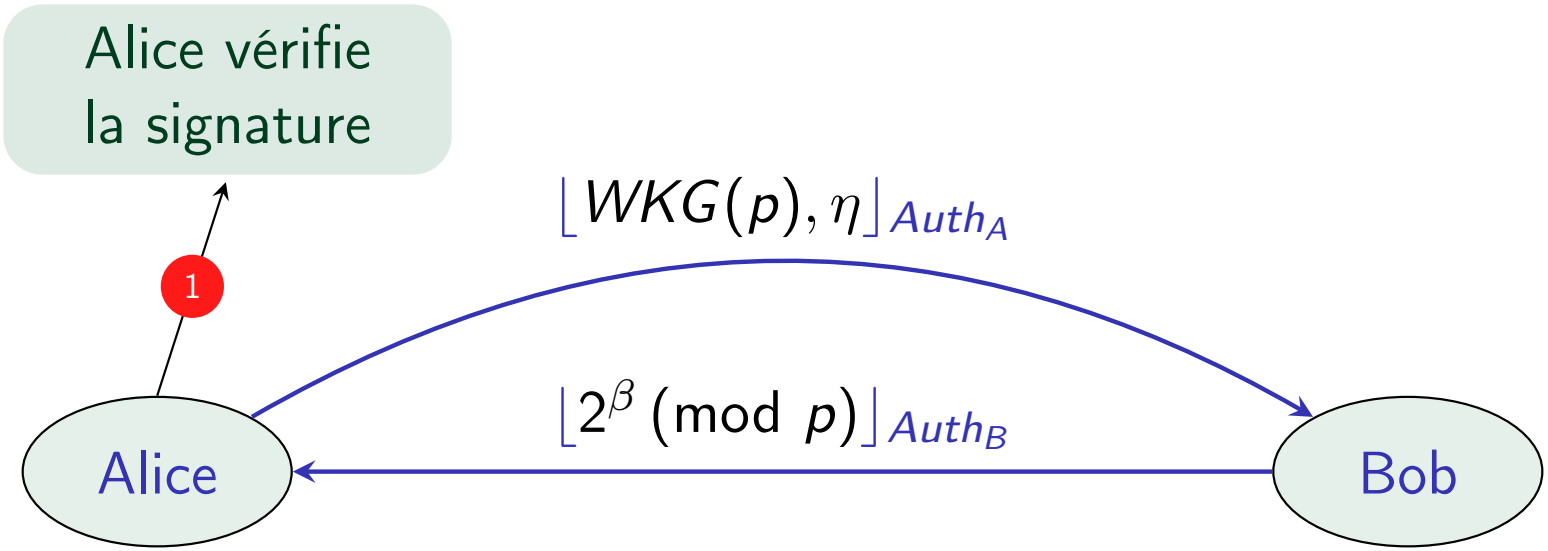


Le schéma final du protocole

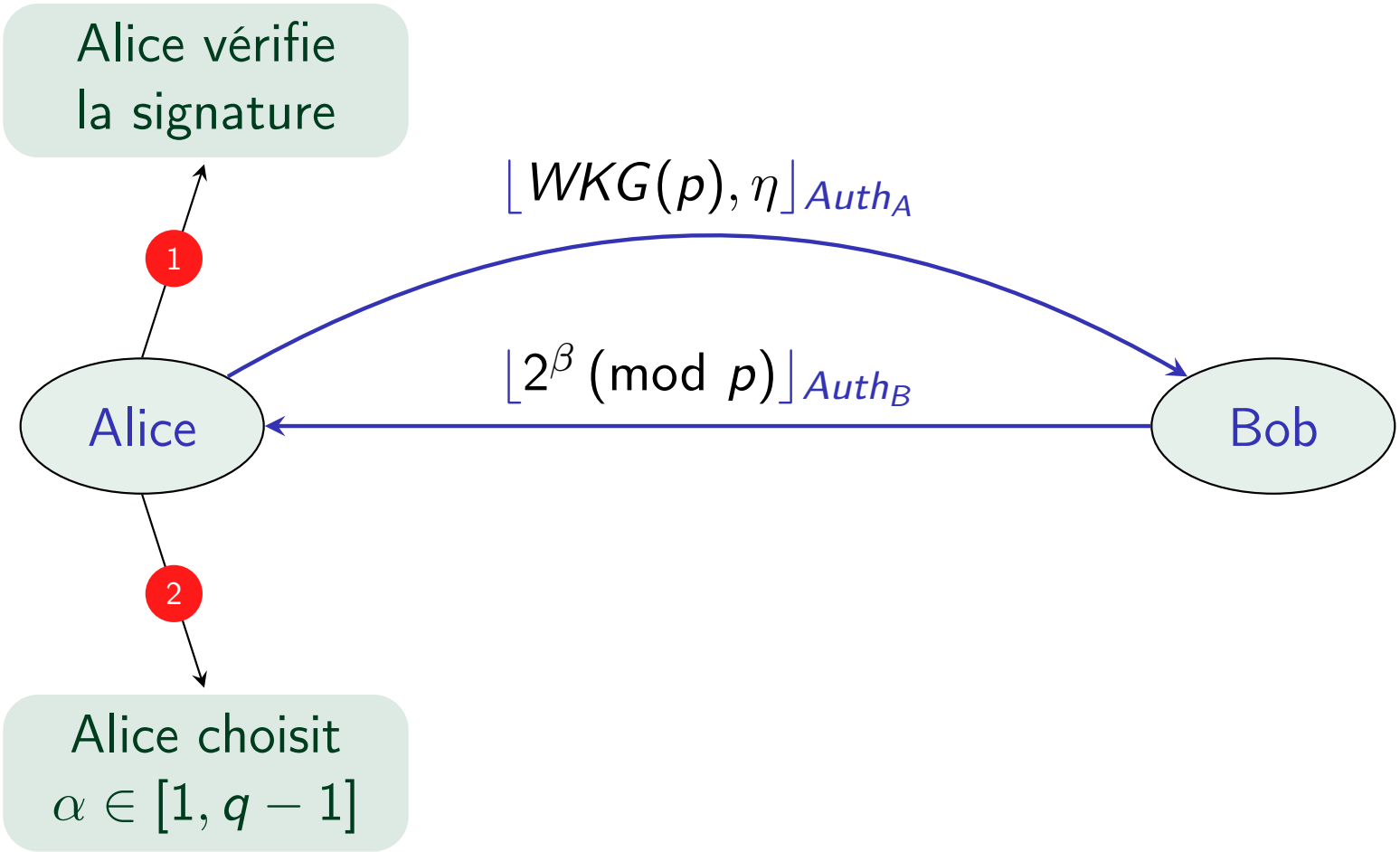
$$Auth_B = sgn_{RSA}^{Bob}(WKG(p) \parallel \eta \parallel 2^{\beta} \pmod{p})$$



Le schéma final du protocole

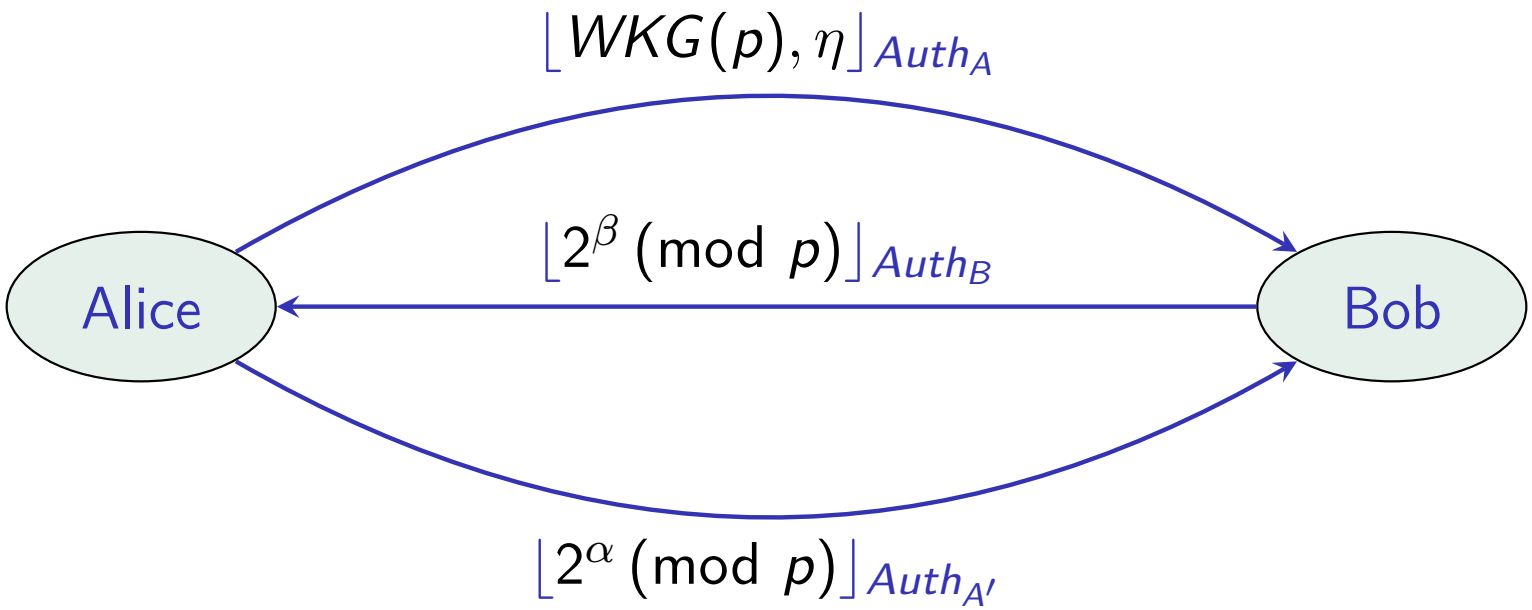


Le schéma final du protocole

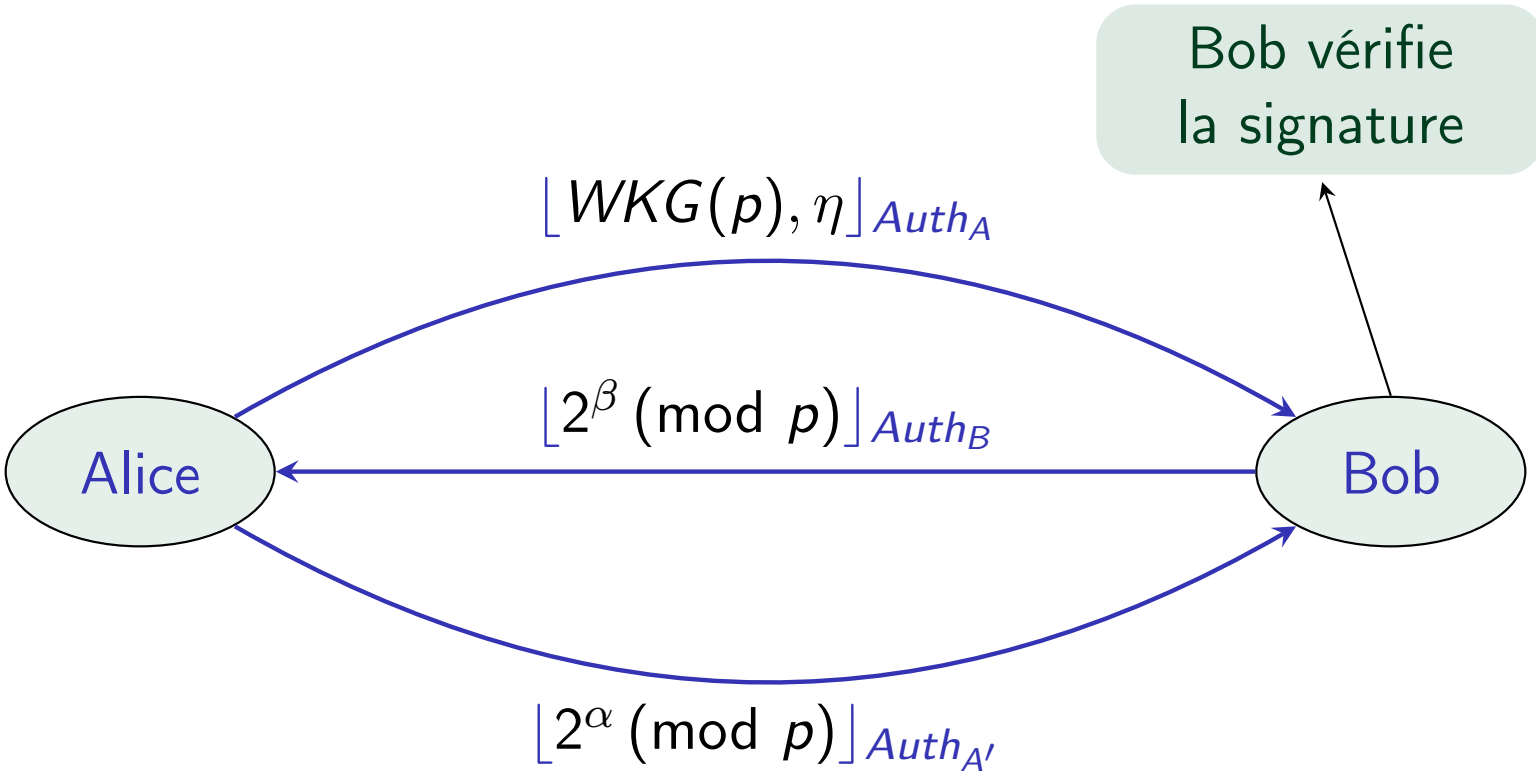


Le schéma final du protocole

$$Auth_{A'} = sgn_{RSA}^{Alice}(WKG(p) \parallel \eta \parallel 2^{\beta} \pmod{p} \parallel 2^{\alpha} \pmod{p})$$

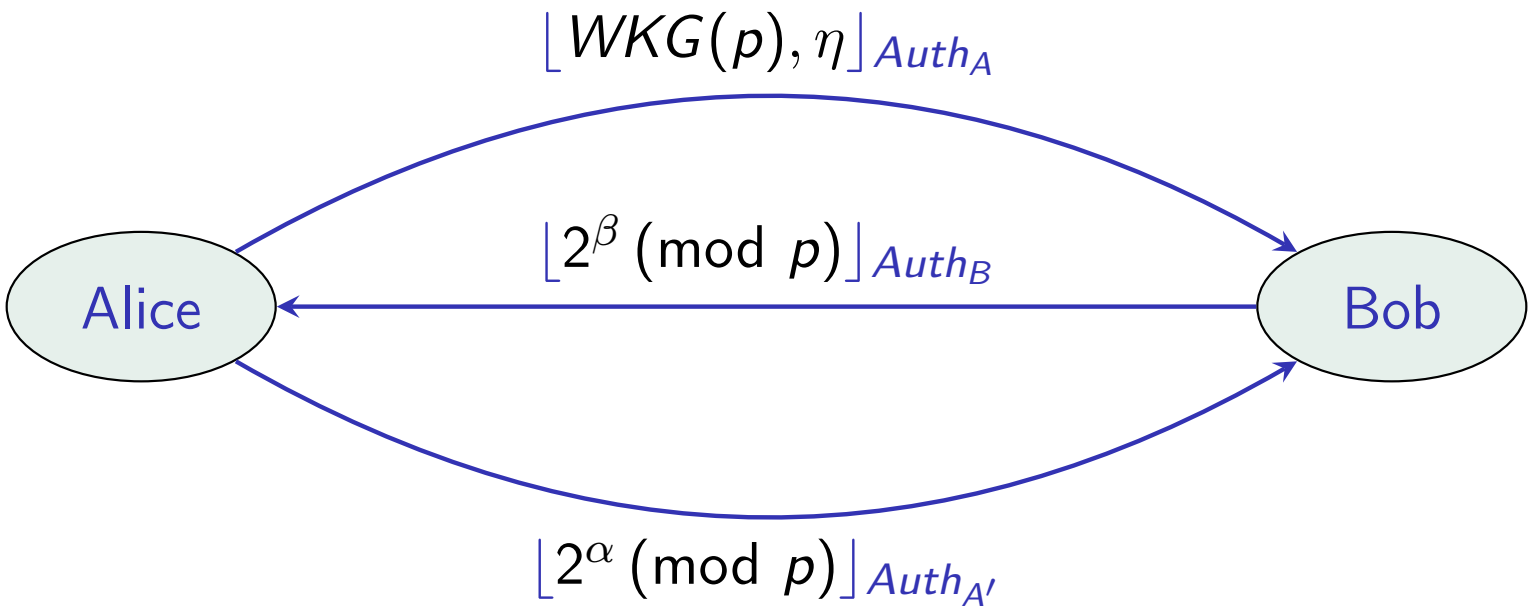


Le schéma final du protocole



Le schéma final du protocole

Secret partagé : $\mathrm{SHA}_{256}(2^{\alpha\beta} \pmod p)$



Quelques remarques complémentaires

- Les secrets partagés devraient être renouvelés périodiquement. Par conséquent, il faut prévoir dans le protocole précédent la possibilité d'associer au secret partagé une **durée de vie**.
  Par exemple, *Alice* propose une durée de vie finie ou non (dans ce dernier cas, par convention, la durée de vie est égale à 0) et *Bob* accepte ou non cette proposition.
- Une fois le secret partagé établi, il faut pouvoir le **stocker de manière sûre** chez chacun des protagonistes. Un tel dispositif a été normalisé dans la RFC-2367 (*PF_KEY Key Management API, Version 2*) et est utilisé, entre autres, par le protocole *IPsec*.
- Il faut définir une politique de **gestion des incidents et erreurs** lors de l'exécution du protocole en particulier du côté du serveur (ici *Bob*). Par exemple, il n'est pas forcément recommandé, pour le serveur, de renvoyer un message d'erreur sur chaque incident avec un client : risque de *déni de service*. Autre exemple : faut-il logger ou non les incidents ou erreurs ? Et cætera.

Quelques remarques complémentaires

- Les secrets partagés devraient être renouvelés périodiquement. Par conséquent, il faut prévoir dans le protocole précédent la possibilité d'associer au secret partagé une **durée de vie**.
  Par exemple, *Alice* propose une durée de vie finie ou non (dans ce dernier cas, par convention, la durée de vie est égale à 0) et *Bob* accepte ou non cette proposition.
- Une fois le secret partagé établi, il faut pouvoir le **stocker de manière sûre** chez chacun des protagonistes. Un tel dispositif a été normalisé dans la ▸ RFC-2367 (*PF_KEY Key Management API, Version 2*) et est utilisé, entre autres, par le protocole *IPsec*.
- Il faut définir une politique de **gestion des incidents et erreurs** lors de l'exécution du protocole en particulier du côté du serveur (ici *Bob*). Par exemple, il n'est pas forcément recommandé, pour le serveur, de renvoyer un message d'erreur sur chaque incident avec un client : risque de *déni de service*. Autre exemple : faut-il logger ou non les incidents ou erreurs ? Et cætera.

Quelques remarques complémentaires

- Les secrets partagés devraient être renouvelés périodiquement. Par conséquent, il faut prévoir dans le protocole précédent la possibilité d'associer au secret partagé une **durée de vie**.
  Par exemple, *Alice* propose une durée de vie finie ou non (dans ce dernier cas, par convention, la durée de vie est égale à 0) et *Bob* accepte ou non cette proposition.
- Une fois le secret partagé établi, il faut pouvoir le **stocker de manière sûre** chez chacun des protagonistes. Un tel dispositif a été normalisé dans la ▸ RFC-2367 (*PF_KEY Key Management API, Version 2*) et est utilisé, entre autres, par le protocole *IPsec*.
- Il faut définir une politique de **gestion des incidents et erreurs** lors de l'exécution du protocole en particulier du côté du serveur (ici *Bob*). Par exemple, il n'est pas forcément recommandé, pour le serveur, de renvoyer un message d'erreur sur chaque incident avec un client : risque de *déni de service*. Autre exemple : faut-il logger ou non les incidents ou erreurs ? Et cætera.

Quelques généralités

- Une courbe elliptique est un objet mathématique sur lequel existe une structure "naturelle" de groupe, cette structure étant construite géométriquement.
- L'idée d'utiliser ce groupe en cryptographie est due indépendamment à *Koblitz* et *Miller* (1985).
- Ces groupes seront, en particulier, utilisés dans le cadre du protocole de Diffie-Hellman, la résolution du *problème du logarithme discret* étant encore plus ardue qu'avec les groupes classiques $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$.
- Attention : les courbes elliptiques n'ont rien à voir (ou presque) avec les ellipses.
- Les courbes elliptiques apparaissent dans nombre de domaines mathématiques et sont parmi les objets mathématiques les plus étudiés.

Quelques généralités

- Une courbe elliptique est un objet mathématique sur lequel existe une structure "naturelle" de groupe, cette structure étant construite géométriquement.
- L'idée d'utiliser ce groupe en cryptographie est due indépendamment à *Koblitz* et *Miller* (1985).
- Ces groupes seront, en particulier, utilisés dans le cadre du protocole de Diffie-Hellman, la résolution du *problème du logarithme discret* étant encore plus ardue qu'avec les groupes classiques $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$.
- Attention : les courbes elliptiques n'ont rien à voir (ou presque) avec les ellipses.
- Les courbes elliptiques apparaissent dans nombre de domaines mathématiques et sont parmi les objets mathématiques les plus étudiés.

Quelques généralités

- Une courbe elliptique est un objet mathématique sur lequel existe une structure "naturelle" de groupe, cette structure étant construite géométriquement.
- L'idée d'utiliser ce groupe en cryptographie est due indépendamment à *Koblitz* et *Miller* (1985).
- Ces groupes seront, en particulier, utilisés dans le cadre du protocole de Diffie-Hellman, la résolution du *problème du logarithme discret* étant encore plus ardue qu'avec les groupes classiques $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$.
- Attention : les courbes elliptiques n'ont rien à voir (ou presque) avec les ellipses.
- Les courbes elliptiques apparaissent dans nombre de domaines mathématiques et sont parmi les objets mathématiques les plus étudiés.

Quelques généralités

- Une courbe elliptique est un objet mathématique sur lequel existe une structure "naturelle" de groupe, cette structure étant construite géométriquement.
- L'idée d'utiliser ce groupe en cryptographie est due indépendamment à *Koblitz* et *Miller* (1985).
- Ces groupes seront, en particulier, utilisés dans le cadre du protocole de Diffie-Hellman, la résolution du *problème du logarithme discret* étant encore plus ardue qu'avec les groupes classiques $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$.
- Attention : les courbes elliptiques n'ont rien à voir (ou presque) avec les ellipses.
- Les courbes elliptiques apparaissent dans nombre de domaines mathématiques et sont parmi les objets mathématiques les plus étudiés.

Quelques généralités

- Une courbe elliptique est un objet mathématique sur lequel existe une structure "naturelle" de groupe, cette structure étant construite géométriquement.
- L'idée d'utiliser ce groupe en cryptographie est due indépendamment à *Koblitz* et *Miller* (1985).
- Ces groupes seront, en particulier, utilisés dans le cadre du protocole de Diffie-Hellman, la résolution du *problème du logarithme discret* étant encore plus ardue qu'avec les groupes classiques $((\mathbb{Z}/p\mathbb{Z})^*, \cdot)$.
- Attention : les courbes elliptiques n'ont rien à voir (ou presque) avec les ellipses.
- Les courbes elliptiques apparaissent dans nombre de domaines mathématiques et sont parmi les objets mathématiques les plus étudiés.

Définition et premiers résultats

Définition (courbe elliptique)

Une courbe elliptique E définie sur un corps commutatif k est une cubique lisse d'équation de Weierstrass :

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

La lissité de E signifie qu'en tout point (x, y) de E la tangente est définie : si $(a, b) \in E$ et $P(x, y) = y^2 + a_1xy + a_3y - x^3 + a_2x^2 + a_4x + a_6$, alors $\frac{\partial P}{\partial x}(a, b) \neq 0$ et $\frac{\partial P}{\partial y}(a, b) \neq 0$.

Proposition (équation réduite)

On montre que, si le corps k est de caractéristique différente de 2 et 3, alors l'équation précédente peut se réduire à :

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

Définition et premiers résultats

Définition (courbe elliptique)

Une courbe elliptique E définie sur un corps commutatif k est une cubique lisse d'équation de Weierstrass :

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

La lissité de E signifie qu'en tout point (x, y) de E la tangente est définie : si $(a, b) \in E$ et $P(x, y) = y^2 + a_1xy + a_3y - x^3 + a_2x^2 + a_4x + a_6$, alors $\frac{\partial P}{\partial x}(a, b) \neq 0$ et $\frac{\partial P}{\partial y}(a, b) \neq 0$.

Proposition (équation réduite)

On montre que, si le corps k est de caractéristique différente de 2 et 3, alors l'équation précédente peut se réduire à :

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

Définition et premiers résultats

Définition (courbe elliptique)

Une courbe elliptique E définie sur un corps commutatif k est une cubique lisse d'équation de Weierstrass :

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

La lissité de E signifie qu'en tout point (x, y) de E la tangente est définie : si $(a, b) \in E$ et $P(x, y) = y^2 + a_1xy + a_3y - x^3 + a_2x^2 + a_4x + a_6$, alors $\frac{\partial P}{\partial x}(a, b) \neq 0$ et $\frac{\partial P}{\partial y}(a, b) \neq 0$.

Proposition (équation réduite)

On montre que, si le corps k est de caractéristique différente de 2 et 3, alors l'équation précédente peut se réduire à :

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

Définition et premiers résultats

Définition (discriminant)

Le nombre $\Delta(E) = -16(4a^3 + 27b^2)$ *est appelé discriminant de la courbe* $E : y^2 = x^3 + ax + b$.

Proposition

La courbe E *est une courbe elliptique si et seulement si* $\Delta(E) \neq 0$.

Pour pouvoir définir une loi de groupe sur E, on aura besoin de lui ajouter un point O appelé *point à l'infini*. Donc, ce que nous appelerons courbe elliptique sera l'ensemble suivant :

$$E(k) = \{(x, y) \in k^2 \mid y^2 = x^3 + ax + b\} \cup \{O\} \quad \text{avec } \Delta(E) \neq 0$$

Avant de définir la loi de groupe de E, nous allons voir quelques exemples de cubiques lorsque $k = \mathbb{R}$.

Définition et premiers résultats

Définition (discriminant)

Le nombre $\Delta(E) = -16(4a^3 + 27b^2)$ *est appelé discriminant de la courbe* $E : y^2 = x^3 + ax + b$.

Proposition

La courbe E *est une courbe elliptique si et seulement si* $\Delta(E) \neq 0$.

Pour pouvoir définir une loi de groupe sur E, on aura besoin de lui ajouter un point O appelé *point à l'infini*. Donc, ce que nous appelerons courbe elliptique sera l'ensemble suivant :

$$E(k) = \{(x, y) \in k^2 \mid y^2 = x^3 + ax + b\} \cup \{O\} \quad \text{avec } \Delta(E) \neq 0$$

Avant de définir la loi de groupe de E, nous allons voir quelques exemples de cubiques lorsque $k = \mathbb{R}$.

Définition et premiers résultats

Définition (discriminant)

Le nombre $\Delta(E) = -16(4a^3 + 27b^2)$ *est appelé discriminant de la courbe* $E : y^2 = x^3 + ax + b$.

Proposition

La courbe E *est une courbe elliptique si et seulement si* $\Delta(E) \neq 0$.

Pour pouvoir définir une loi de groupe sur E, on aura besoin de lui ajouter un point O appelé *point à l'infini*. Donc, ce que nous appelerons courbe elliptique sera l'ensemble suivant :

$$E(k) = \{(x, y) \in k^2 \mid y^2 = x^3 + ax + b\} \cup \{O\} \quad \text{avec } \Delta(E) \neq 0$$

Avant de définir la loi de groupe de E, nous allons voir quelques exemples de cubiques lorsque $k = \mathbb{R}$.

Définition et premiers résultats

Définition (discriminant)

Le nombre $\Delta(E) = -16(4a^3 + 27b^2)$ *est appelé discriminant de la courbe* $E : y^2 = x^3 + ax + b$.

Proposition

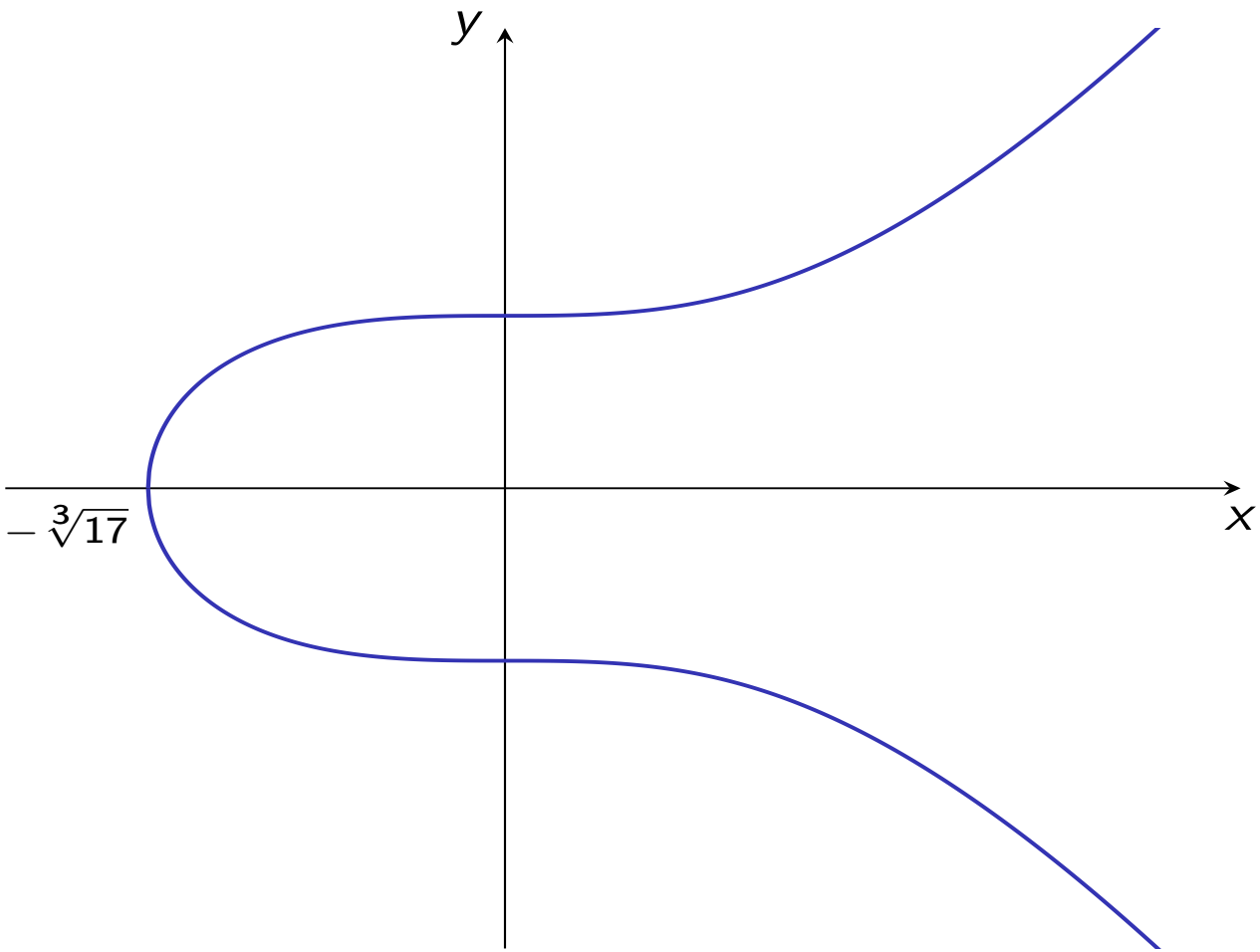
La courbe E *est une courbe elliptique si et seulement si* $\Delta(E) \neq 0$.

Pour pouvoir définir une loi de groupe sur E, on aura besoin de lui ajouter un point O appelé *point à l'infini*. Donc, ce que nous appelerons courbe elliptique sera l'ensemble suivant :

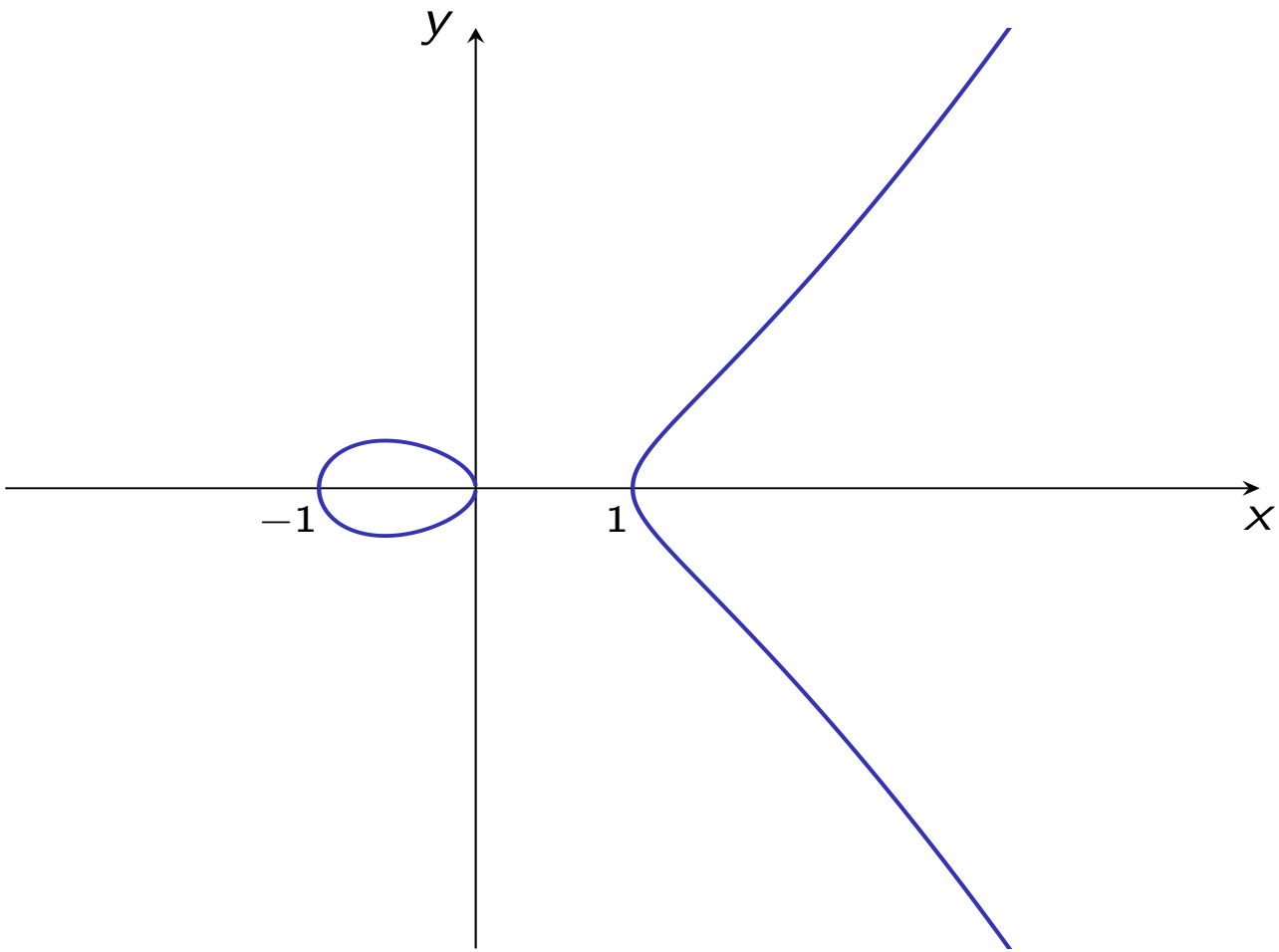
$$E(k) = \{(x, y) \in k^2 \mid y^2 = x^3 + ax + b\} \cup \{O\} \quad \text{avec } \Delta(E) \neq 0$$

Avant de définir la loi de groupe de E, nous allons voir quelques exemples de cubiques lorsque $k = \mathbb{R}$.

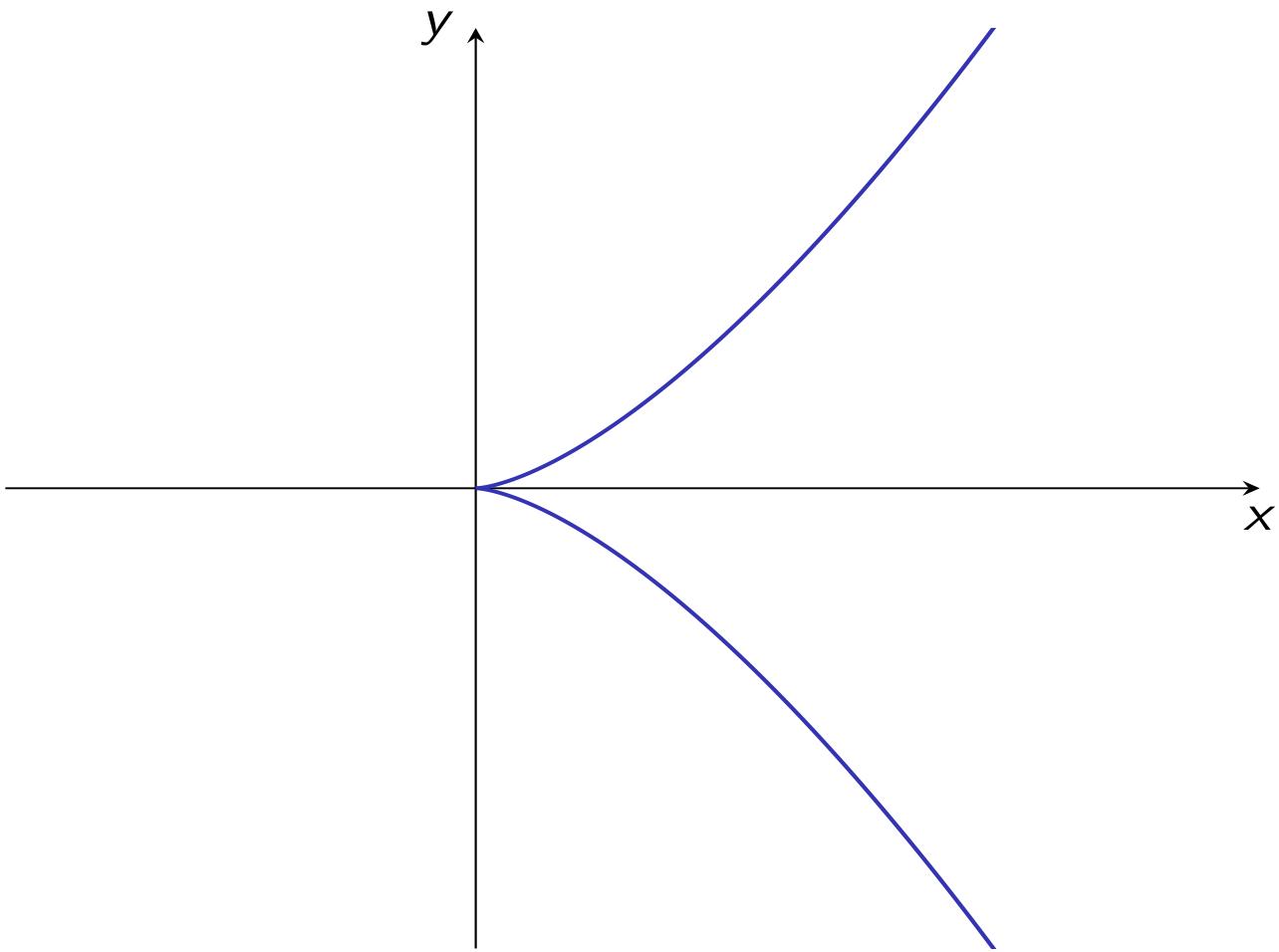
La courbe elliptique : $y^2 = x^3 + 17$



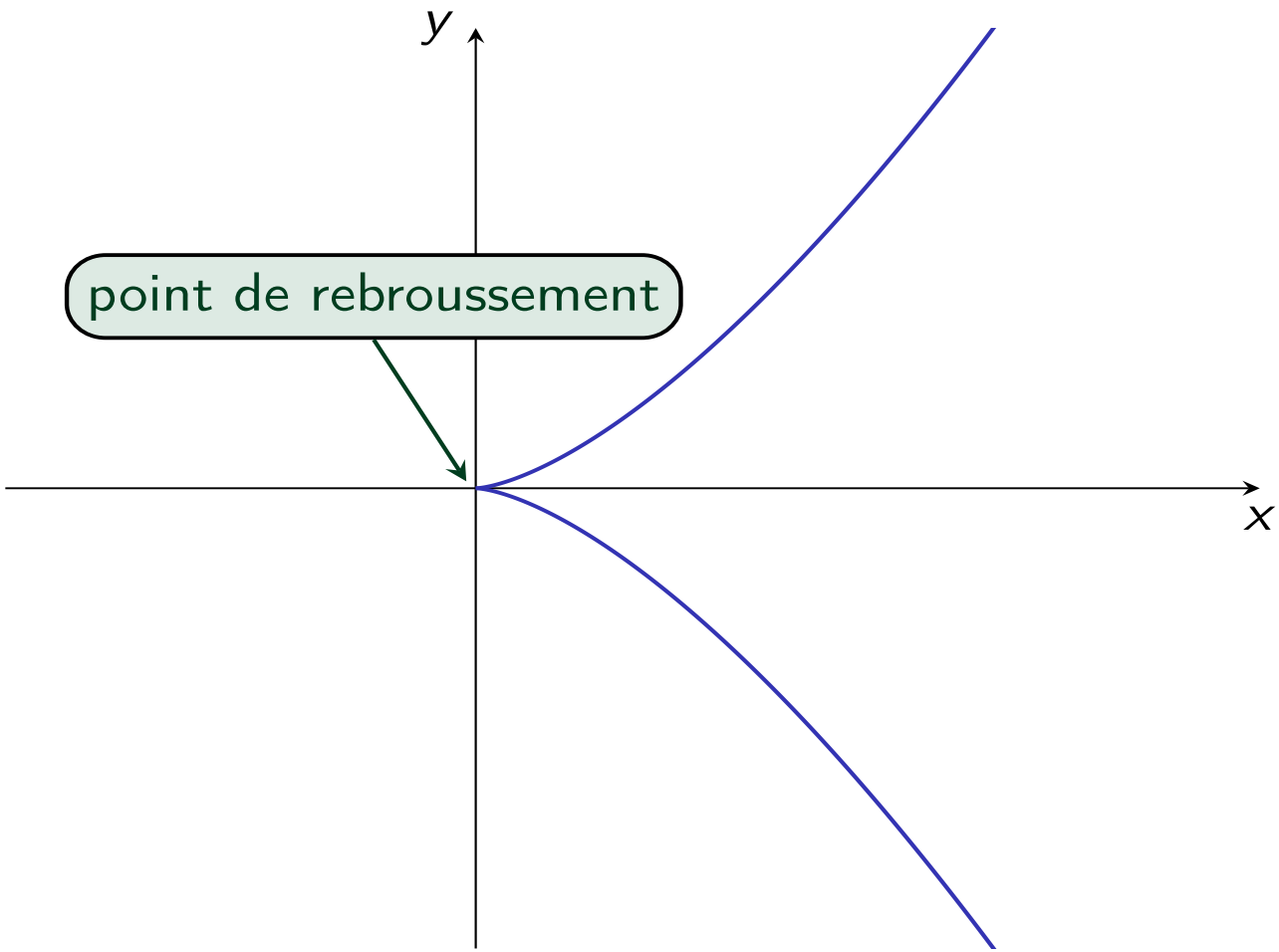
La courbe elliptique : $y^2 = x^3 - x$



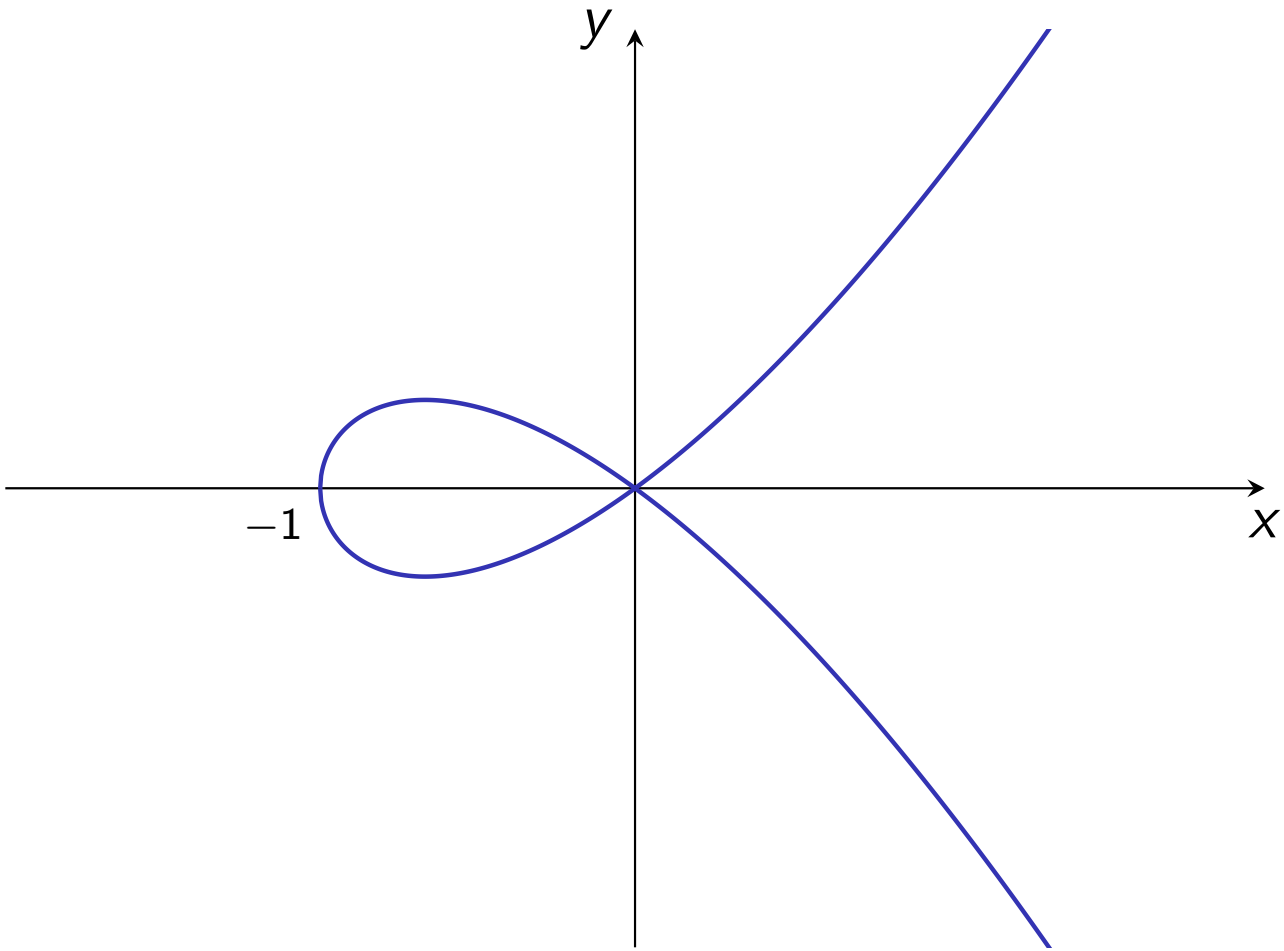
La cubique singulière : $y^2 = x^3$



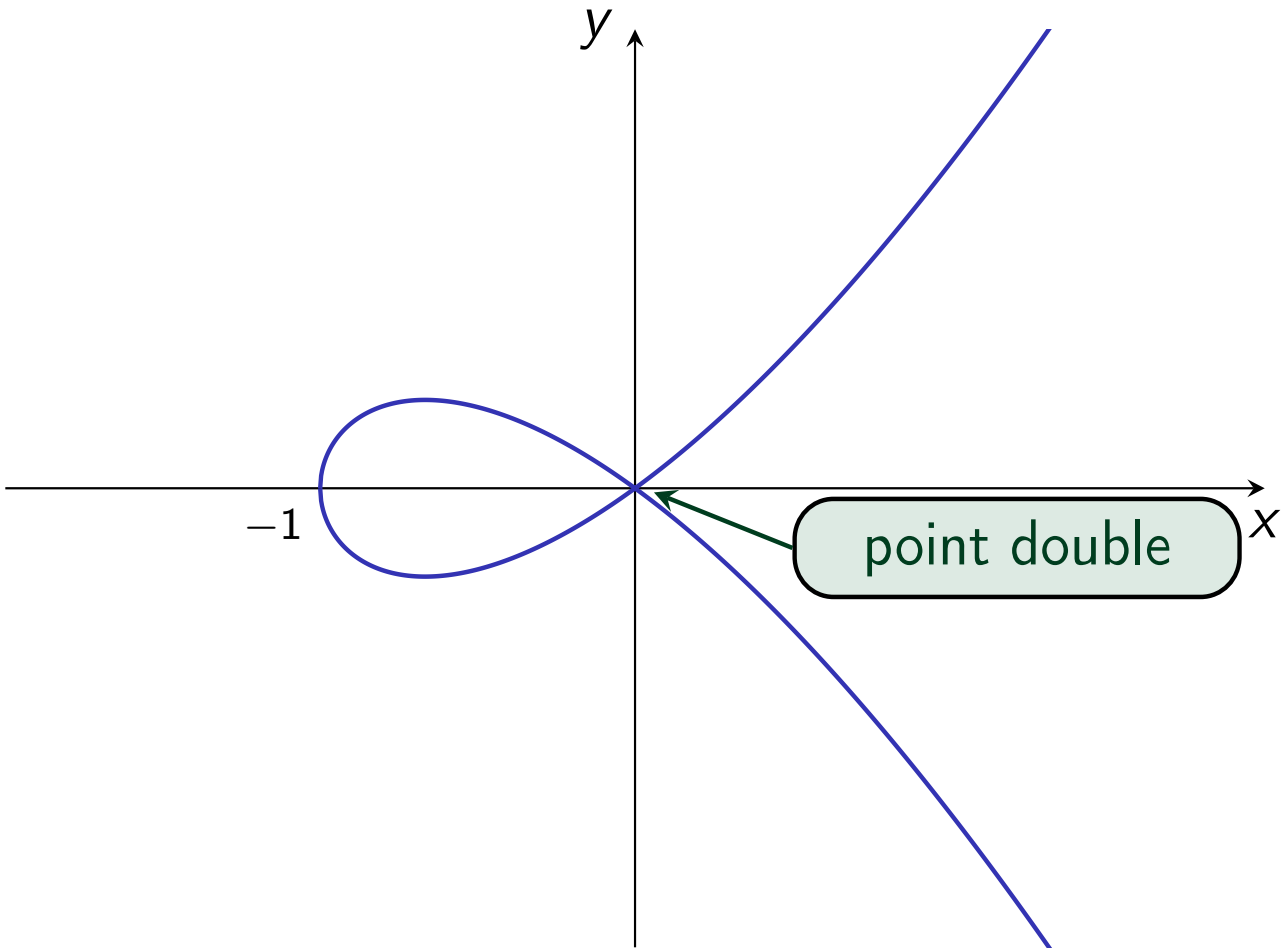
La cubique singulière : $y^2 = x^3$



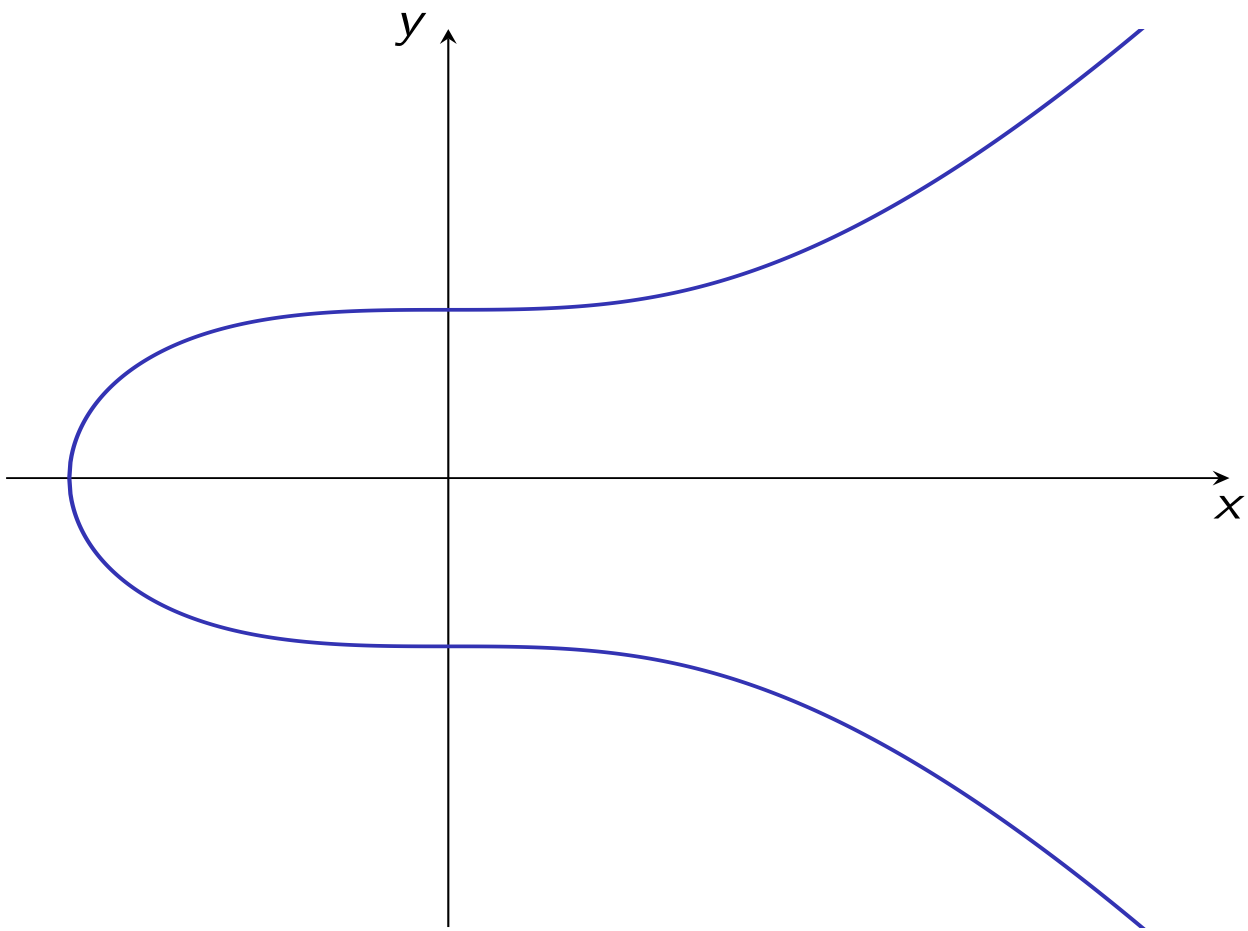
La cubique singulière : $y^2 = x^2(x+1)$



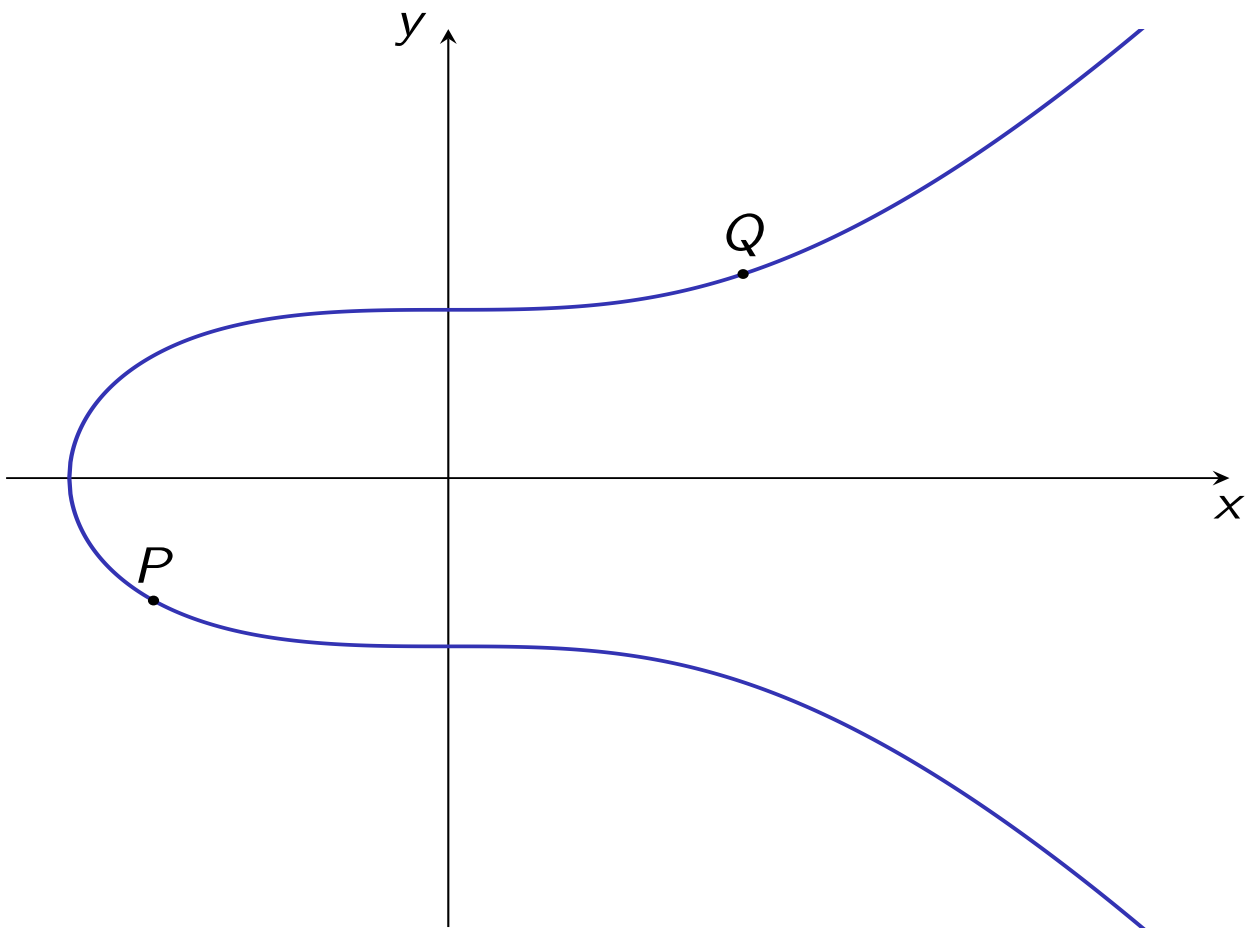
La cubique singulière : $y^2 = x^2(x+1)$



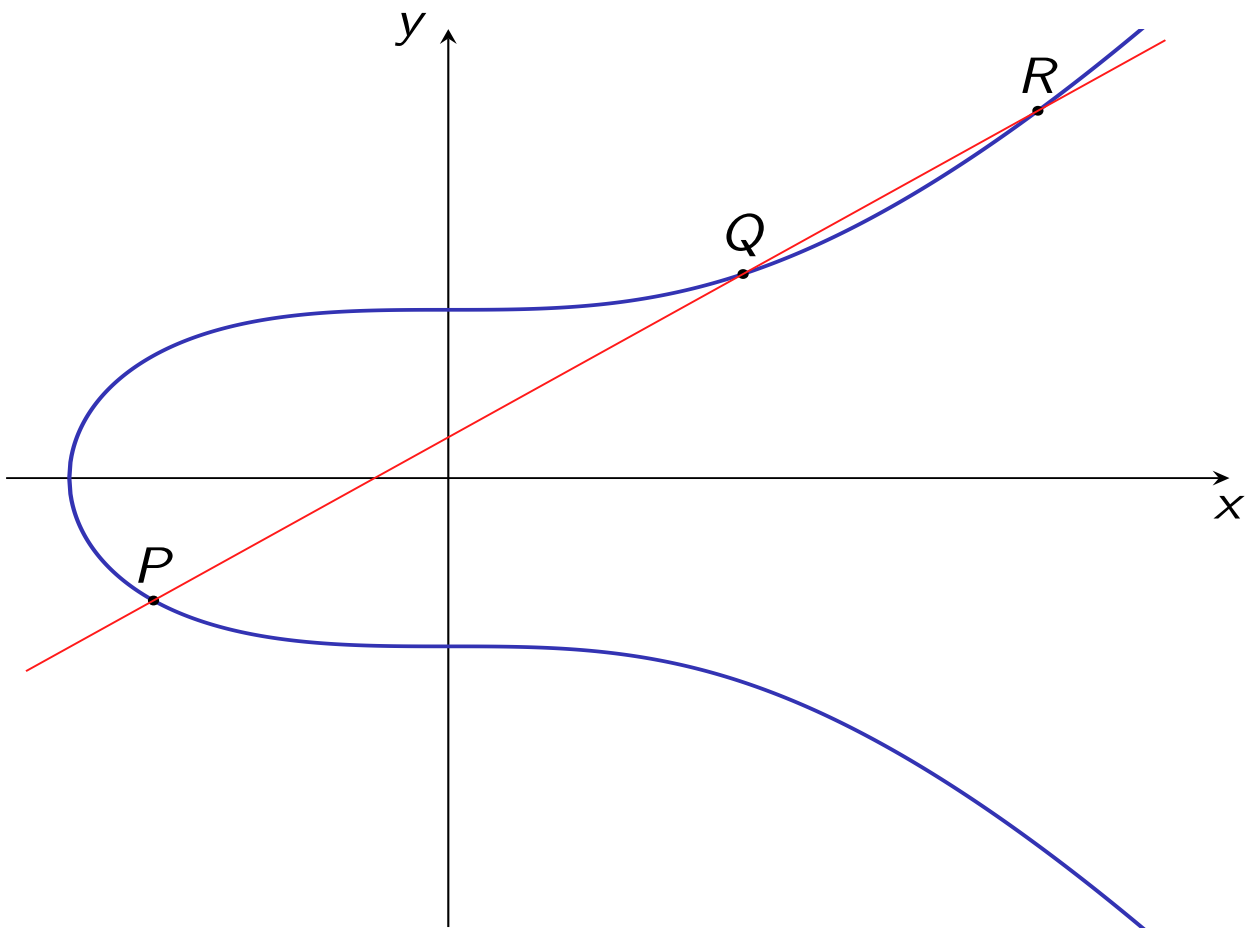
$P + Q$ (cas général)



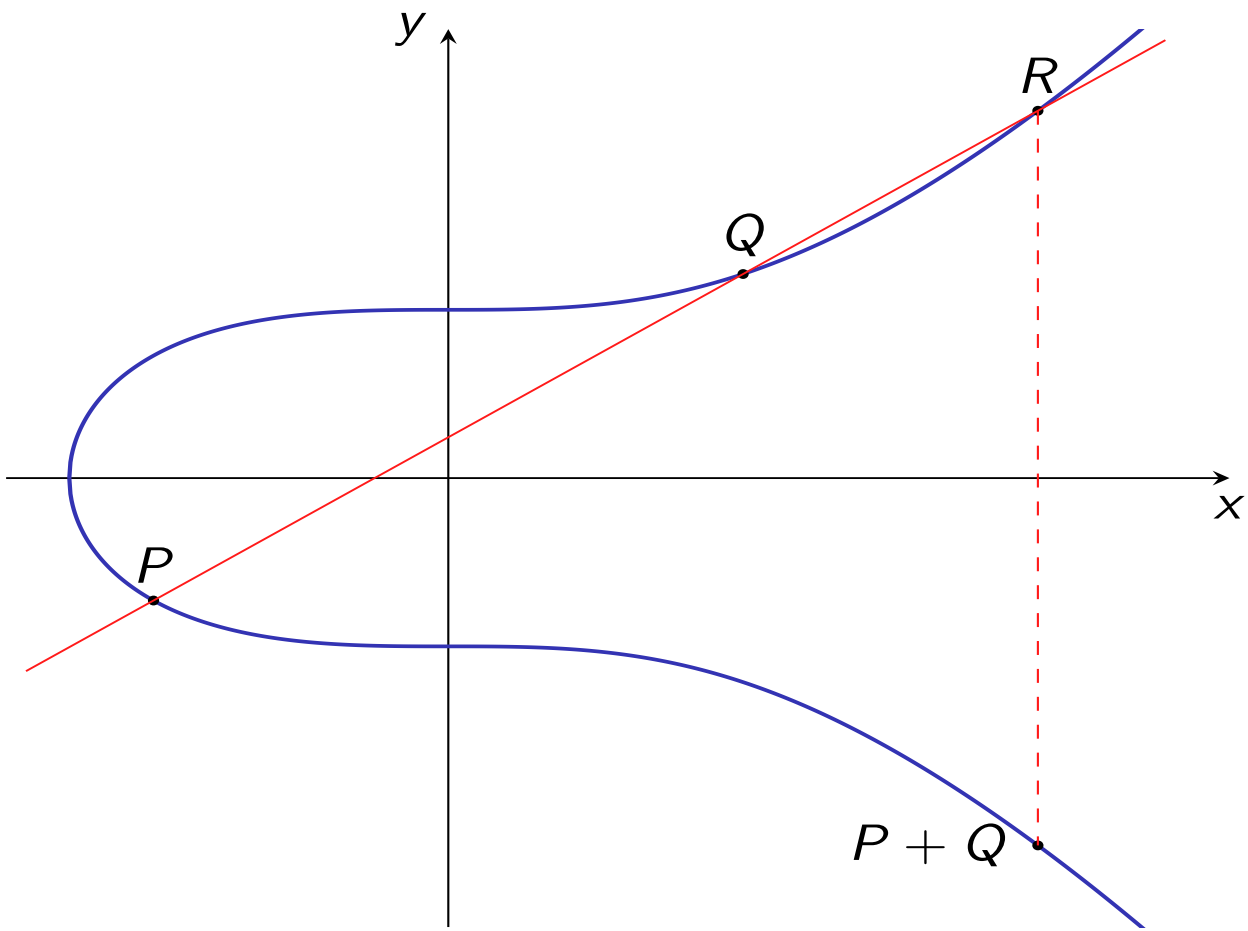
$P + Q$ (cas général)



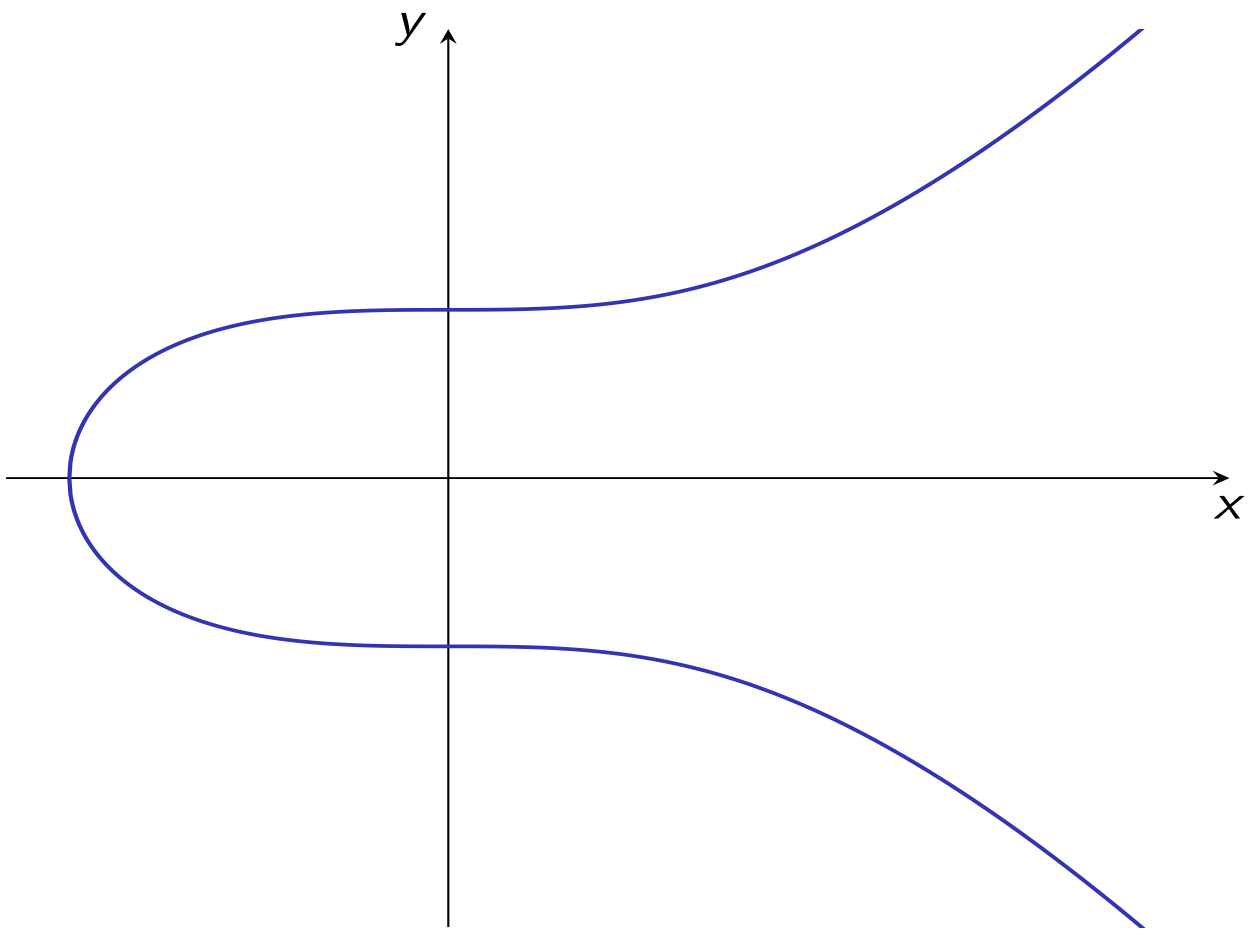
$P + Q$ (cas général)



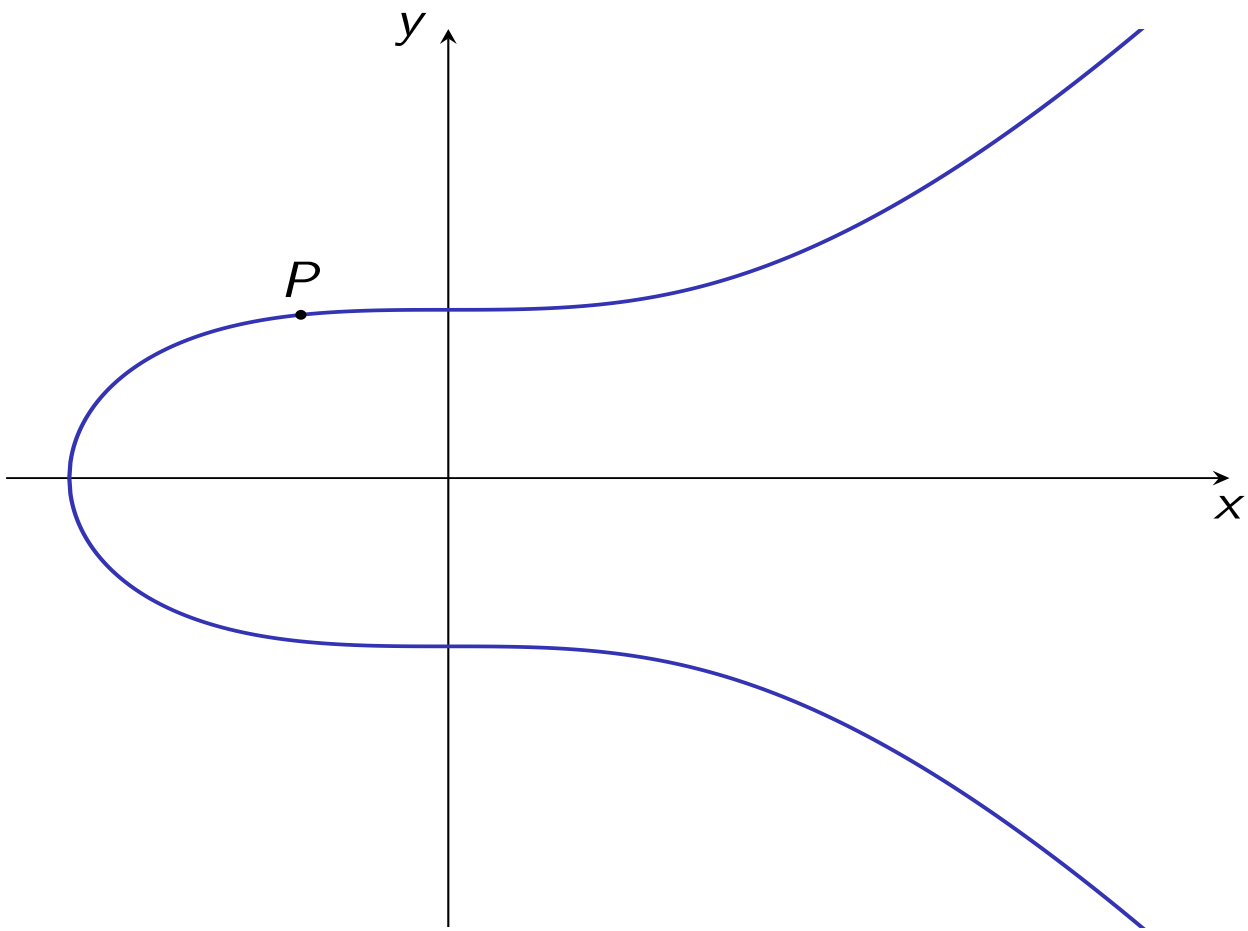
$P + Q$ (cas général)



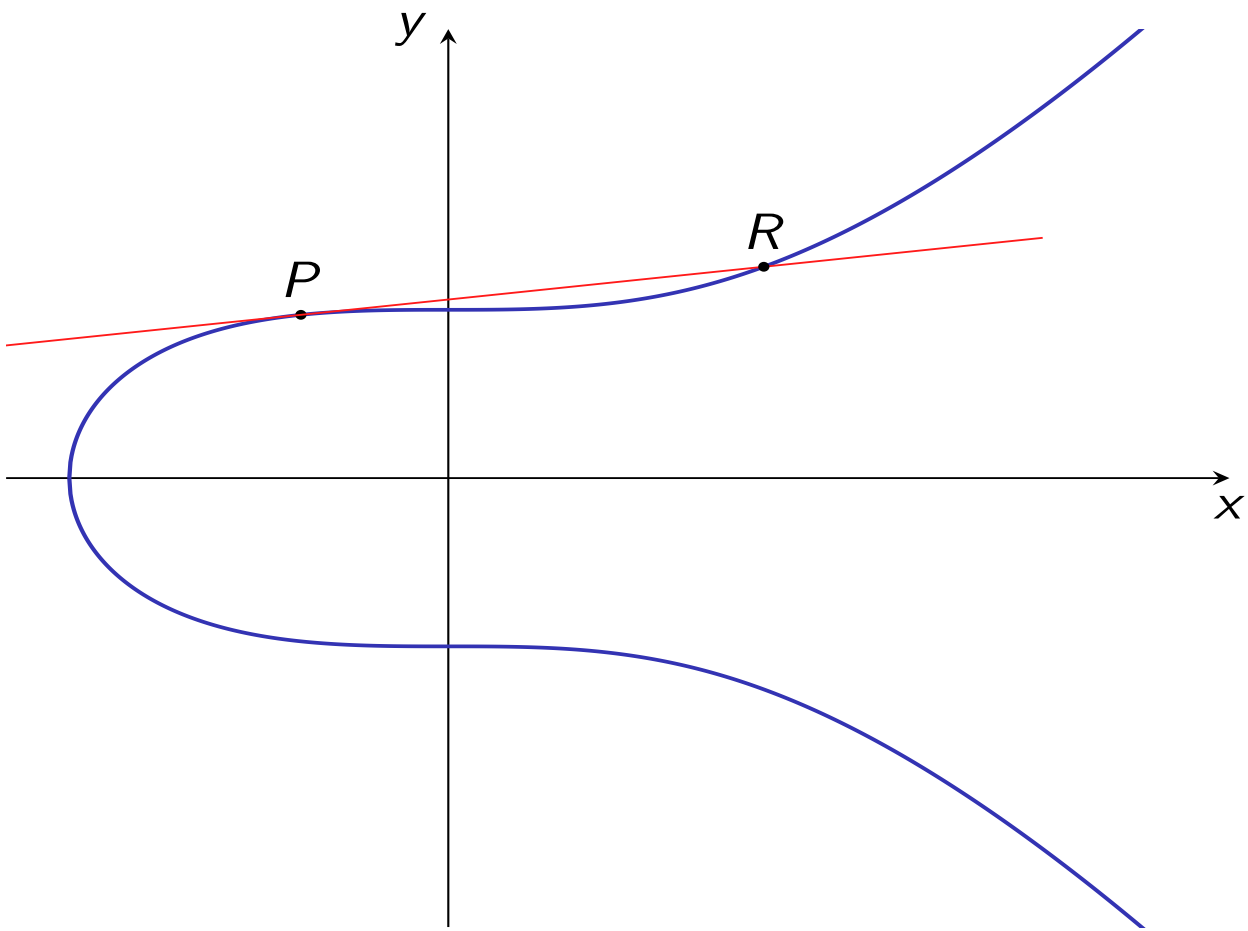
$2P = P + P$ (doublement d'un point)



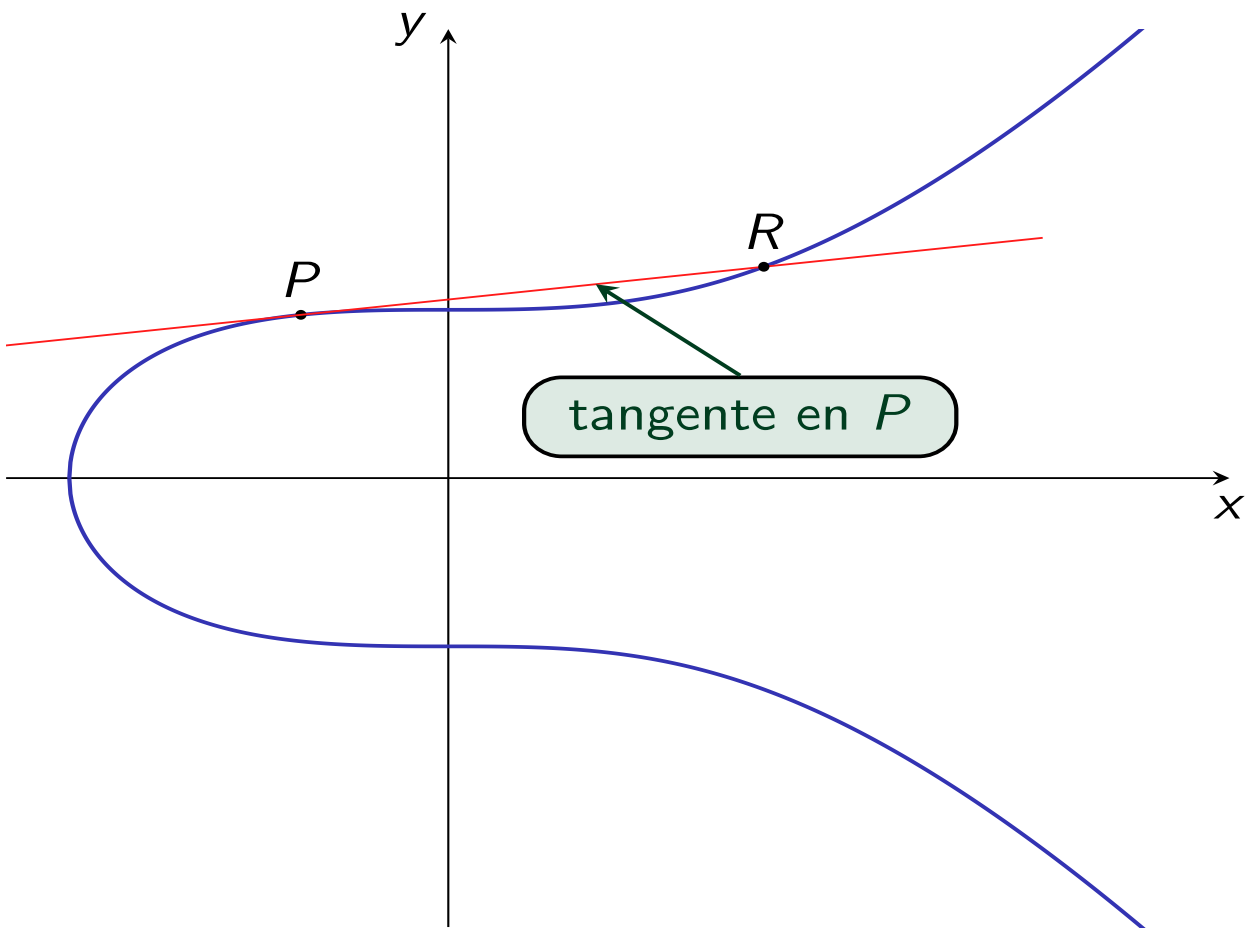
$2P = P + P$ (doublement d'un point)



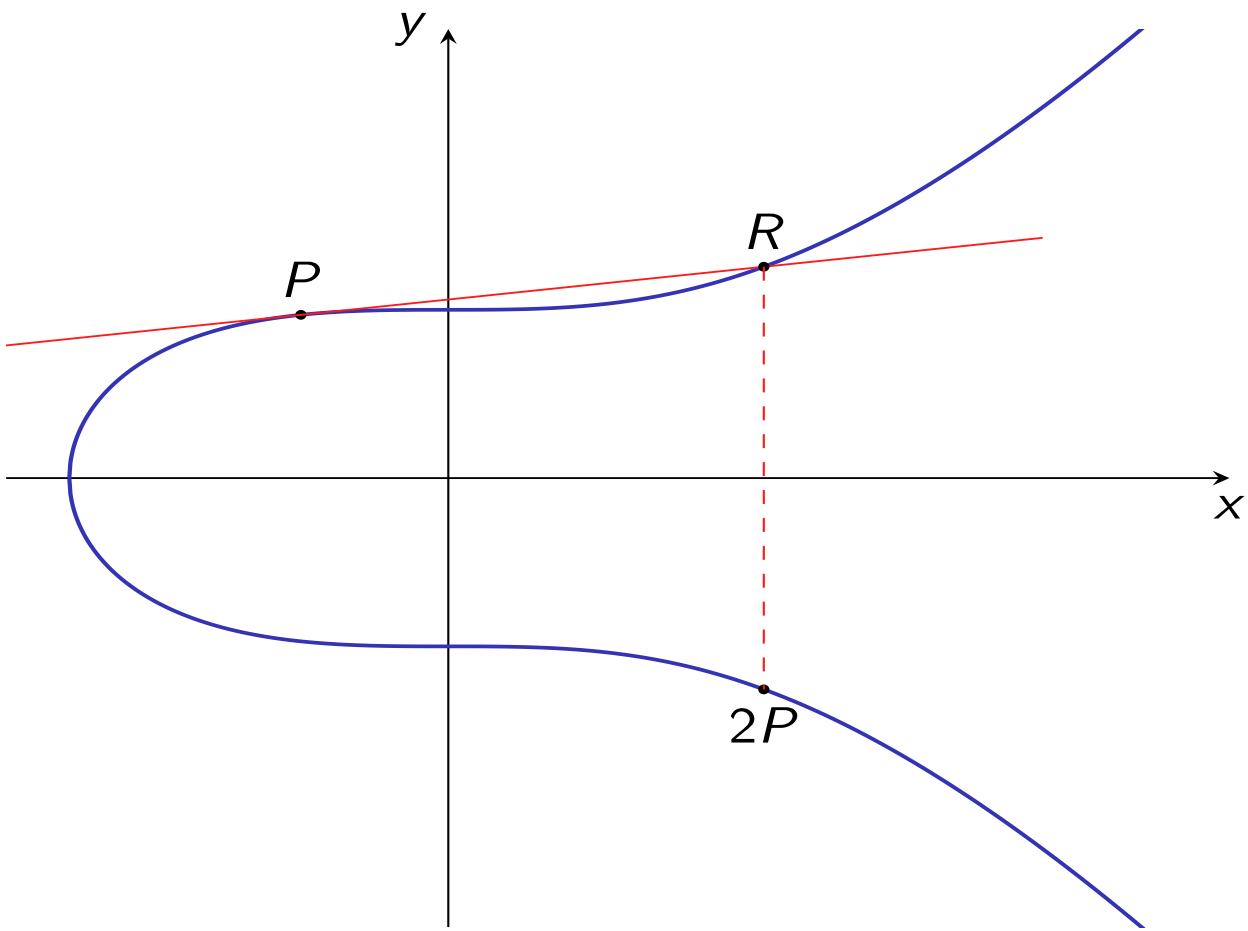
$2P = P + P$ (doublement d'un point)



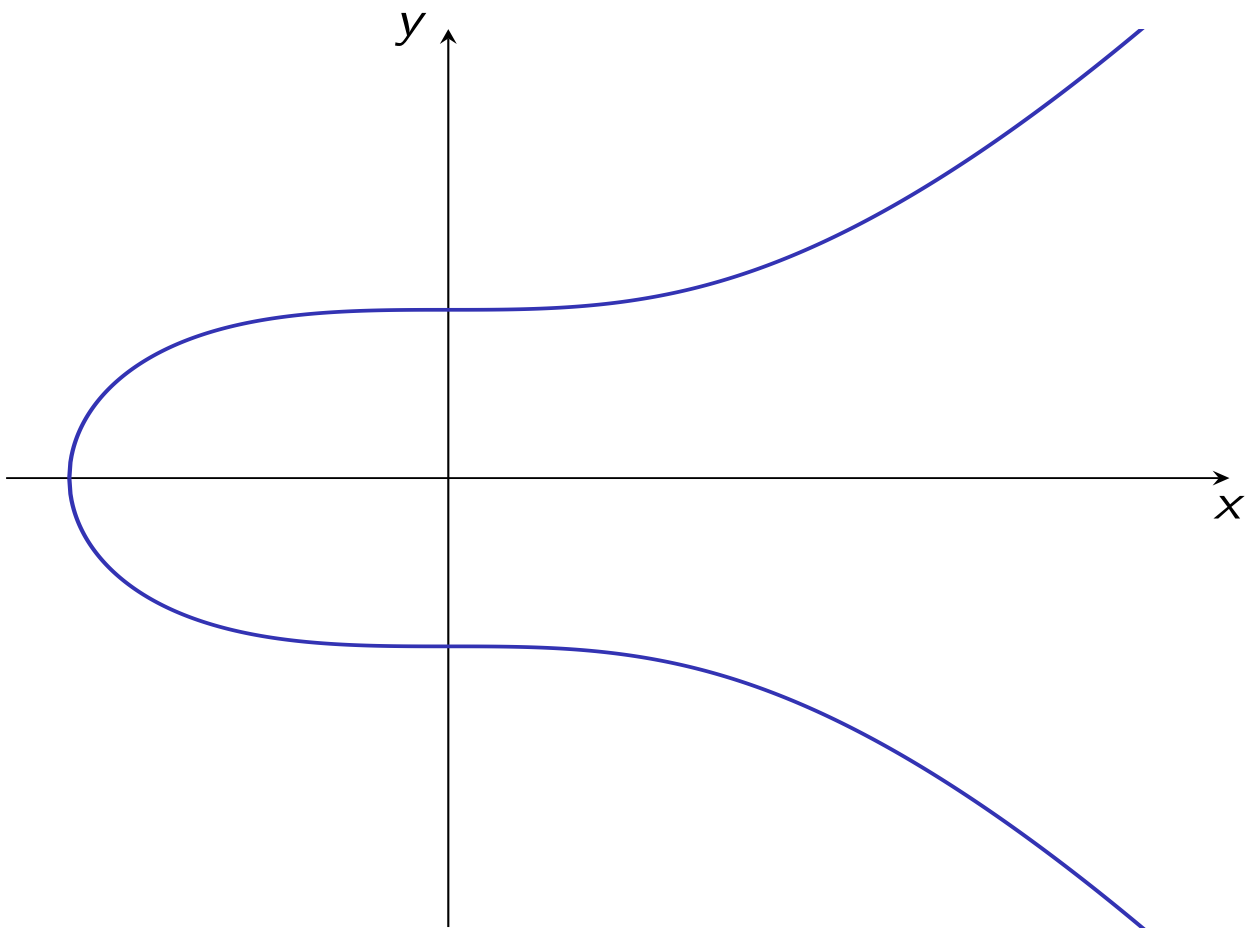
$2P = P + P$ (doublement d'un point)



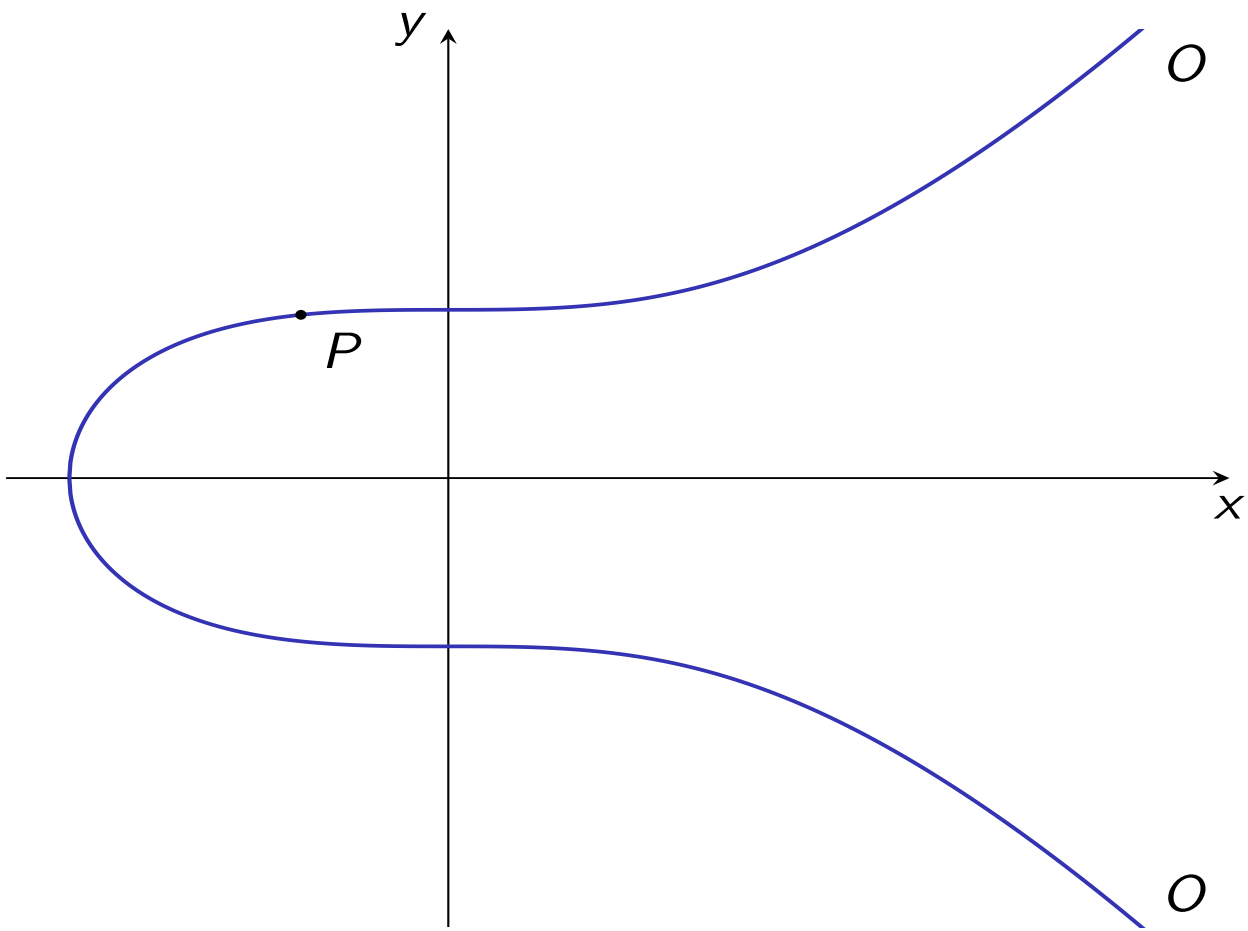
$2P = P + P$ (doublement d'un point)



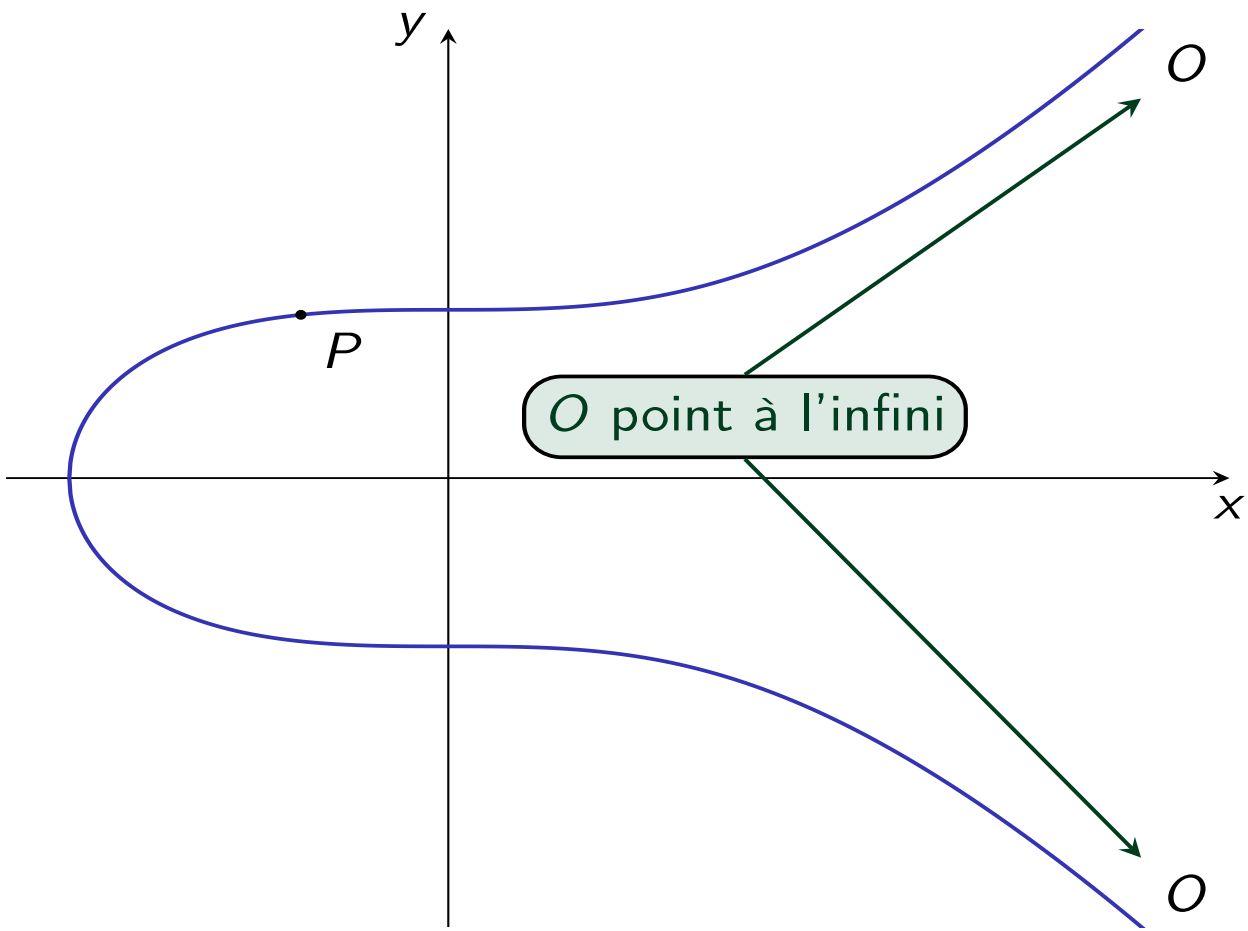
$P + O = O + P = P$ *(élément neutre)*



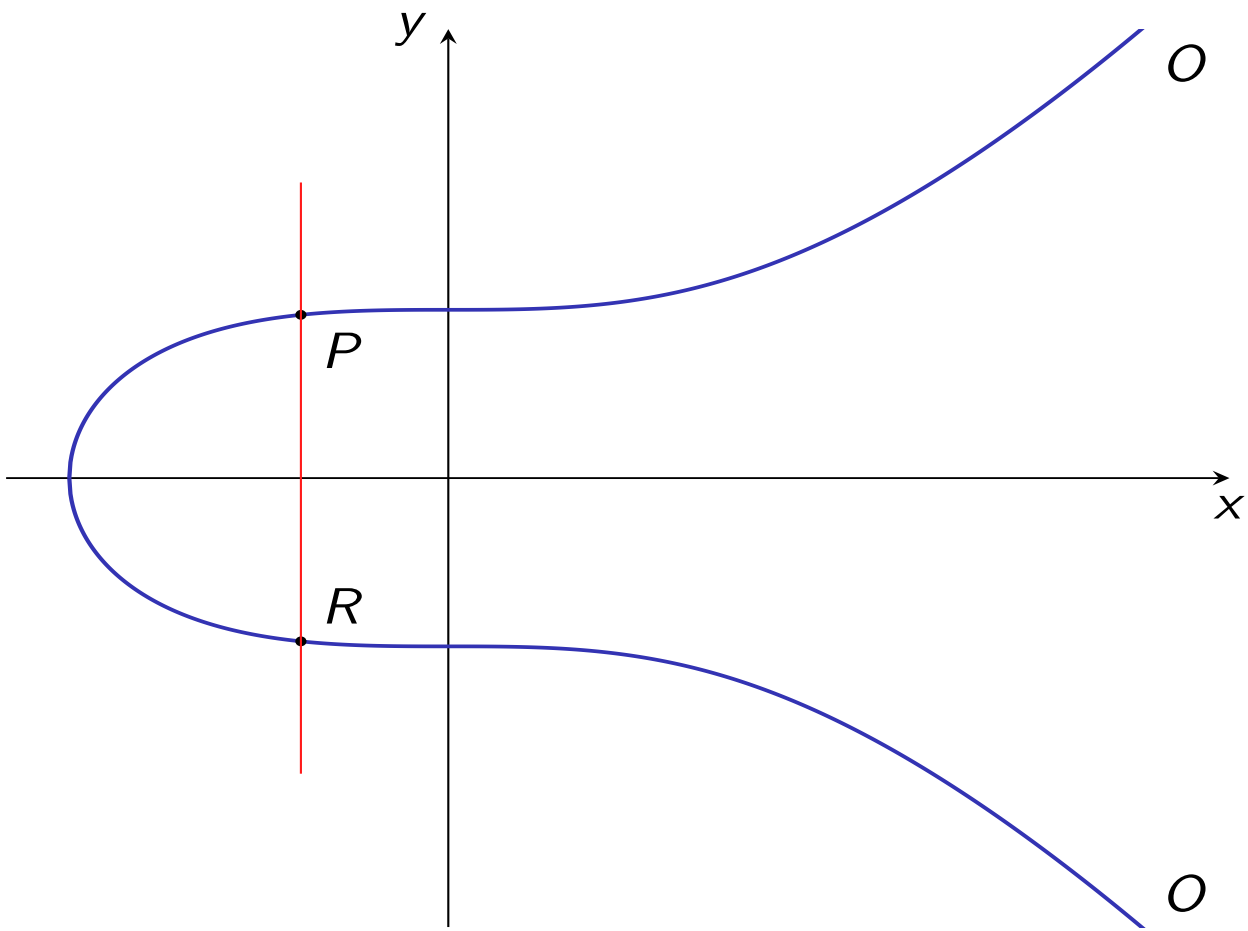
$P + O = O + P = P$ *(élément neutre)*



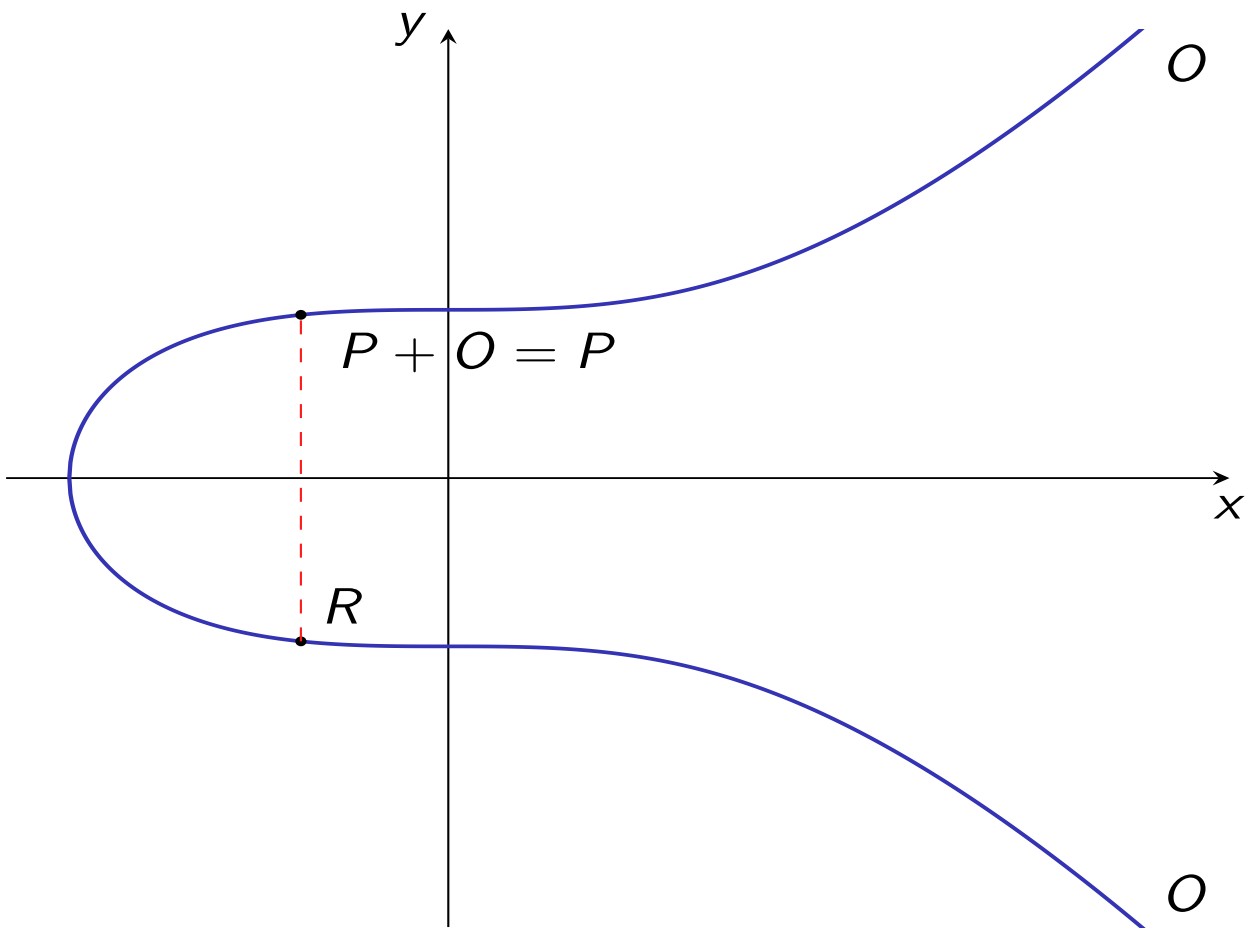
$P + O = O + P = P$ *(élément neutre)*



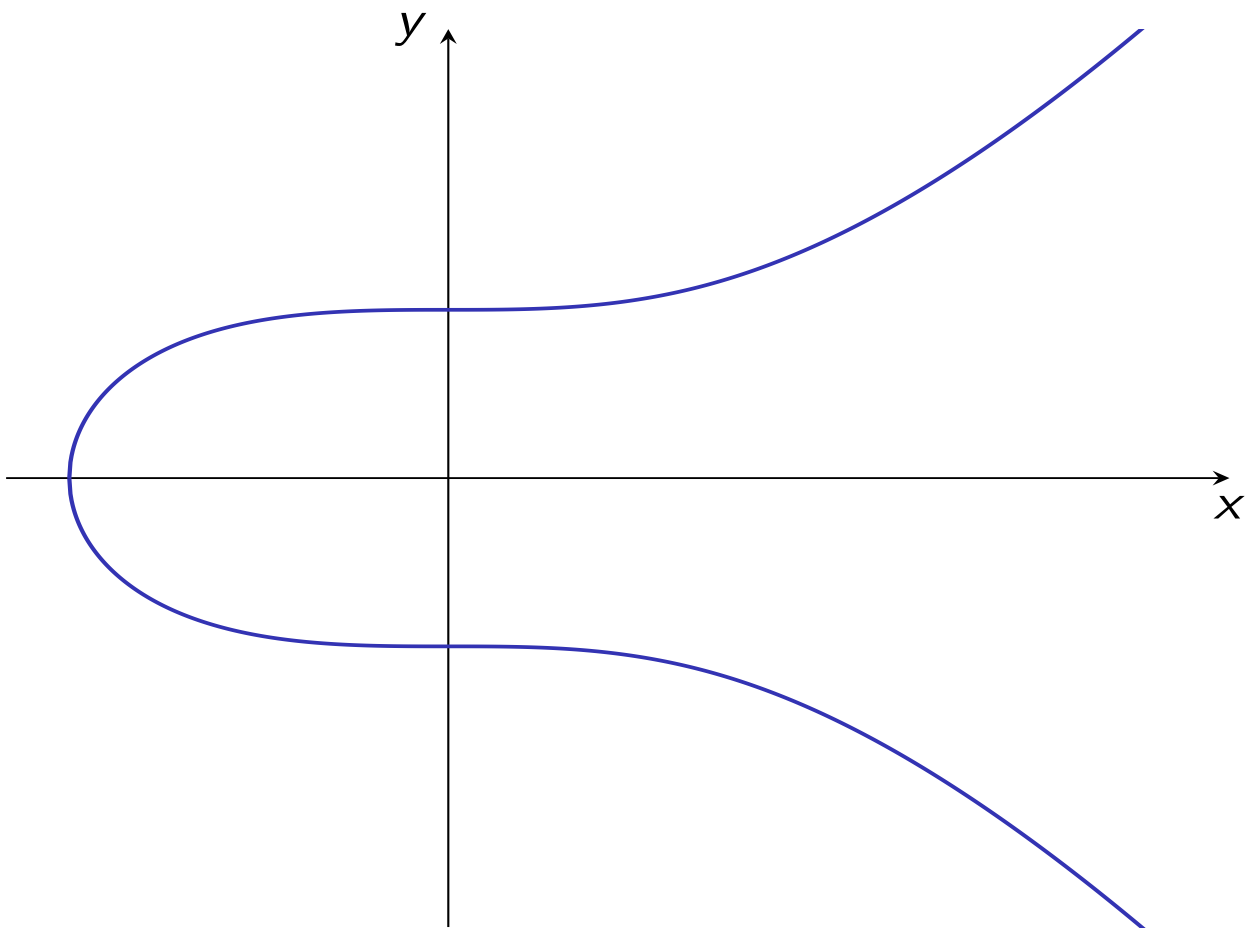
$P + O = O + P = P$ *(élément neutre)*



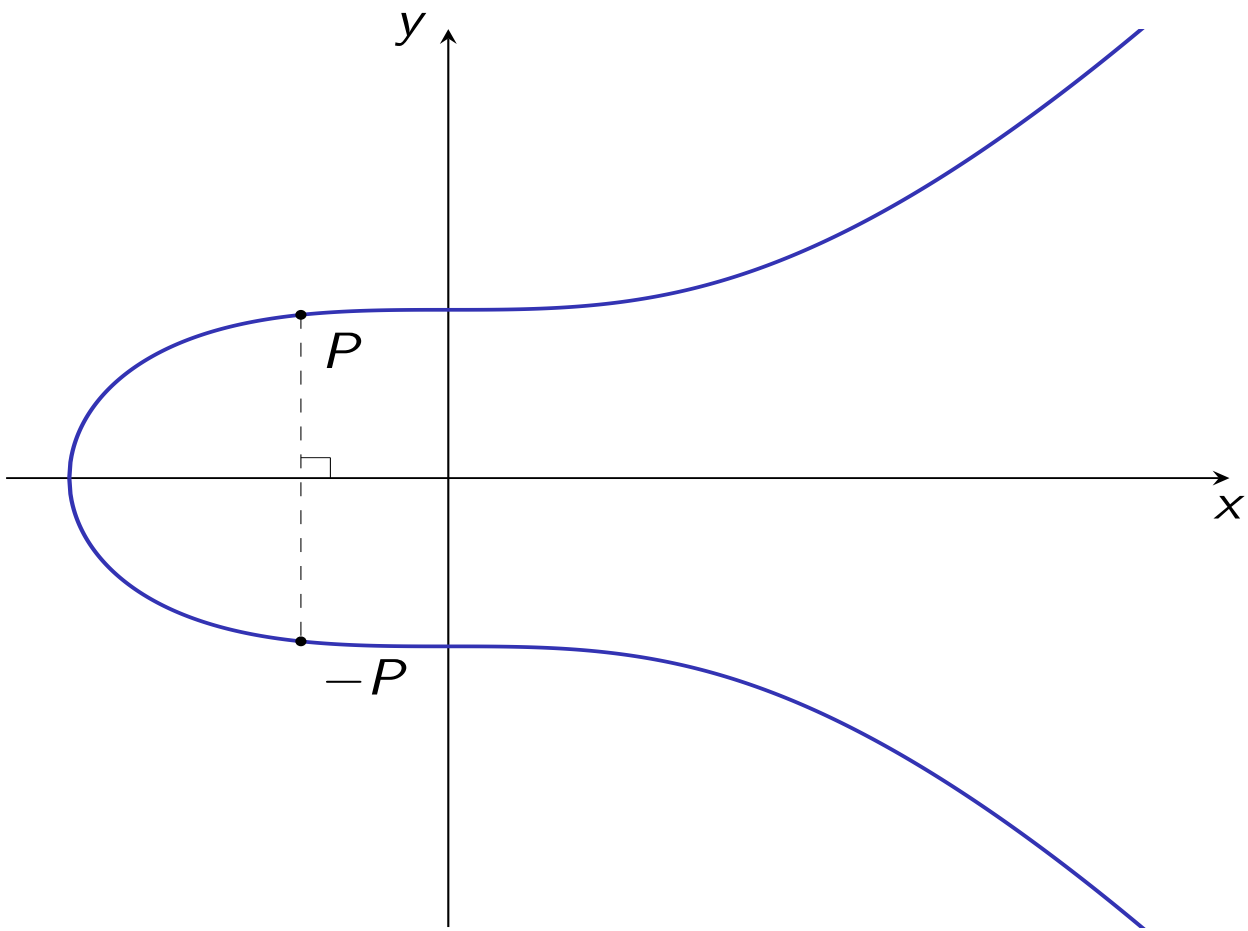
$P + O = O + P = P$ *(élément neutre)*



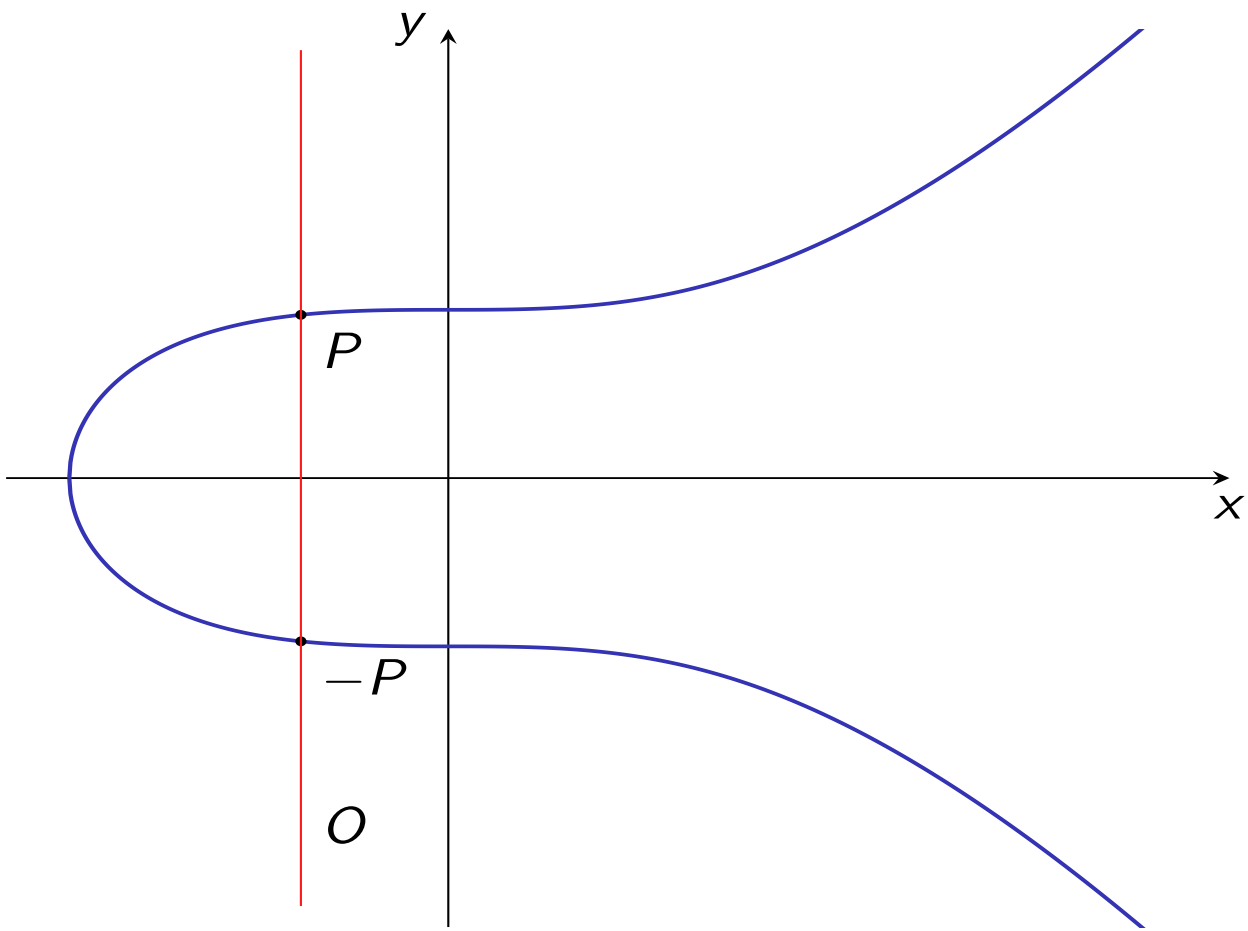
$P + (-P) = (-P) + P = O$ *(élément symétrique)*



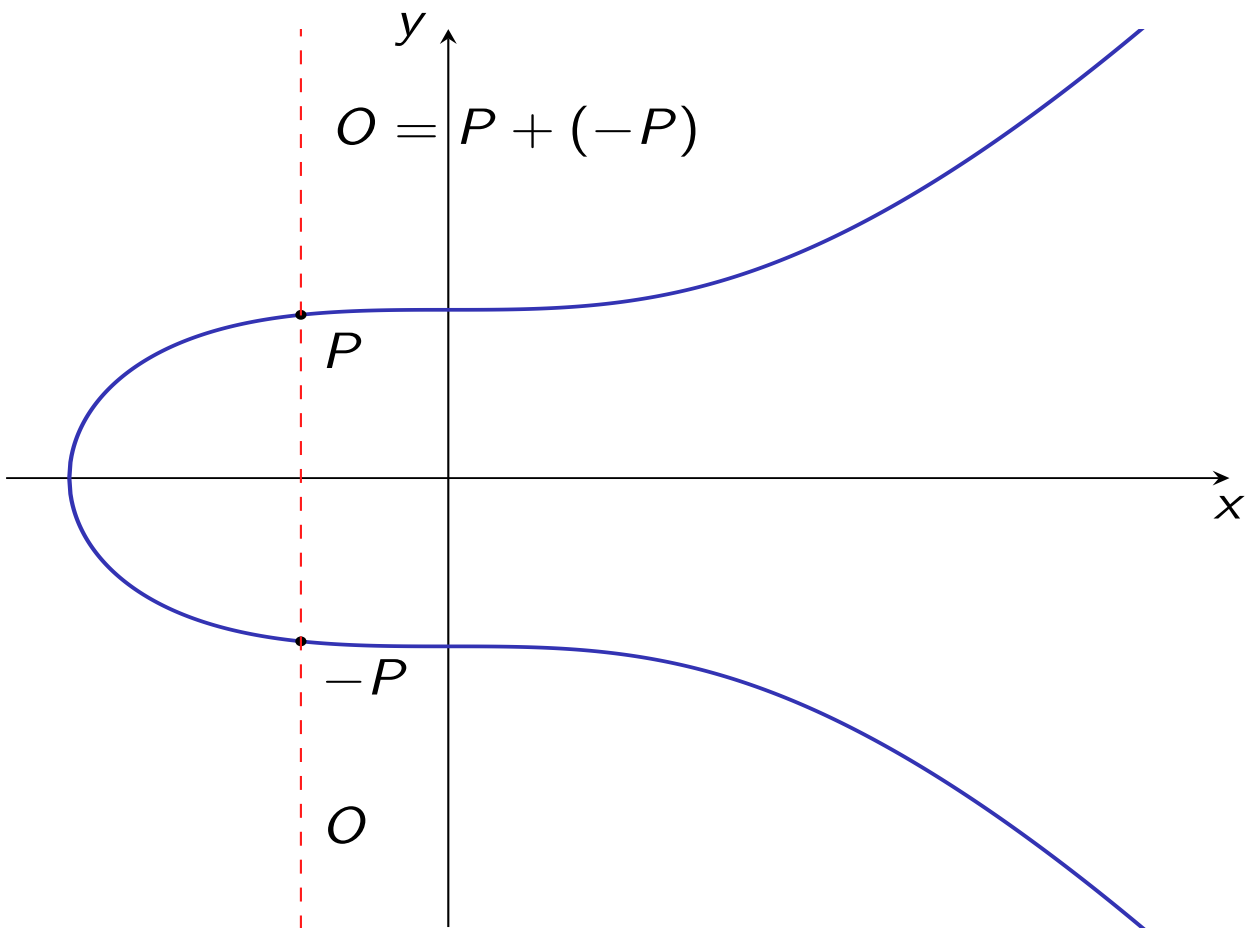
$P + (-P) = (-P) + P = O$ *(élément symétrique)*



$P + (-P) = (-P) + P = O$ *(élément symétrique)*



$P + (-P) = (-P) + P = O$ *(élément symétrique)*



Loi de groupe

- La loi de composition que nous venons de définir sur la courbe elliptique E est bien une loi de groupe (commutatif).
- Tous les axiomes sont trivialement satisfaits excepté l'associativité. Il existe cependant une démonstration élégante de l'associativité à l'aide d'un théorème de *Bezout* sur l'intersection de courbes algébriques.
- L'élément neutre est le "point à l'infini" O.
- L'élément symétrique $-P$ d'un point P est le symétrique de ce point par rapport à l'axe des abscisses $\{y = 0\}$.
- On notera qu'un point de $E \cap \{y = 0\}$ est son propre inverse.
- L'hypothèse de lissité de E est fondamentale. Sinon, par exemple, on ne pourrait pas définir $P + P$ en un point P sans tangente.
- On va maintenant oublier l'origine géométrique de cette loi de groupe et traduire cette dernière en termes algébriques. C'est une étape **capitale** pour la suite.

Loi de groupe

- La loi de composition que nous venons de définir sur la courbe elliptique E est bien une loi de groupe (commutatif).
- Tous les axiomes sont trivialement satisfaits excepté l'associativité. Il existe cependant une démonstration élégante de l'associativité à l'aide d'un théorème de *Bezout* sur l'intersection de courbes algébriques.
- L'élément neutre est le "point à l'infini" O.
- L'élément symétrique $-P$ d'un point P est le symétrique de ce point par rapport à l'axe des abscisses $\{y = 0\}$.
- On notera qu'un point de $E \cap \{y = 0\}$ est son propre inverse.
- L'hypothèse de lissité de E est fondamentale. Sinon, par exemple, on ne pourrait pas définir $P + P$ en un point P sans tangente.
- On va maintenant oublier l'origine géométrique de cette loi de groupe et traduire cette dernière en termes algébriques. C'est une étape **capitale** pour la suite.

Loi de groupe

- La loi de composition que nous venons de définir sur la courbe elliptique E est bien une loi de groupe (commutatif).
- Tous les axiomes sont trivialement satisfaits excepté l'associativité. Il existe cependant une démonstration élégante de l'associativité à l'aide d'un théorème de *Bezout* sur l'intersection de courbes algébriques.
- L'élément neutre est le "point à l'infini" O.
- L'élément symétrique $-P$ d'un point P est le symétrique de ce point par rapport à l'axe des abscisses $\{y = 0\}$.
- On notera qu'un point de $E \cap \{y = 0\}$ est son propre inverse.
- L'hypothèse de lissité de E est fondamentale. Sinon, par exemple, on ne pourrait pas définir $P + P$ en un point P sans tangente.
- On va maintenant oublier l'origine géométrique de cette loi de groupe et traduire cette dernière en termes algébriques. C'est une étape **capitale** pour la suite.

Loi de groupe

- La loi de composition que nous venons de définir sur la courbe elliptique E est bien une loi de groupe (commutatif).
- Tous les axiomes sont trivialement satisfaits excepté l'associativité. Il existe cependant une démonstration élégante de l'associativité à l'aide d'un théorème de *Bezout* sur l'intersection de courbes algébriques.
- L'élément neutre est le "point à l'infini" O.
- L'élément symétrique $-P$ d'un point P est le symétrique de ce point par rapport à l'axe des abscisses $\{y = 0\}$.
- On notera qu'un point de $E \cap \{y = 0\}$ est son propre inverse.
- L'hypothèse de lissité de E est fondamentale. Sinon, par exemple, on ne pourrait pas définir $P + P$ en un point P sans tangente.
- On va maintenant oublier l'origine géométrique de cette loi de groupe et traduire cette dernière en termes algébriques. C'est une étape **capitale** pour la suite.

Loi de groupe

- La loi de composition que nous venons de définir sur la courbe elliptique E est bien une loi de groupe (commutatif).
- Tous les axiomes sont trivialement satisfaits excepté l'associativité. Il existe cependant une démonstration élégante de l'associativité à l'aide d'un théorème de *Bezout* sur l'intersection de courbes algébriques.
- L'élément neutre est le "point à l'infini" O.
- L'élément symétrique $-P$ d'un point P est le symétrique de ce point par rapport à l'axe des abscisses $\{y = 0\}$.
- On notera qu'un point de $E \cap \{y = 0\}$ est son propre inverse.
- L'hypothèse de lissité de E est fondamentale. Sinon, par exemple, on ne pourrait pas définir $P + P$ en un point P sans tangente.
- On va maintenant oublier l'origine géométrique de cette loi de groupe et traduire cette dernière en termes algébriques. C'est une étape **capitale** pour la suite.

Loi de groupe

- La loi de composition que nous venons de définir sur la courbe elliptique E est bien une loi de groupe (commutatif).
- Tous les axiomes sont trivialement satisfaits excepté l'associativité. Il existe cependant une démonstration élégante de l'associativité à l'aide d'un théorème de *Bezout* sur l'intersection de courbes algébriques.
- L'élément neutre est le "point à l'infini" O.
- L'élément symétrique $-P$ d'un point P est le symétrique de ce point par rapport à l'axe des abscisses $\{y = 0\}$.
- On notera qu'un point de $E \cap \{y = 0\}$ est son propre inverse.
- L'hypothèse de lissité de E est fondamentale. Sinon, par exemple, on ne pourrait pas définir $P + P$ en un point P sans tangente.
- On va maintenant oublier l'origine géométrique de cette loi de groupe et traduire cette dernière en termes algébriques. C'est une étape **capitale** pour la suite.

Loi de groupe

- La loi de composition que nous venons de définir sur la courbe elliptique E est bien une loi de groupe (commutatif).
- Tous les axiomes sont trivialement satisfaits excepté l'associativité. Il existe cependant une démonstration élégante de l'associativité à l'aide d'un théorème de *Bezout* sur l'intersection de courbes algébriques.
- L'élément neutre est le "point à l'infini" O.
- L'élément symétrique $-P$ d'un point P est le symétrique de ce point par rapport à l'axe des abscisses $\{y = 0\}$.
- On notera qu'un point de $E \cap \{y = 0\}$ est son propre inverse.
- L'hypothèse de lissité de E est fondamentale. Sinon, par exemple, on ne pourrait pas définir $P + P$ en un point P sans tangente.
- On va maintenant oublier l'origine géométrique de cette loi de groupe et traduire cette dernière en termes algébriques. C'est une étape **capitale** pour la suite.

Formules algébriques

Soit E la courbe elliptique d'équation $y^2 = x^3 + ax + b$ (avec $O \notin E$ "point à l'infini") et soient $P = (x_p, y_p)$ et $Q = (x_q, y_q)$ deux points de E. Alors $P + Q$ est défini comme suit :

(a) Si $P \neq Q$ et $x_p = x_q$, alors $P + Q = O$ ($Q = -P$).

(b) Si $P = Q$ et $y_p = 0$, alors $P + Q = 2P = O$.

(c) Si $P \neq Q$ et $x_p \neq x_q$, alors
$P + Q = (\lambda^2 - x_p - x_q, -\lambda^3 + \lambda(x_p + x_q) - \mu)$ avec $\lambda = \frac{y_p - y_q}{x_p - x_q}$ et $\mu = \frac{y_q x_p - y_p x_q}{x_p - x_q}$.

(d) Si $P = Q$ et $y_p \neq 0$, alors
$P + Q = 2P = (\lambda^2 - 2x_p, -\lambda^3 + 2\lambda x_p - \mu)$ avec $\lambda = \frac{3x_p^2 + a}{2y_p}$ et $\mu = \frac{-x_p^3 + ax_p + 2b}{2y_p}$.

Remarque importante : si $a, b \in k$ où k est un corps commutatif et, si P, Q ont leurs coordonnées dans k, il en est de même pour $P + Q$.

Formules algébriques

Soit E la courbe elliptique d'équation $y^2 = x^3 + ax + b$ (avec $O \notin E$ "point à l'infini") et soient $P = (x_p, y_p)$ et $Q = (x_q, y_q)$ deux points de E. Alors $P + Q$ est défini comme suit :

(a) Si $P \neq Q$ et $x_p = x_q$, alors $P + Q = O$ ($Q = -P$).

(b) Si $P = Q$ et $y_p = 0$, alors $P + Q = 2P = O$.

(c) Si $P \neq Q$ et $x_p \neq x_q$, alors
$P + Q = (\lambda^2 - x_p - x_q, -\lambda^3 + \lambda(x_p + x_q) - \mu)$ avec $\lambda = \frac{y_p - y_q}{x_p - x_q}$ et $\mu = \frac{y_q x_p - y_p x_q}{x_p - x_q}$.

(d) Si $P = Q$ et $y_p \neq 0$, alors
$P + Q = 2P = (\lambda^2 - 2x_p, -\lambda^3 + 2\lambda x_p - \mu)$ avec $\lambda = \frac{3x_p^2 + a}{2y_p}$ et $\mu = \frac{-x_p^3 + a x_p + 2b}{2y_p}$.

Remarque importante : si $a, b \in k$ où k est un corps commutatif et, si P, Q ont leurs coordonnées dans k, il en est de même pour $P + Q$.

Formules algébriques

Soit E la courbe elliptique d'équation $y^2 = x^3 + ax + b$ (avec $O \notin E$ "point à l'infini") et soient $P = (x_p, y_p)$ et $Q = (x_q, y_q)$ deux points de E. Alors $P + Q$ est défini comme suit :

(a) Si $P \neq Q$ et $x_p = x_q$, alors $P + Q = O$ ($Q = -P$).

(b) Si $P = Q$ et $y_p = 0$, alors $P + Q = 2P = O$.

(c) Si $P \neq Q$ et $x_p \neq x_q$, alors
$P + Q = (\lambda^2 - x_p - x_q, -\lambda^3 + \lambda(x_p + x_q) - \mu)$ avec $\lambda = \frac{y_p - y_q}{x_p - x_q}$ et $\mu = \frac{y_q x_p - y_p x_q}{x_p - x_q}$.

(d) Si $P = Q$ et $y_p \neq 0$, alors
$P + Q = 2P = (\lambda^2 - 2x_p, -\lambda^3 + 2\lambda x_p - \mu)$ avec $\lambda = \frac{3x_p^2 + a}{2y_p}$ et $\mu = \frac{-x_p^3 + ax_p + 2b}{2y_p}$.

Remarque importante : si $a, b \in k$ où k est un corps commutatif et, si P, Q ont leurs coordonnées dans k, il en est de même pour $P + Q$.

Formules algébriques

Soit E la courbe elliptique d'équation $y^2 = x^3 + ax + b$ (avec $O \notin E$ "point à l'infini") et soient $P = (x_p, y_p)$ et $Q = (x_q, y_q)$ deux points de E. Alors $P + Q$ est défini comme suit :

(a) Si $P \neq Q$ et $x_p = x_q$, alors $P + Q = O$ ($Q = -P$).

(b) Si $P = Q$ et $y_p = 0$, alors $P + Q = 2P = O$.

(c) Si $P \neq Q$ et $x_p \neq x_q$, alors
$P + Q = (\lambda^2 - x_p - x_q, -\lambda^3 + \lambda(x_p + x_q) - \mu)$ avec $\lambda = \frac{y_p - y_q}{x_p - x_q}$ et $\mu = \frac{y_q x_p - y_p x_q}{x_p - x_q}$.

(d) Si $P = Q$ et $y_p \neq 0$, alors
$P + Q = 2P = (\lambda^2 - 2x_p, -\lambda^3 + 2\lambda x_p - \mu)$ avec $\lambda = \frac{3x_p^2 + a}{2y_p}$ et $\mu = \frac{-x_p^3 + ax_p + 2b}{2y_p}$.

Remarque importante : si $a, b \in k$ où k est un corps commutatif et, si P, Q ont leurs coordonnées dans k, il en est de même pour $P + Q$.

Formules algébriques

Soit E la courbe elliptique d'équation $y^2 = x^3 + ax + b$ (avec $O \notin E$ "point à l'infini") et soient $P = (x_p, y_p)$ et $Q = (x_q, y_q)$ deux points de E. Alors $P + Q$ est défini comme suit :

(a) Si $P \neq Q$ et $x_p = x_q$, alors $P + Q = O$ ($Q = -P$).

(b) Si $P = Q$ et $y_p = 0$, alors $P + Q = 2P = O$.

(c) Si $P \neq Q$ et $x_p \neq x_q$, alors
$P + Q = (\lambda^2 - x_p - x_q, -\lambda^3 + \lambda(x_p + x_q) - \mu)$ avec $\lambda = \frac{y_p - y_q}{x_p - x_q}$ et $\mu = \frac{y_q x_p - y_p x_q}{x_p - x_q}$.

(d) Si $P = Q$ et $y_p \neq 0$, alors
$P + Q = 2P = (\lambda^2 - 2x_p, -\lambda^3 + 2\lambda x_p - \mu)$ avec $\lambda = \frac{3x_p^2 + a}{2y_p}$ et $\mu = \frac{-x_p^3 + a x_p + 2b}{2y_p}$.

Remarque importante : si $a, b \in k$ où k est un corps commutatif et, si P, Q ont leurs coordonnées dans k, il en est de même pour $P + Q$.

Formules algébriques

Soit E la courbe elliptique d'équation $y^2 = x^3 + ax + b$ (avec $O \notin E$ "point à l'infini") et soient $P = (x_p, y_p)$ et $Q = (x_q, y_q)$ deux points de E. Alors $P + Q$ est défini comme suit :

(a) Si $P \neq Q$ et $x_p = x_q$, alors $P + Q = O$ $(Q = -P)$.

(b) Si $P = Q$ et $y_p = 0$, alors $P + Q = 2P = O$.

(c) Si $P \neq Q$ et $x_p \neq x_q$, alors
$P + Q = (\lambda^2 - x_p - x_q, -\lambda^3 + \lambda(x_p + x_q) - \mu)$ avec $\lambda = \frac{y_p - y_q}{x_p - x_q}$ et $\mu = \frac{y_q x_p - y_p x_q}{x_p - x_q}$.

(d) Si $P = Q$ et $y_p \neq 0$, alors
$P + Q = 2P = (\lambda^2 - 2x_p, -\lambda^3 + 2\lambda x_p - \mu)$ avec $\lambda = \frac{3x_p^2 + a}{2y_p}$ et $\mu = \frac{-x_p^3 + ax_p + 2b}{2y_p}$.

Remarque importante : si $a, b \in k$ où k est un corps commutatif et, si P, Q ont leurs coordonnées dans k, il en est de même pour $P + Q$.

Problème du logarithme discret pour les courbes elliptiques

- Les courbes elliptiques que nous avons vues en exemple jusqu'à présent étaient dans le plan réel $\mathbb{R}^2$. Or le problème du logarithme discret pour (les groupes construits) sur de telles courbes (PLDCE en abrégé) est trivial à résoudre en utilisant le logarithme népérien, comme dans le cas du groupe $(\mathbb{R}^{+*}, \cdot)$ vu précédemment.
- Si on remplace l'ensemble $\mathbb{R}$ par un ensemble fini, nous verrons que dans ce cas le PLDCE est en général **impossible à résoudre en un temps raisonnable** pour peu que l'ensemble fini ait suffisamment d'éléments. *Koblitz* et *Miller* ont été les premiers à voir le potentiel cryptographique de cette impossibilité.
- Si on veut définir une loi de groupe comme précédemment, il va falloir également que cet ensemble fini soit muni d'une addition et d'une multiplication ayant des propriétés analogues à celles de $\mathbb{R}$.
- Un tel ensemble est appelé *corps fini*.

Problème du logarithme discret pour les courbes elliptiques

- Les courbes elliptiques que nous avons vues en exemple jusqu'à présent étaient dans le plan réel $\mathbb{R}^2$. Or le problème du logarithme discret pour (les groupes construits) sur de telles courbes (PLDCE en abrégé) est trivial à résoudre en utilisant le logarithme népérien, comme dans le cas du groupe $(\mathbb{R}^{+*}, \cdot)$ vu précédemment.
- Si on remplace l'ensemble $\mathbb{R}$ par un ensemble fini, nous verrons que dans ce cas le PLDCE est en général **impossible à résoudre en un temps raisonnable** pour peu que l'ensemble fini ait suffisamment d'éléments. *Koblitz* et *Miller* ont été les premiers à voir le potentiel cryptographique de cette impossibilité.
- Si on veut définir une loi de groupe comme précédemment, il va falloir également que cet ensemble fini soit muni d'une addition et d'une multiplication ayant des propriétés analogues à celles de $\mathbb{R}$.
- Un tel ensemble est appelé *corps fini*.

Problème du logarithme discret pour les courbes elliptiques

- Les courbes elliptiques que nous avons vues en exemple jusqu'à présent étaient dans le plan réel $\mathbb{R}^2$. Or le problème du logarithme discret pour (les groupes construits) sur de telles courbes (PLDCE en abrégé) est trivial à résoudre en utilisant le logarithme népérien, comme dans le cas du groupe $(\mathbb{R}^{+*}, \cdot)$ vu précédemment.
- Si on remplace l'ensemble $\mathbb{R}$ par un ensemble fini, nous verrons que dans ce cas le PLDCE est en général **impossible à résoudre en un temps raisonnable** pour peu que l'ensemble fini ait suffisamment d'éléments. *Koblitz* et *Miller* ont été les premiers à voir le potentiel cryptographique de cette impossibilité.
- Si on veut définir une loi de groupe comme précédemment, il va falloir également que cet ensemble fini soit muni d'une addition et d'une multiplication ayant des propriétés analogues à celles de $\mathbb{R}$.
- Un tel ensemble est appelé *corps fini*.

Problème du logarithme discret pour les courbes elliptiques

- Les courbes elliptiques que nous avons vues en exemple jusqu'à présent étaient dans le plan réel $\mathbb{R}^2$. Or le problème du logarithme discret pour (les groupes construits) sur de telles courbes (PLDCE en abrégé) est trivial à résoudre en utilisant le logarithme népérien, comme dans le cas du groupe $(\mathbb{R}^{+*}, \cdot)$ vu précédemment.
- Si on remplace l'ensemble $\mathbb{R}$ par un ensemble fini, nous verrons que dans ce cas le PLDCE est en général **impossible à résoudre en un temps raisonnable** pour peu que l'ensemble fini ait suffisamment d'éléments. *Koblitz* et *Miller* ont été les premiers à voir le potentiel cryptographique de cette impossibilité.
- Si on veut définir une loi de groupe comme précédemment, il va falloir également que cet ensemble fini soit muni d'une addition et d'une multiplication ayant des propriétés analogues à celles de $\mathbb{R}$.
- Un tel ensemble est appelé *corps fini*.

Corps finis

Définition (corps commutatif)

Un corps commutatif est un ensemble k, ayant au moins deux éléments, muni de deux lois de composition notées $+$, $\cdot$ et tel que :

(i) $(k, +)$ *est groupe commutatif, l'élément neutre étant noté* 0,

(ii) $(k \setminus \{0\}, \cdot)$ *est groupe commutatif, l'élément neutre étant noté* 1,

(iii) $\forall x, y, z \in k, \quad x(y+z) = xy + xz$ *(distributivité).*

Si k est un ensemble fini, k est alors appelé corps fini.

Remarque

Dans le cas d'un corps fini, l'hypothèse de la commutativité de la multiplication est redondante car tous les corps finis sont commutatifs (thèorème de Wedderburn).

Corps finis

Définition (corps commutatif)

Un corps commutatif est un ensemble k, ayant au moins deux éléments, muni de deux lois de composition notées $+$, $\cdot$ et tel que :

(i) $(k, +)$ *est groupe commutatif, l'élément neutre étant noté* 0,

(ii) $(k \setminus \{0\}, \cdot)$ *est groupe commutatif, l'élément neutre étant noté* 1,

(iii) $\forall x, y, z \in k, \quad x(y+z) = xy + xz$ *(distributivité).*

Si k est un ensemble fini, k est alors appelé corps fini.

Remarque

Dans le cas d'un corps fini, l'hypothèse de la commutativité de la multiplication est redondante car tous les corps finis sont commutatifs (thèorème de Wedderburn).

Corps finis

Un premier exemple de corps fini est $(\mathbb{Z}/p\mathbb{Z}, +, \cdot)$, p premier. On le note généralement $\mathbb{F}_p$ (ou GF(p) en hommage à *Evariste Galois*). Le plus petit corps fini est $\mathbb{F}_2 = \{\overline{0}, \overline{1}\}$, nous l'avons déjà rencontré. Y-a-t-il d'autres corps finis ?

Théorème (E. H. Moore, 1893)

(i) *Si k est un corps fini, alors $\#(k) = p^n$, p premier et $n > 0$.*

(ii) *Réciproquement, pour tout $q = p^n$, p premier et $n > 0$, il existe un corps fini, unique à isomorphisme près, noté $\mathbb{F}_q$ tel que $\#(\mathbb{F}_q) = q$.*

Si P est un polynôme irréductible de degré n sur $\mathbb{F}_p[X]$ (il en existe toujours un), $\mathbb{F}_q \cong \mathbb{F}_p[X]/P$. Autrement dit, $\mathbb{F}_q$ peut être vu comme l'ensemble des polynômes sur $\mathbb{F}_p$ de degré $< n$, l'addition étant l'addition des polynômes sur $\mathbb{F}_p$ et la multiplication étant le reste de la division euclidienne par P du produit des polynômes sur $\mathbb{F}_p$.

Corps finis

Un premier exemple de corps fini est $(\mathbb{Z}/p\mathbb{Z}, +, \cdot)$, p premier. On le note généralement $\mathbb{F}_p$ (ou GF(p) en hommage à *Evariste Galois*). Le plus petit corps fini est $\mathbb{F}_2 = \{\overline{0}, \overline{1}\}$, nous l'avons déjà rencontré. Y-a-t-il d'autres corps finis ?

Théorème (E. H. Moore, 1893)

(i) *Si k est un corps fini, alors $\#(k) = p^n$, p premier et $n > 0$.*

(ii) *Réciproquement, pour tout $q = p^n$, p premier et $n > 0$, il existe un corps fini, unique à isomorphisme près, noté $\mathbb{F}_q$ tel que $\#(\mathbb{F}_q) = q$.*

Si P est un polynôme irréductible de degré n sur $\mathbb{F}_p[X]$ (il en existe toujours un), $\mathbb{F}_q \cong \mathbb{F}_p[X]/P$. Autrement dit, $\mathbb{F}_q$ peut être vu comme l'ensemble des polynômes sur $\mathbb{F}_p$ de degré $< n$, l'addition étant l'addition des polynômes sur $\mathbb{F}_p$ et la multiplication étant le reste de la division euclidienne par P du produit des polynômes sur $\mathbb{F}_p$.

Corps finis

Un premier exemple de corps fini est $(\mathbb{Z}/p\mathbb{Z}, +, \cdot)$, p premier. On le note généralement $\mathbb{F}_p$ (ou GF(p) en hommage à *Evariste Galois*). Le plus petit corps fini est $\mathbb{F}_2 = \{\overline{0}, \overline{1}\}$, nous l'avons déjà rencontré. Y-a-t-il d'autres corps finis ?

Théorème (E. H. Moore, 1893)

(i) *Si k est un corps fini, alors $\#(k) = p^n$, p premier et $n > 0$.*

(ii) *Réciproquement, pour tout $q = p^n$, p premier et $n > 0$, il existe un corps fini, unique à isomorphisme près, noté $\mathbb{F}_q$ tel que $\#(\mathbb{F}_q) = q$.*

Si P est un polynôme irréductible de degré n sur $\mathbb{F}_p[X]$ (il en existe toujours un), $\mathbb{F}_q \cong \mathbb{F}_p[X]/P$. Autrement dit, $\mathbb{F}_q$ peut être vu comme l'ensemble des polynômes sur $\mathbb{F}_p$ de degré $< n$, l'addition étant l'addition des polynômes sur $\mathbb{F}_p$ et la multiplication étant le reste de la division euclidienne par P du produit des polynômes sur $\mathbb{F}_p$.

Un exemple : $\mathbb{F}_4$

- On a : $\mathbb{F}_4 = \mathbb{F}_{2^2}$. Cherchons un polynôme irréductible de degré 2 sur $\mathbb{F}_2$. Le polynôme $P(X) = X^2 + X + 1$ en est un (c'est d'ailleurs le seul) car il n'a pas de racine dans $\mathbb{F}_2 = \{0, 1\}$.
- Les éléments de $\mathbb{F}_4$ sont les polynômes de degré 1 à coefficients dans $\mathbb{F}_2$, donc $\mathbb{F}_4 = \{0, 1, X, X + 1\}$. On peut voir aussi $\mathbb{F}_4$ comme l'ensemble des nombres binaires à deux chiffres, c'est à dire :

$$\mathbb{F}_4 = \{\overbrace{00}^{0}, \overbrace{01}^{1}, \overbrace{10}^{X}, \overbrace{11}^{X+1}\},$$

  L'addition étant alors l'opérateur $\oplus$ (ou exclusif). Par exemple : $X + (X + 1) = 10 \oplus 11 = 01 = 1$.
- Exemples de multiplications dans $\mathbb{F}_4$:
  $(X + 1)(X + 1) = X^2 + X + X + 1 = P(X) + X = X$,
  $X \cdot X = X^2 = X^2 + (X + 1) + (X + 1) = P(X) + (X + 1) = X + 1$

Un exemple : $\mathbb{F}_4$

- On a : $\mathbb{F}_4 = \mathbb{F}_{2^2}$. Cherchons un polynôme irréductible de degré 2 sur $\mathbb{F}_2$. Le polynôme $P(X) = X^2 + X + 1$ en est un (c'est d'ailleurs le seul) car il n'a pas de racine dans $\mathbb{F}_2 = \{0, 1\}$.
- Les éléments de $\mathbb{F}_4$ sont les polynômes de degré 1 à coefficients dans $\mathbb{F}_2$, donc $\mathbb{F}_4 = \{0, 1, X, X + 1\}$. On peut voir aussi $\mathbb{F}_4$ comme l'ensemble des nombres binaires à deux chiffres, c'est à dire :

$$\mathbb{F}_4 = \{\overbrace{00}^{0}, \overbrace{01}^{1}, \overbrace{10}^{X}, \overbrace{11}^{X+1}\},$$

L'addition étant alors l'opérateur $\oplus$ (ou exclusif). Par exemple : $X + (X + 1) = 10 \oplus 11 = 01 = 1$.

- Exemples de multiplications dans $\mathbb{F}_4$:
$(X + 1)(X + 1) = X^2 + X + X + 1 = P(X) + X = X,$
$X \cdot X = X^2 = X^2 + (X + 1) + (X + 1) = P(X) + (X + 1) = X + 1$

Un exemple : $\mathbb{F}_4$

- On a : $\mathbb{F}_4 = \mathbb{F}_{2^2}$. Cherchons un polynôme irréductible de degré 2 sur $\mathbb{F}_2$. Le polynôme $P(X) = X^2 + X + 1$ en est un (c'est d'ailleurs le seul) car il n'a pas de racine dans $\mathbb{F}_2 = \{0, 1\}$.
- Les éléments de $\mathbb{F}_4$ sont les polynômes de degré 1 à coefficients dans $\mathbb{F}_2$, donc $\mathbb{F}_4 = \{0, 1, X, X + 1\}$. On peut voir aussi $\mathbb{F}_4$ comme l'ensemble des nombres binaires à deux chiffres, c'est à dire :

$$\mathbb{F}_4 = \{\overbrace{00}^{0}, \overbrace{01}^{1}, \overbrace{10}^{X}, \overbrace{11}^{X+1}\},$$

  L'addition étant alors l'opérateur $\oplus$ (ou exclusif). Par exemple : $X + (X + 1) = 10 \oplus 11 = 01 = 1$.
- Exemples de multiplications dans $\mathbb{F}_4$:
  $(X + 1)(X + 1) = X^2 + X + X + 1 = P(X) + X = X$,
  $X \cdot X = X^2 = X^2 + (X + 1) + (X + 1) = P(X) + (X + 1) = X + 1$

Tables de $\mathbb{F}_4$

$+$	0	1	X	$X+1$
0	0	1	X	$X+1$
1	1	0	$X+1$	X
X	X	$X+1$	0	1
$X+1$	$X+1$	X	1	0

Addition

$\cdot$	0	1	X	$X+1$
0	0	0	0	0
1	0	1	X	$X+1$
X	0	X	$X+1$	1
$X+1$	0	$X+1$	1	X

Multiplication

Tables de $\mathbb{F}_4$

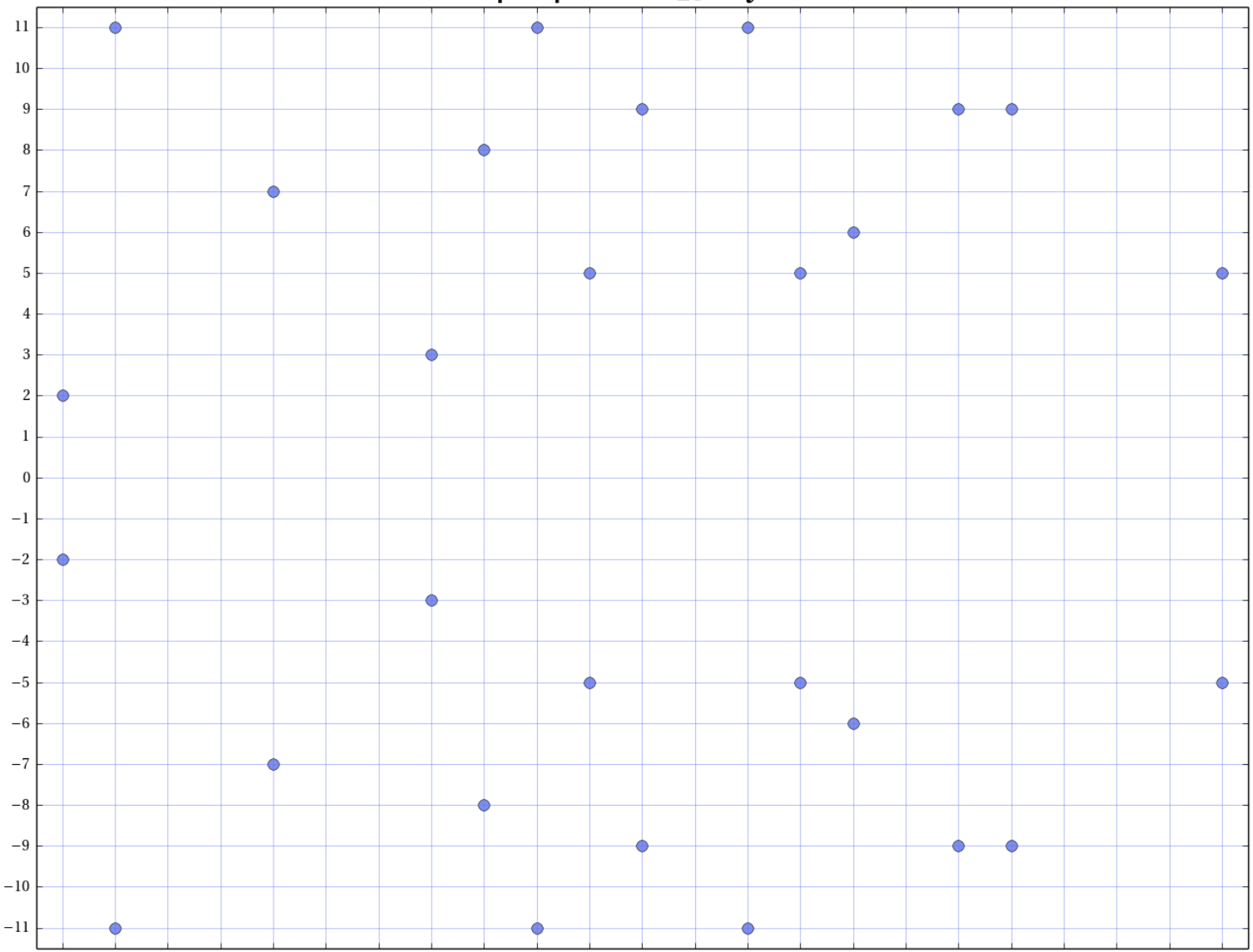
$+$	0	1	X	$X+1$
0	0	1	X	$X+1$
1	1	0	$X+1$	X
X	X	$X+1$	0	1
$X+1$	$X+1$	X	1	0

Addition

$\cdot$	0	1	X	$X+1$
0	0	0	0	0
1	0	1	X	$X+1$
X	0	X	$X+1$	1
$X+1$	0	$X+1$	1	X

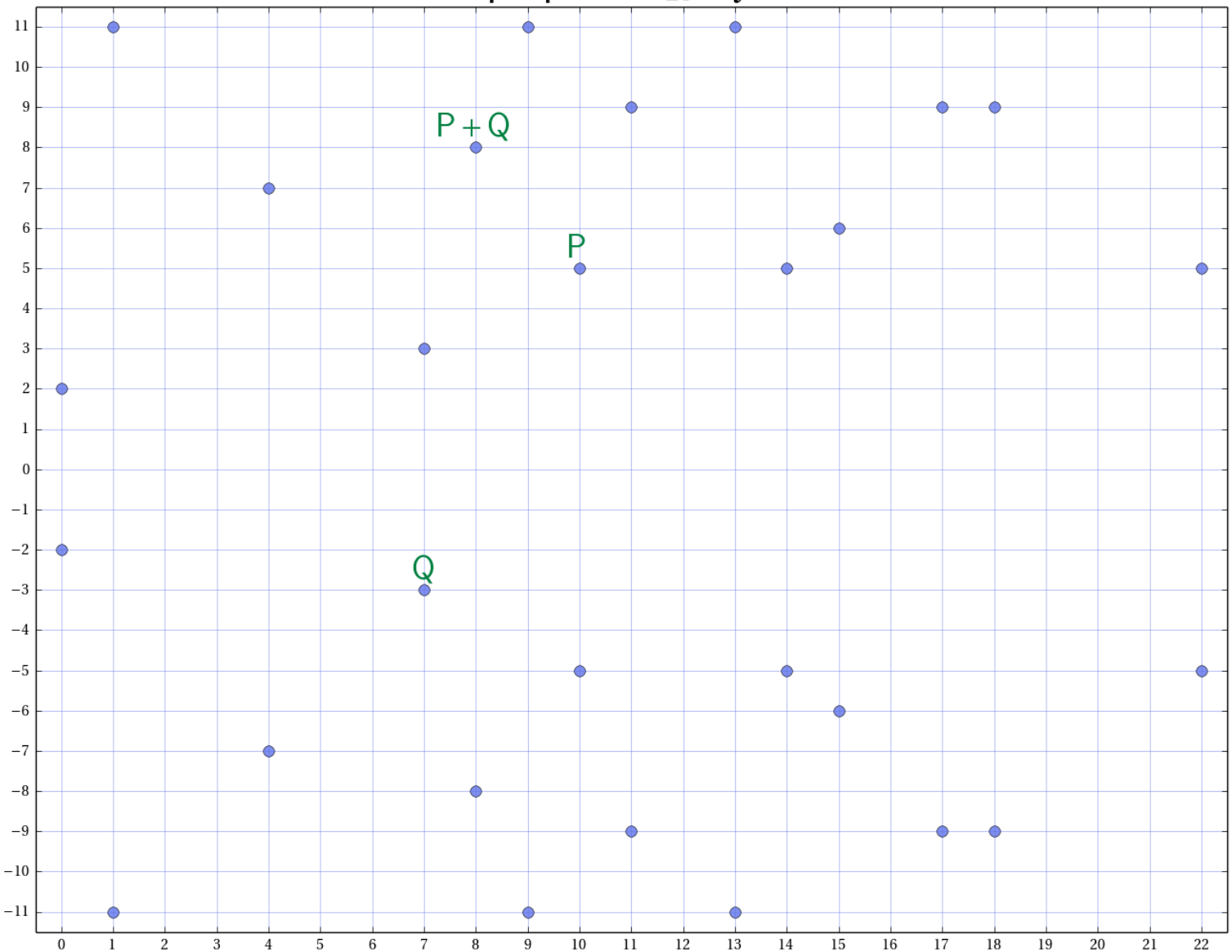
Multiplication

Courbe elliptique $E(\mathbb{F}_{23}) : y^2 = x^3 + x + 4$



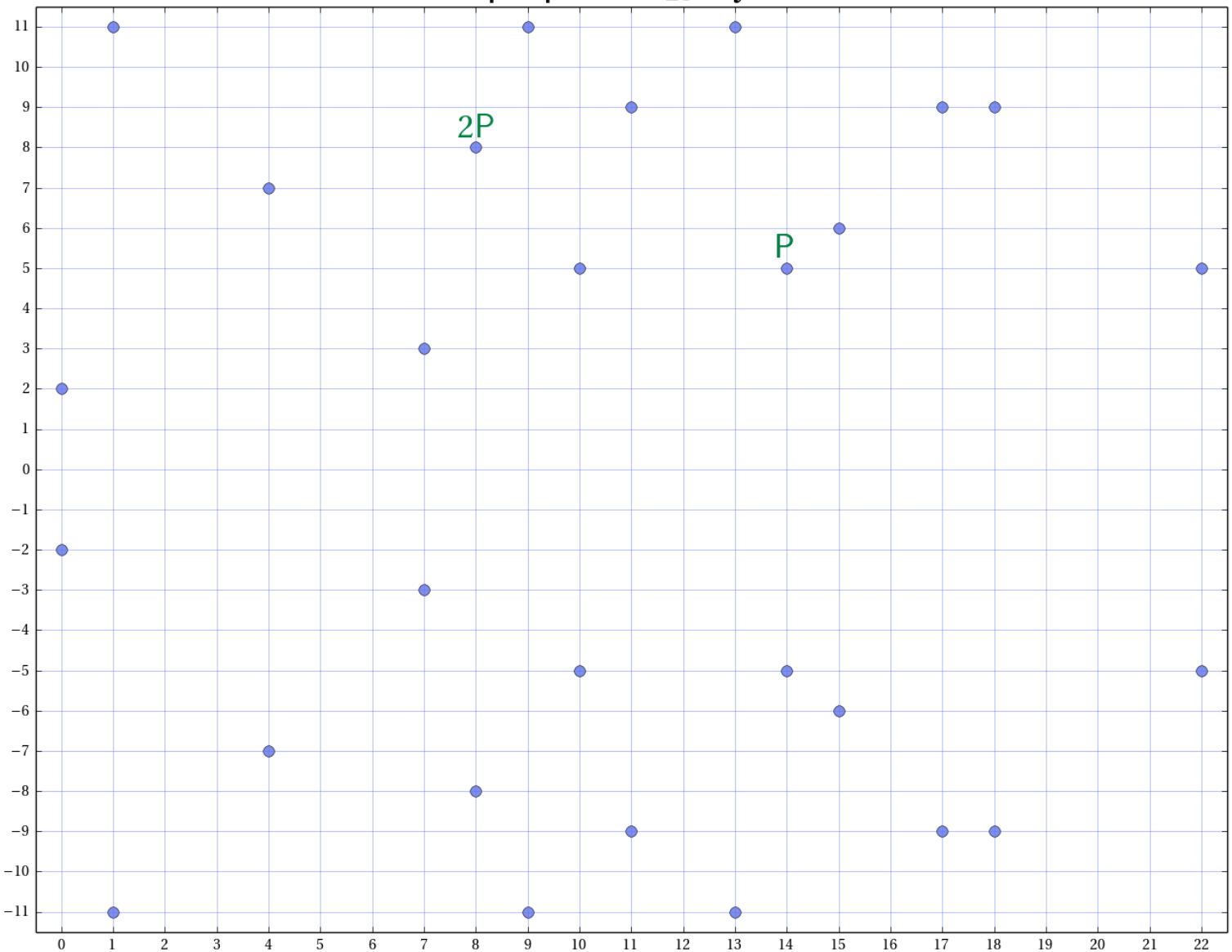
$\#(E(\mathbb{F}_{23})) = 29,\ E(\mathbb{F}_{23}) \simeq \mathbb{Z}/29\mathbb{Z}$

Courbe elliptique $E(\mathbb{F}_{23}) : y^2 = x^3 + x + 4$

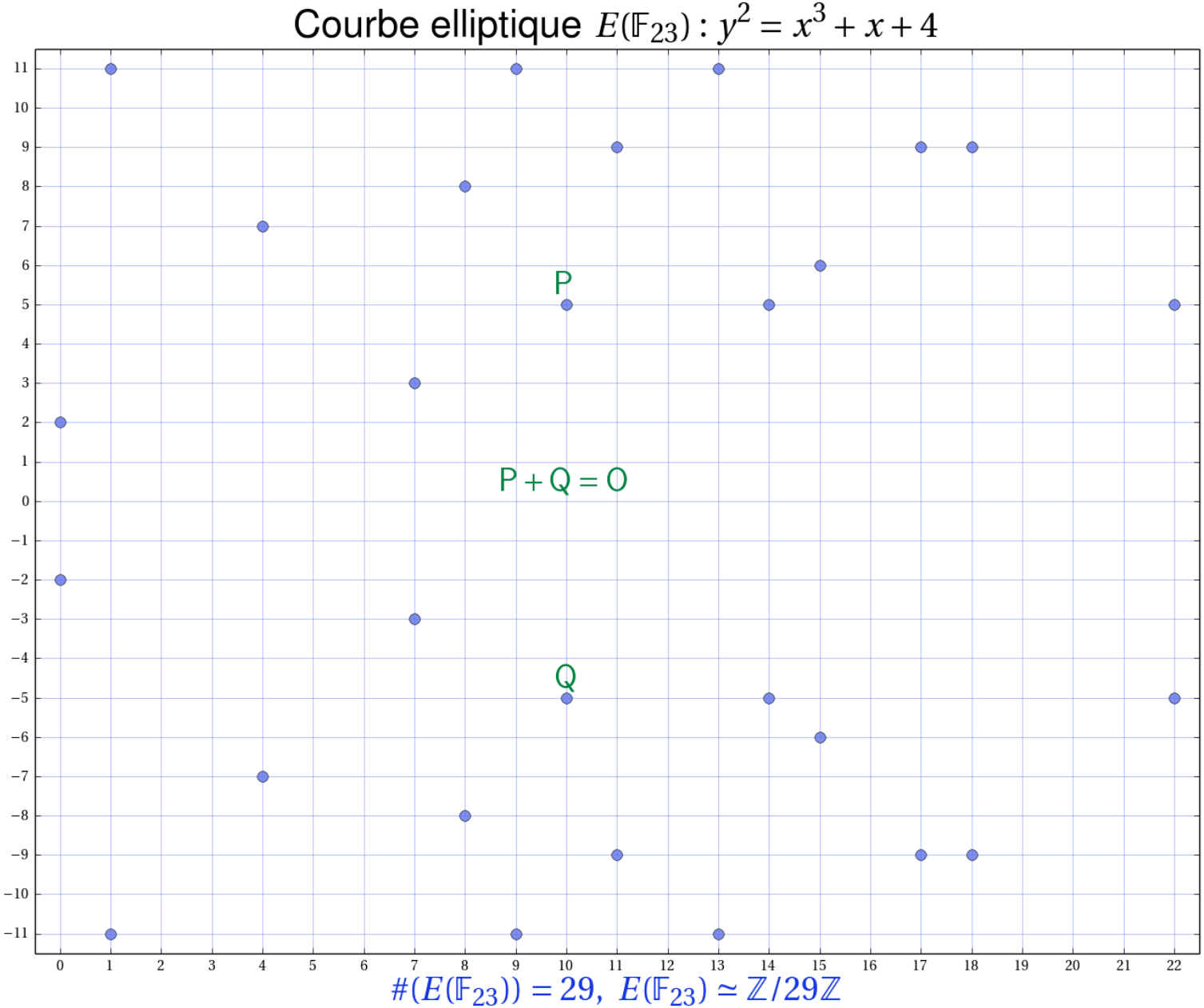


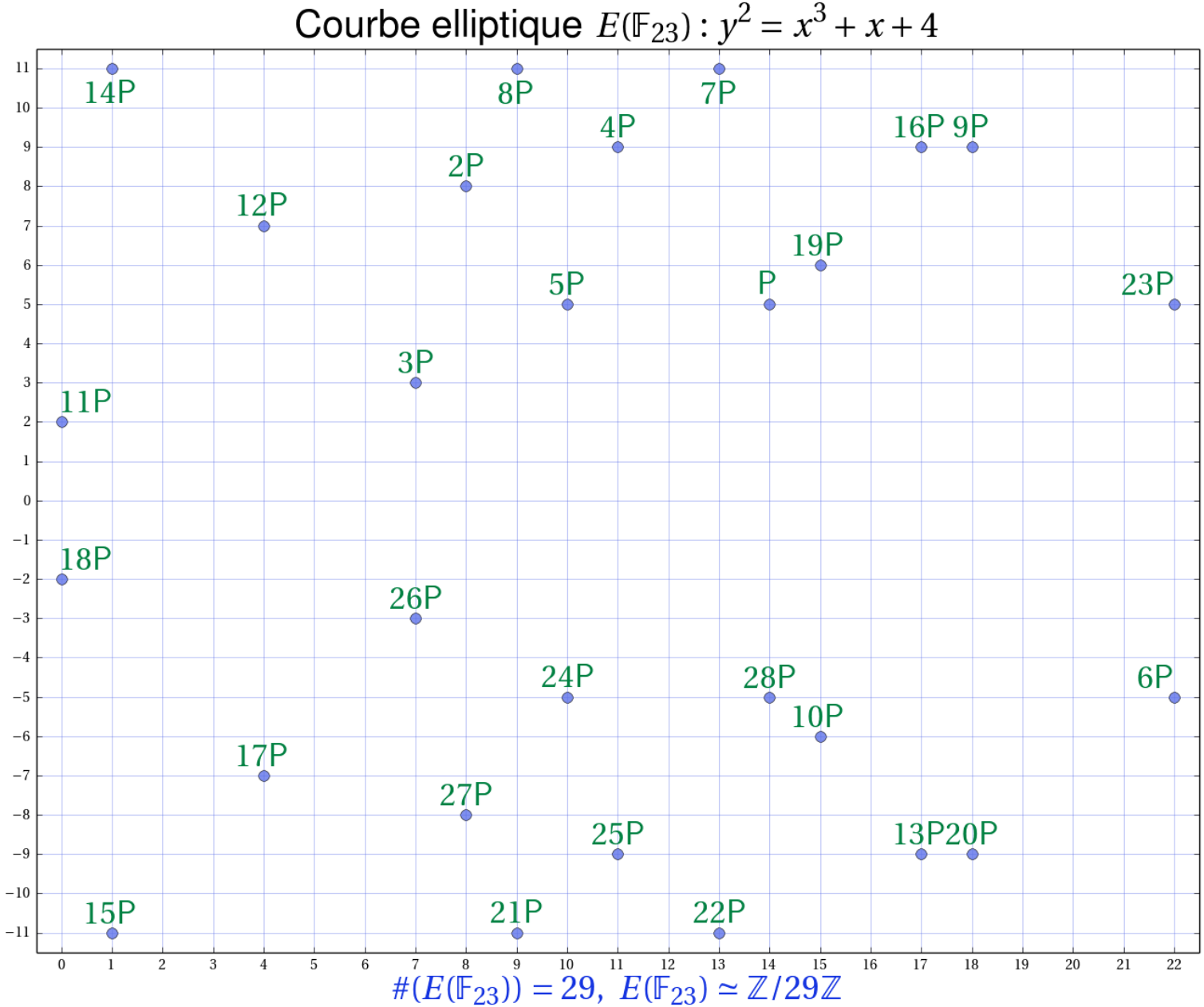
$\#(E(\mathbb{F}_{23})) = 29,\ E(\mathbb{F}_{23}) \simeq \mathbb{Z}/29\mathbb{Z}$

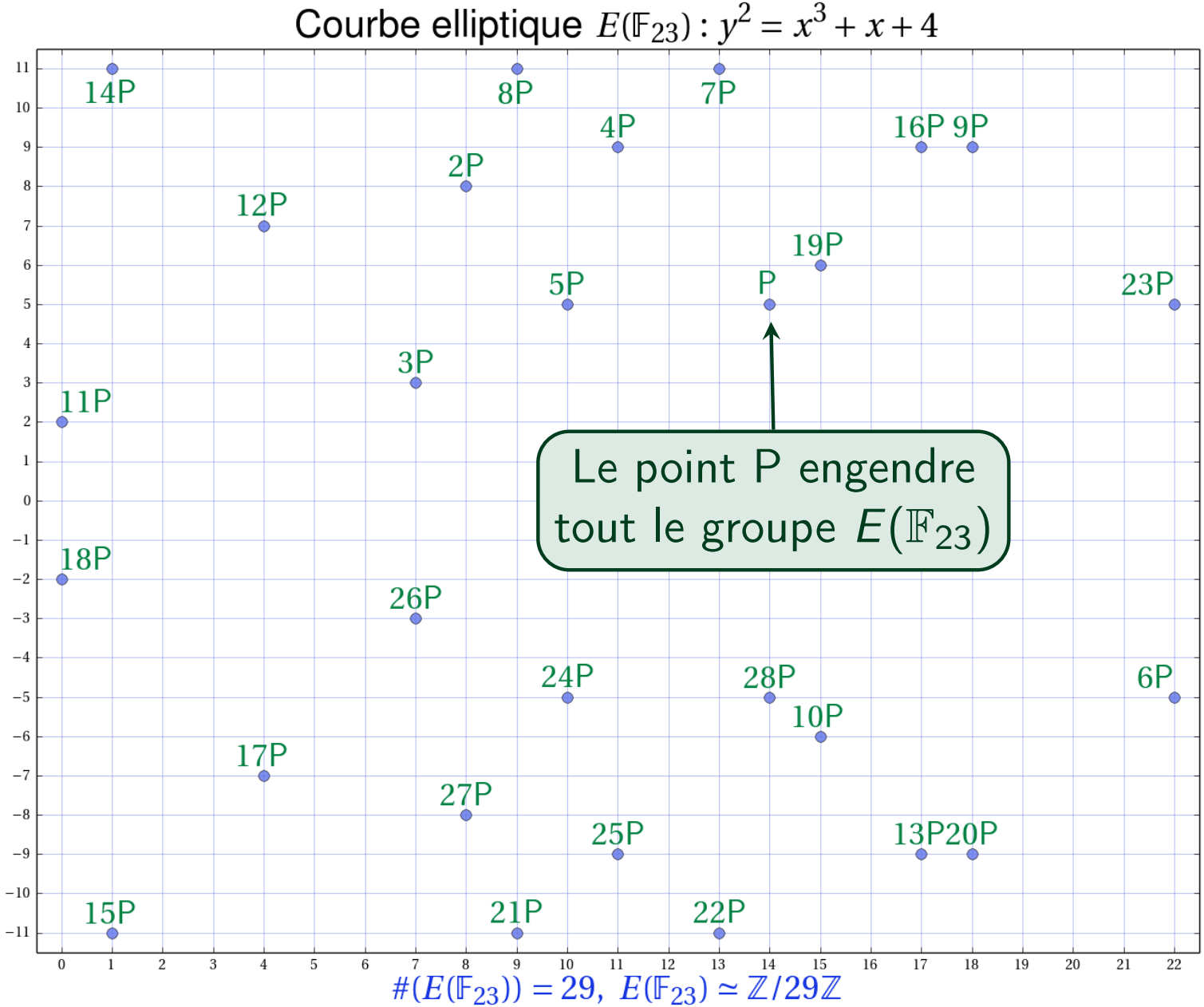
Courbe elliptique $E(\mathbb{F}_{23}) : y^2 = x^3 + x + 4$



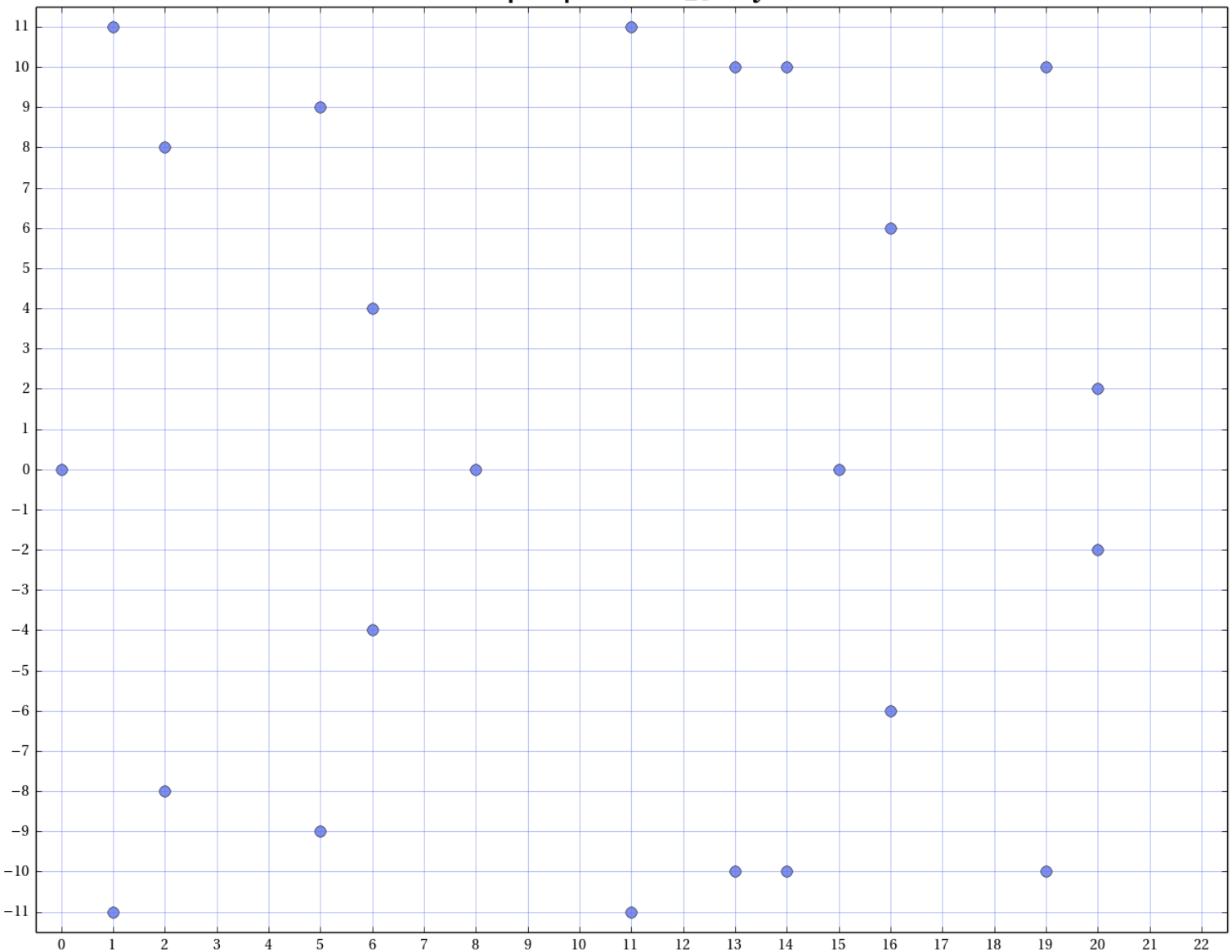
$\#(E(\mathbb{F}_{23})) = 29,\ E(\mathbb{F}_{23}) \simeq \mathbb{Z}/29\mathbb{Z}$


Courbe elliptique $E(\mathbb{F}_{23}) : y^2 = x^3 + x + 4$
P
P + Q = O
Q
$\#(E(\mathbb{F}_{23})) = 29,\ E(\mathbb{F}_{23}) \simeq \mathbb{Z}/29\mathbb{Z}$

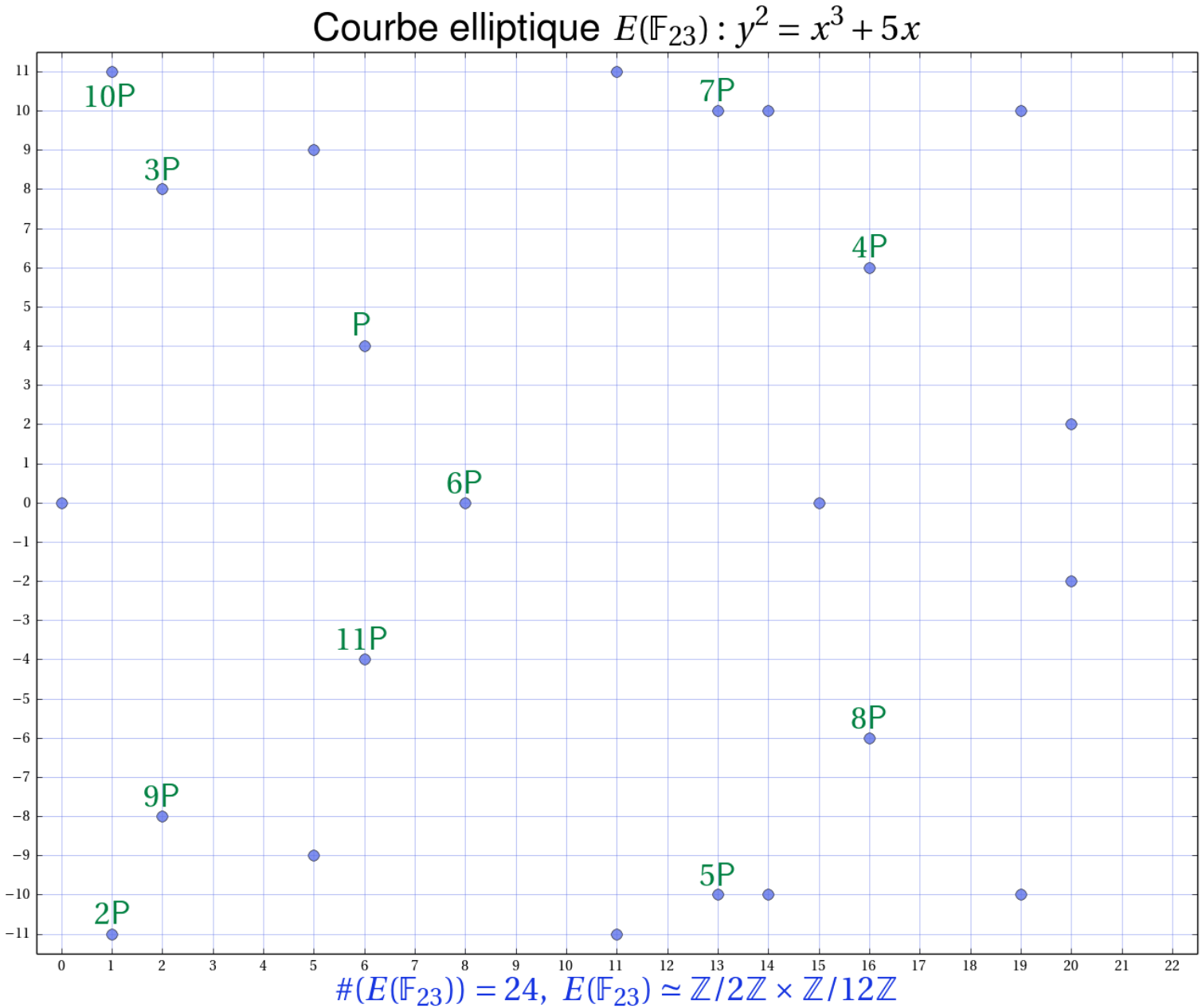

Courbe elliptique $E(\mathbb{F}_{23}) : y^2 = x^3 + x + 4$
14P
8P
7P
4P
16P 9P
2P
12P
19P
5P
P
23P
3P
11P
18P
26P
24P
28P
6P
10P
17P
27P
25P
13P20P
15P
21P
22P
$\#(E(\mathbb{F}_{23})) = 29,\ E(\mathbb{F}_{23}) \simeq \mathbb{Z}/29\mathbb{Z}$

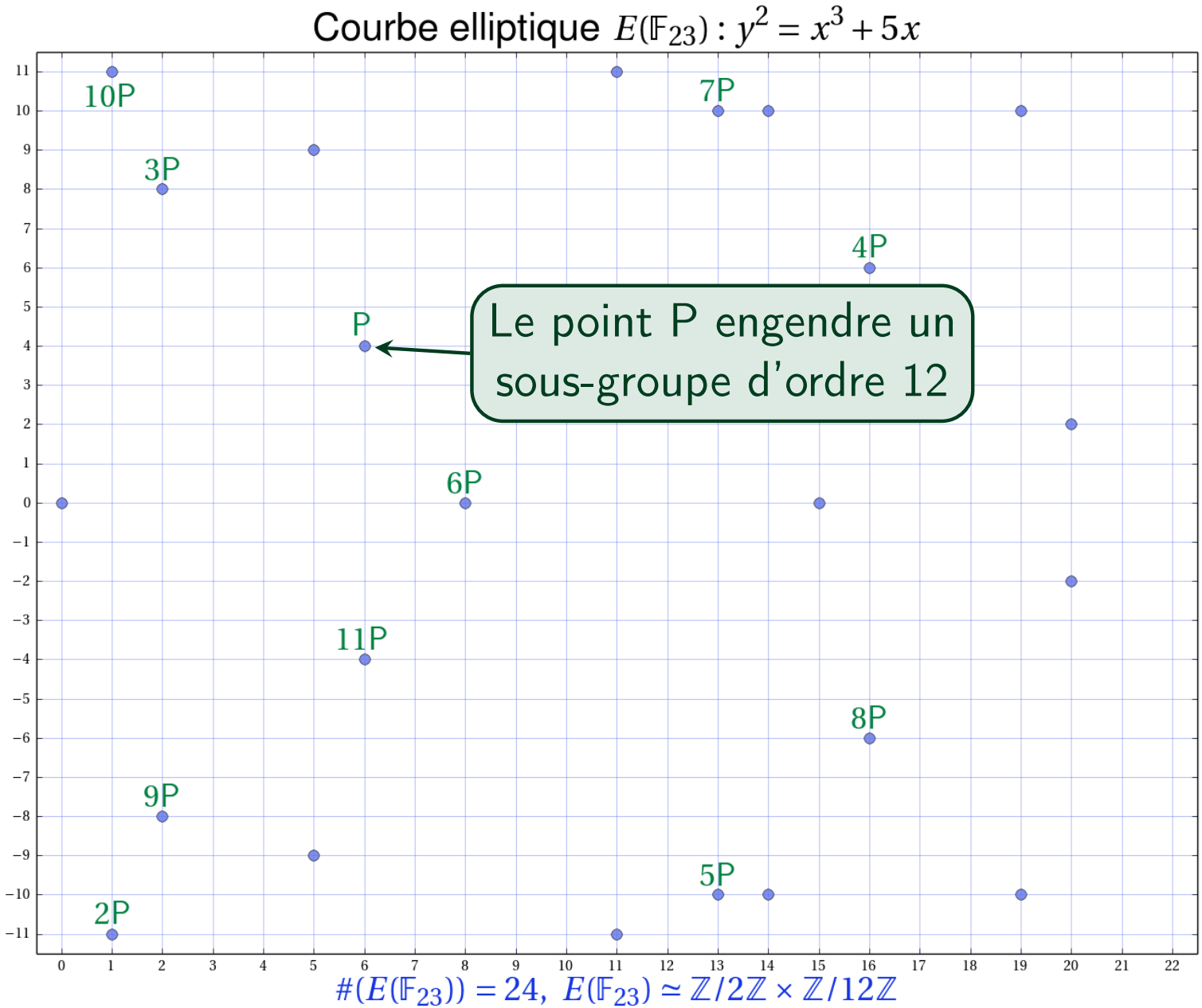

Courbe elliptique $E(\mathbb{F}_{23}) : y^2 = x^3 + x + 4$
P
2P
3P
4P
5P
6P
7P
8P
9P
10P
11P
12P
13P
14P
15P
16P
17P
18P
19P
20P
21P
22P
23P
24P
25P
26P
27P
28P
Le point P engendre tout le groupe $E(\mathbb{F}_{23})$
$\#(E(\mathbb{F}_{23})) = 29,\ E(\mathbb{F}_{23}) \simeq \mathbb{Z}/29\mathbb{Z}$

Courbe elliptique $E(\mathbb{F}_{23}) : y^2 = x^3 + 5x$

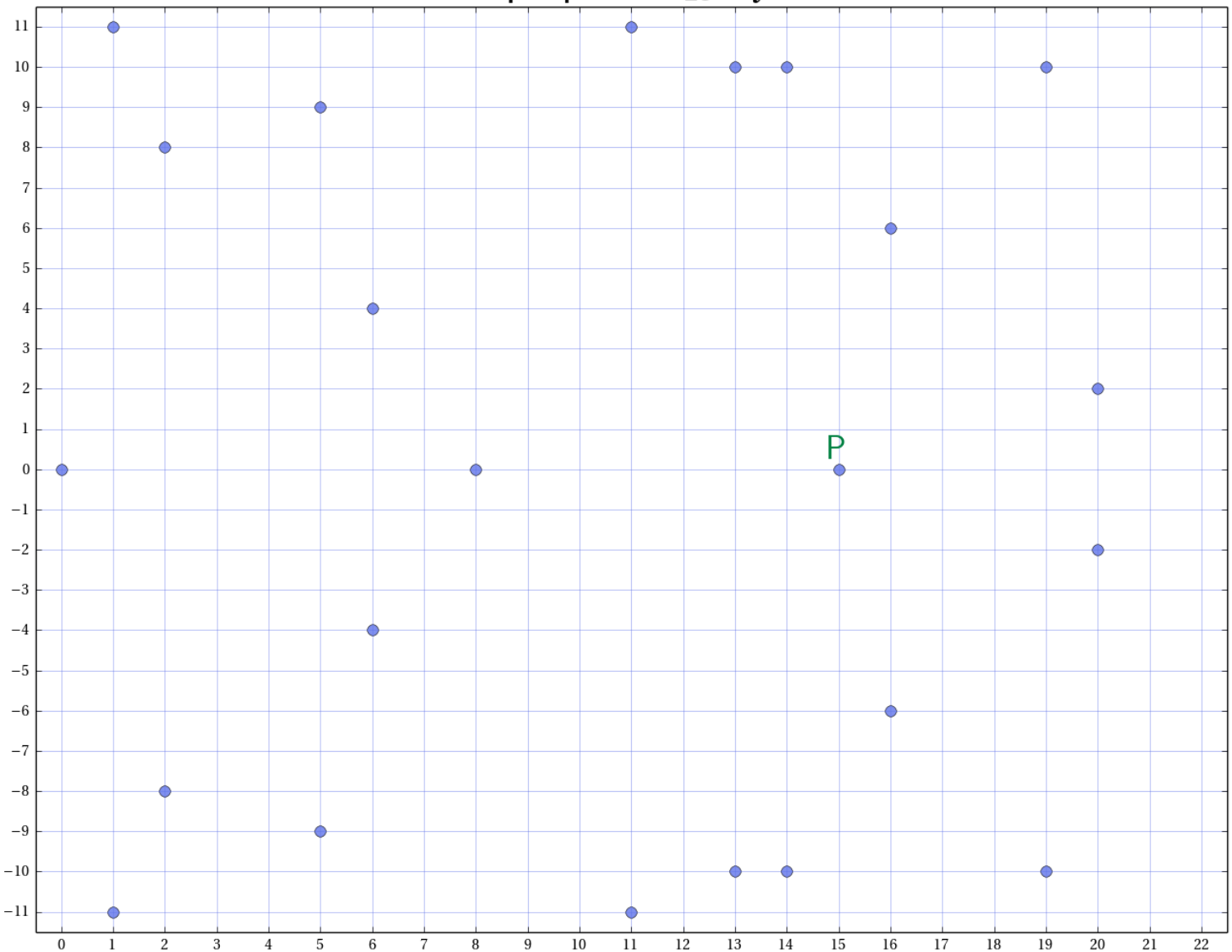


$\#(E(\mathbb{F}_{23})) = 24,\ E(\mathbb{F}_{23}) \simeq \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/12\mathbb{Z}$


Courbe elliptique $E(\mathbb{F}_{23}) : y^2 = x^3 + 5x$
10P
7P
3P
4P
P
6P
11P
8P
9P
5P
2P
$\#(E(\mathbb{F}_{23})) = 24,\ E(\mathbb{F}_{23}) \simeq \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/12\mathbb{Z}$


Courbe elliptique $E(\mathbb{F}_{23}) : y^2 = x^3 + 5x$
10P
7P
3P
4P
P
Le point P engendre un sous-groupe d'ordre 12
6P
11P
8P
9P
5P
2P
$\#(E(\mathbb{F}_{23})) = 24,\ E(\mathbb{F}_{23}) \simeq \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/12\mathbb{Z}$

Courbe elliptique $E(\mathbb{F}_{23}) : y^2 = x^3 + 5x$



$\#(E(\mathbb{F}_{23})) = 24,\ E(\mathbb{F}_{23}) \simeq \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/12\mathbb{Z}$

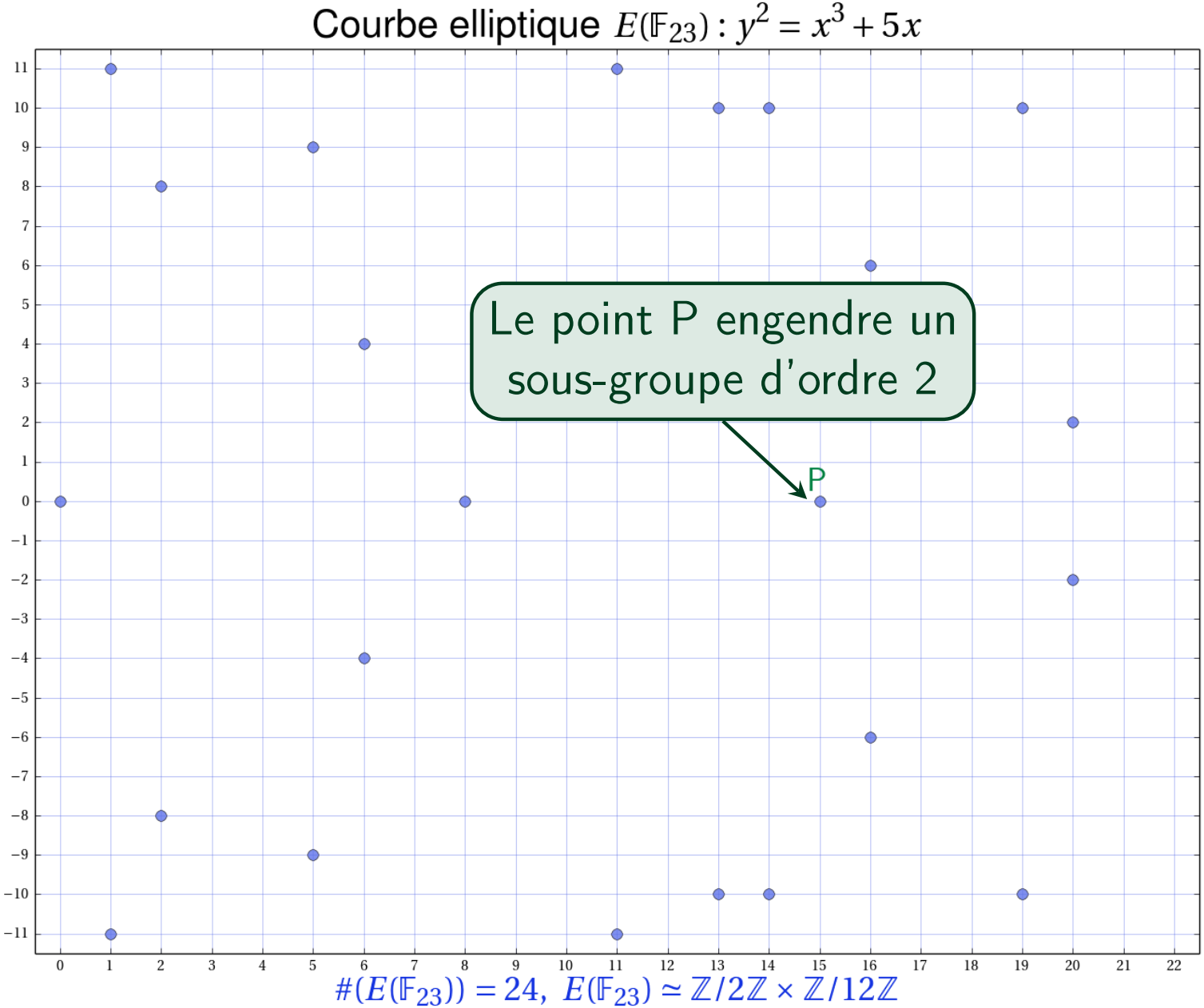

Courbe elliptique $E(\mathbb{F}_{23}) : y^2 = x^3 + 5x$
Le point P engendre un sous-groupe d'ordre 2
P
$\#(E(\mathbb{F}_{23})) = 24,\ E(\mathbb{F}_{23}) \simeq \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/12\mathbb{Z}$

Schéma du protocole de Diffie-Hellman

Schéma du protocole de Diffie-Hellman

Ce qui est public : $\mathbb{F}_q$, $E : y^2 = x^3 + ax + b$ avec $a, b \in \mathbb{F}_q$, et P, n resp. générateur et ordre d'un sous-groupe de $E(\mathbb{F}_q)$

Alice

Bob

Schéma du protocole de Diffie-Hellman

Ce qui est public : $\mathbb{F}_q$, $E : y^2 = x^3 + ax + b$ avec $a, b \in \mathbb{F}_q$, et P, n resp. générateur et ordre d'un sous-groupe de $E(\mathbb{F}_q)$

Alice choisit $\alpha \in]0, n[$ secret

Alice

Bob

Schéma du protocole de Diffie-Hellman

Ce qui est public : $\mathbb{F}_q$, $E : y^2 = x^3 + ax + b$ avec $a, b \in \mathbb{F}_q$, et P, n resp. générateur et ordre d'un sous-groupe de $E(\mathbb{F}_q)$

Alice choisit $\alpha \in]0, n[$ secret

Bob choisit $\beta \in]0, n[$ secret

Alice

Bob

Schéma du protocole de Diffie-Hellman

Ce qui est public : $\mathbb{F}_q$, $E : y^2 = x^3 + ax + b$ avec $a, b \in \mathbb{F}_q$, et P, n resp. générateur et ordre d'un sous-groupe de $E(\mathbb{F}_q)$

Alice choisit $\alpha \in]0, n[$ secret

Bob choisit $\beta \in]0, n[$ secret

Alice

Bob

Alice calcule αP

Schéma du protocole de Diffie-Hellman

Ce qui est public : $\mathbb{F}_q$, $E : y^2 = x^3 + ax + b$ avec $a, b \in \mathbb{F}_q$, et P, n resp. générateur et ordre d'un sous-groupe de $E(\mathbb{F}_q)$

Alice choisit $\alpha \in]0, n[$ secret

Bob choisit $\beta \in]0, n[$ secret

Alice

Bob

Alice calcule αP

Bob calcule βP

Schéma du protocole de Diffie-Hellman

Ce qui est public : $\mathbb{F}_q$, $E : y^2 = x^3 + ax + b$ avec $a, b \in \mathbb{F}_q$, et P, n resp. générateur et ordre d'un sous-groupe de $E(\mathbb{F}_q)$

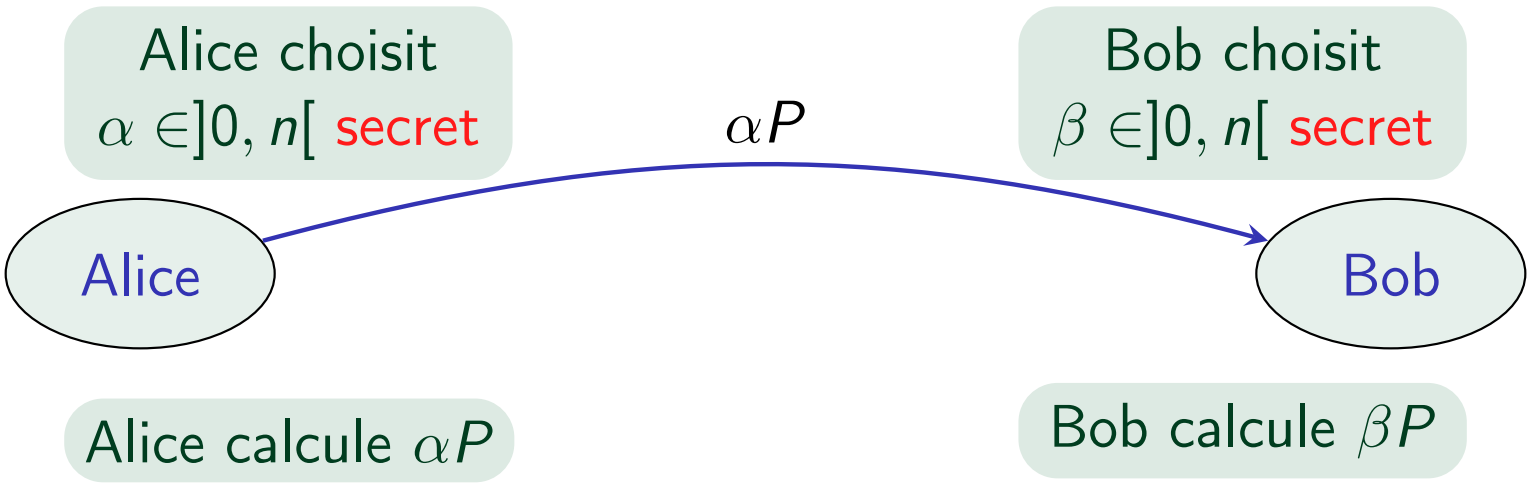


Schéma du protocole de Diffie-Hellman

Ce qui est public : $\mathbb{F}_q$, $E : y^2 = x^3 + ax + b$ avec $a, b \in \mathbb{F}_q$, et P, n resp. générateur et ordre d'un sous-groupe de $E(\mathbb{F}_q)$

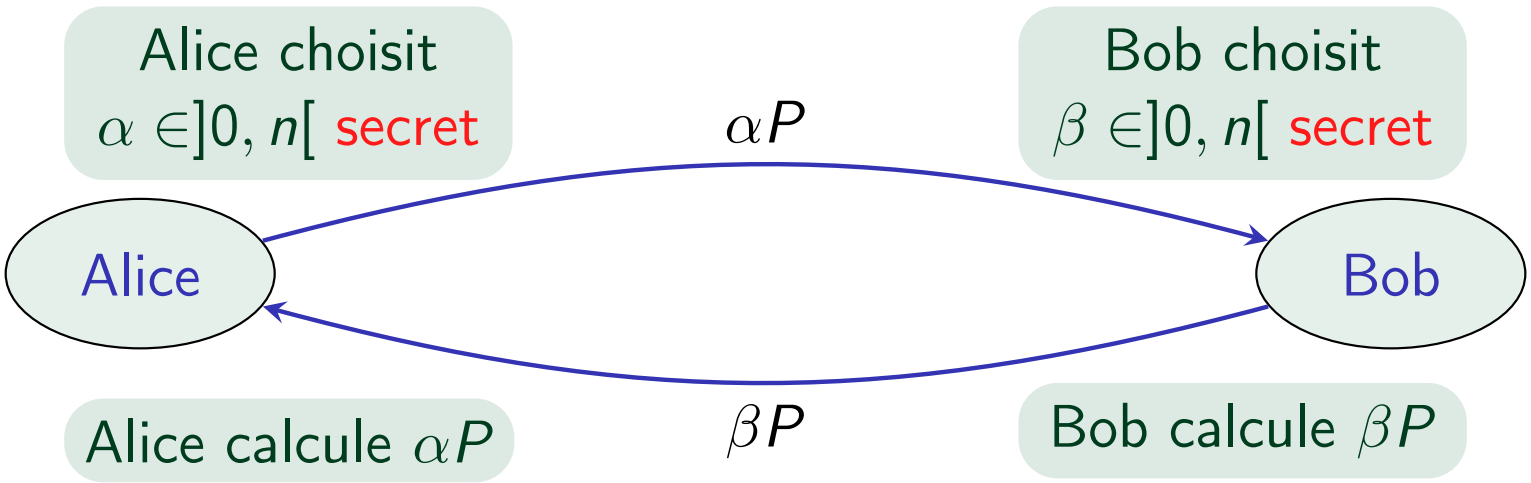


Schéma du protocole de Diffie-Hellman

Ce qui est public : $\mathbb{F}_q$, $E : y^2 = x^3 + ax + b$ avec $a, b \in \mathbb{F}_q$, et P, n resp. générateur et ordre d'un sous-groupe de $E(\mathbb{F}_q)$

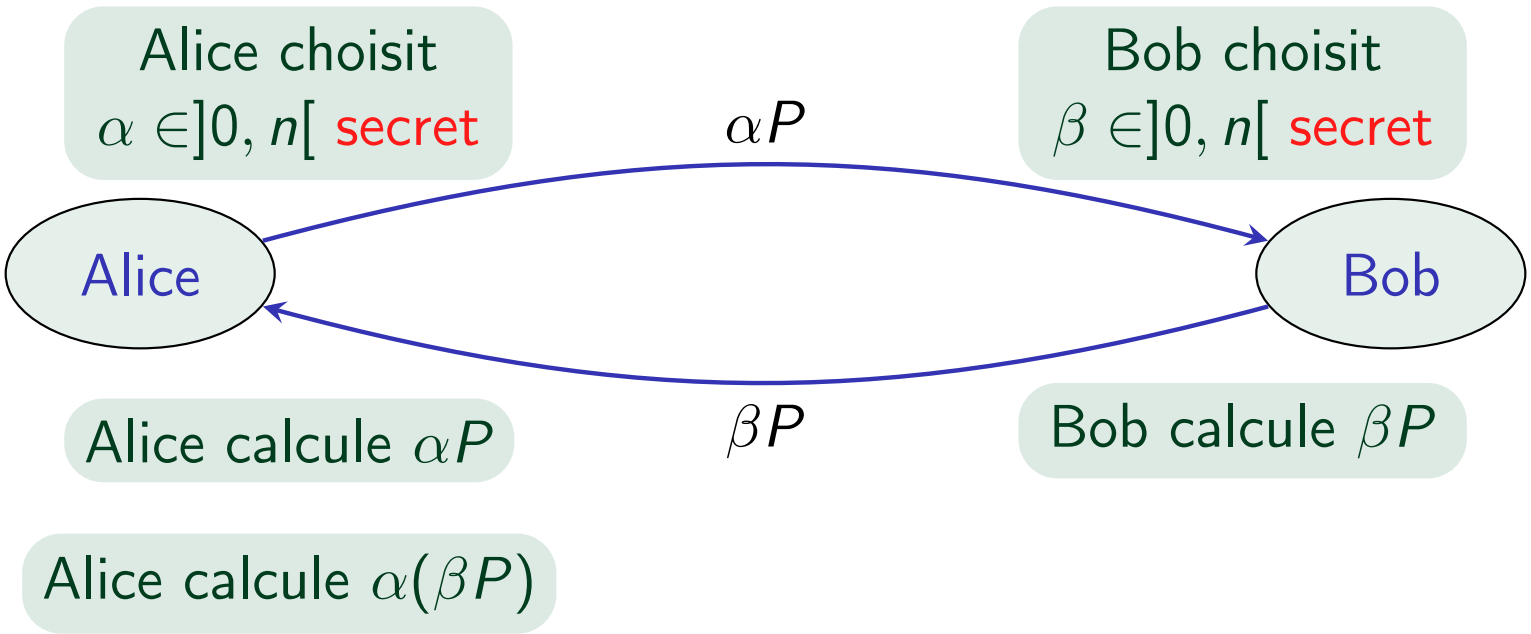


Schéma du protocole de Diffie-Hellman

Ce qui est public : $\mathbb{F}_q$, $E : y^2 = x^3 + ax + b$ avec $a, b \in \mathbb{F}_q$, et P, n resp. générateur et ordre d'un sous-groupe de $E(\mathbb{F}_q)$

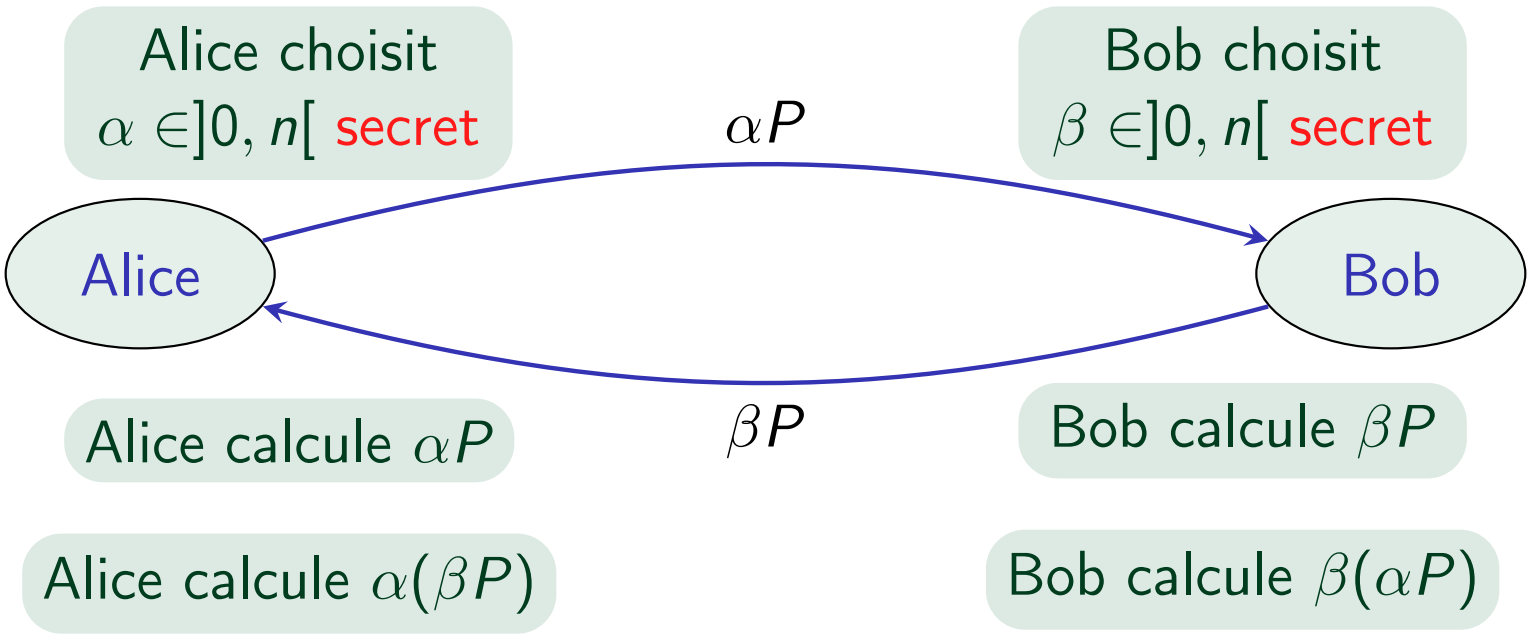
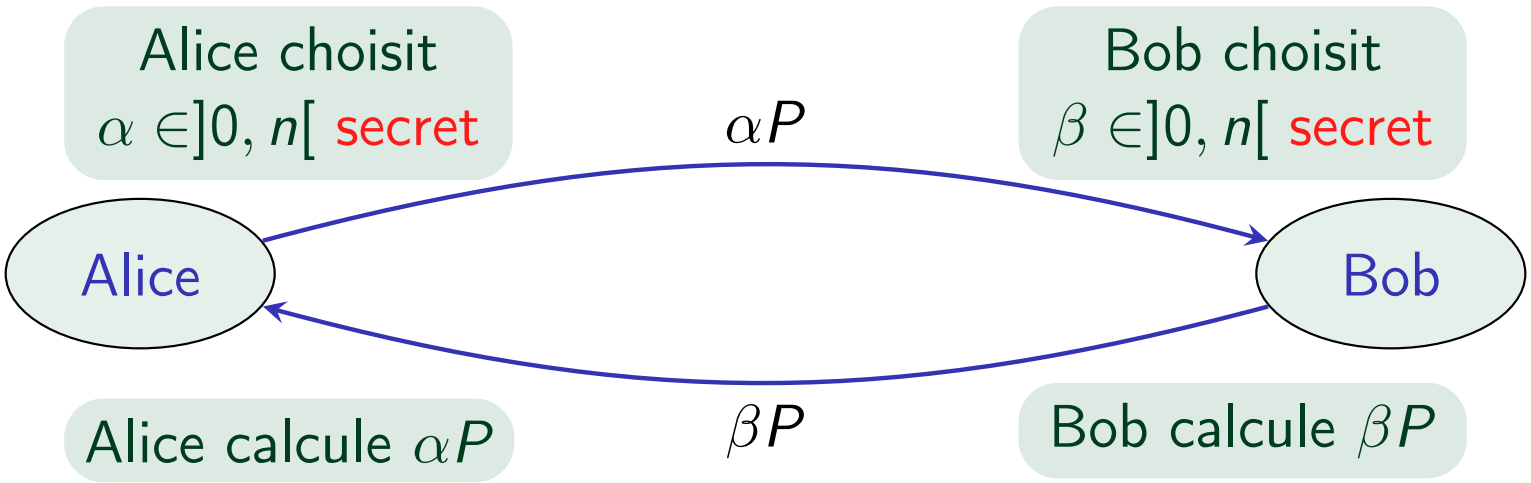


Schéma du protocole de Diffie-Hellman

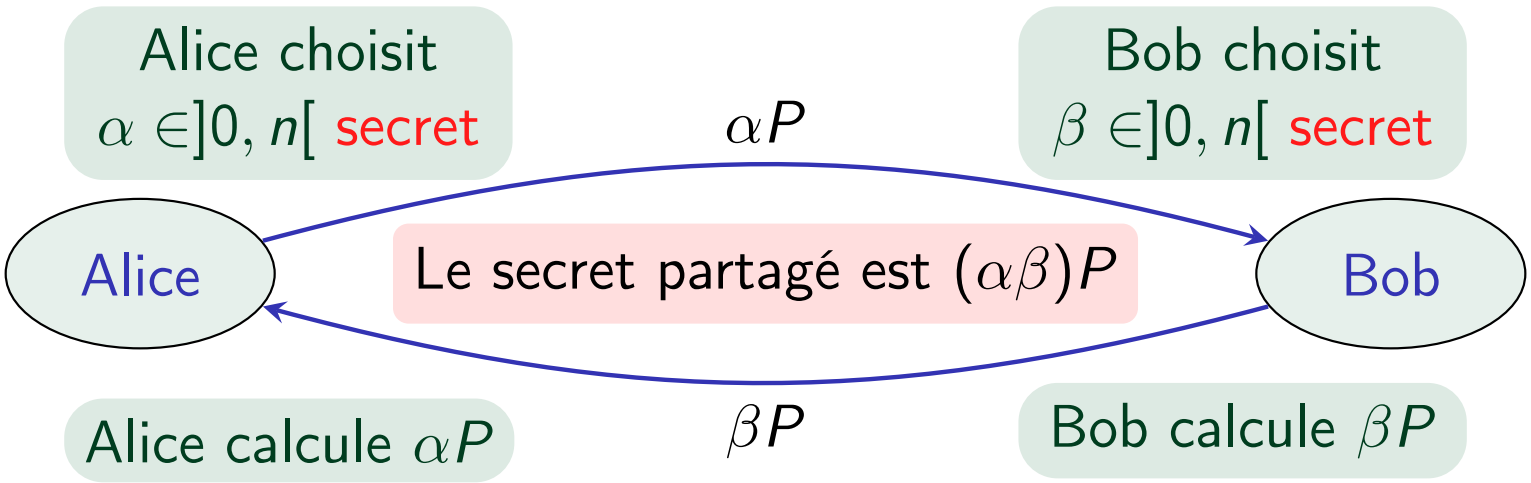
Ce qui est public : $\mathbb{F}_q$, $E : y^2 = x^3 + ax + b$ avec $a, b \in \mathbb{F}_q$, et P, n resp. générateur et ordre d'un sous-groupe de $E(\mathbb{F}_q)$



Alice calcule $\alpha(\beta P)$ $=$ $(\alpha\beta)P$ $=$ Bob calcule $\beta(\alpha P)$

Schéma du protocole de Diffie-Hellman

Ce qui est public : $\mathbb{F}_q$, $E : y^2 = x^3 + ax + b$ avec $a, b \in \mathbb{F}_q$, et P, n resp. générateur et ordre d'un sous-groupe de $E(\mathbb{F}_q)$



Alice calcule $\alpha(\beta P)$ $=$ $(\alpha\beta)P$ $=$ Bob calcule $\beta(\alpha P)$

Calcul de αP

- Le calcul de $\alpha P = \overbrace{P + \cdots + P}^{\alpha \text{ fois}}$ s'effectue comme précédemment avec l'algorithme *d'exponentiation par carrés*, la seule différence étant que la notation est ici additive au lieu d'être multiplicative.
- On écrit α en binaire : $\alpha = \alpha_0 + \alpha_1 \cdot 2 + \cdots + \alpha_r \cdot 2^r$ et donc αP se calcule de la manière suivante : $\alpha P = \alpha_0 P + \alpha_1 \cdot 2P + \cdots + \alpha_r \cdot 2^r P$.
- Donc le calcul de αP nécessite en moyenne $\log_2(\alpha)$ doublements et $\frac{1}{2}\log_2(\alpha)$ additions.
- On peut affiner l'algorithme précédent et ainsi réduire le nombre d'additions en moyenne à $\frac{1}{3}\log_2(\alpha)$.
- Ce qu'il faut retenir c'est que le calcul de αP est rapide même si α est "très grand", ce qui est le cas en pratique.

Calcul de αP

- Le calcul de $\alpha P = \overbrace{P + \cdots + P}^{\alpha \text{ fois}}$ s'effectue comme précédemment avec l'algorithme *d'exponentiation par carrés*, la seule différence étant que la notation est ici additive au lieu d'être multiplicative.
- On écrit α en binaire : $\alpha = \alpha_0 + \alpha_1 \cdot 2 + \cdots + \alpha_r \cdot 2^r$ et donc αP se calcule de la manière suivante : $\alpha P = \alpha_0 P + \alpha_1 \cdot 2P + \cdots + \alpha_r \cdot 2^r P$.
- Donc le calcul de αP nécessite en moyenne $\log_2(\alpha)$ doublements et $\frac{1}{2}\log_2(\alpha)$ additions.
- On peut affiner l'algorithme précédent et ainsi réduire le nombre d'additions en moyenne à $\frac{1}{3}\log_2(\alpha)$.
- Ce qu'il faut retenir c'est que le calcul de αP est rapide même si α est "très grand", ce qui est le cas en pratique.

Calcul de αP

- Le calcul de $\alpha P = \overbrace{P + \cdots + P}^{\alpha \text{ fois}}$ s'effectue comme précédemment avec l'algorithme *d'exponentiation par carrés*, la seule différence étant que la notation est ici additive au lieu d'être multiplicative.
- On écrit α en binaire : $\alpha = \alpha_0 + \alpha_1 \cdot 2 + \cdots + \alpha_r \cdot 2^r$ et donc αP se calcule de la manière suivante : $\alpha P = \alpha_0 P + \alpha_1 \cdot 2P + \cdots + \alpha_r \cdot 2^r P$.
- Donc le calcul de αP nécessite en moyenne $\log_2(\alpha)$ doublements et $\frac{1}{2}\log_2(\alpha)$ additions.
- On peut affiner l'algorithme précédent et ainsi réduire le nombre d'additions en moyenne à $\frac{1}{3}\log_2(\alpha)$.
- Ce qu'il faut retenir c'est que le calcul de αP est rapide même si α est "très grand", ce qui est le cas en pratique.

Calcul de αP

- Le calcul de $\alpha P = \overbrace{P + \cdots + P}^{\alpha \text{ fois}}$ s'effectue comme précédemment avec l'algorithme *d'exponentiation par carrés*, la seule différence étant que la notation est ici additive au lieu d'être multiplicative.
- On écrit α en binaire : $\alpha = \alpha_0 + \alpha_1 \cdot 2 + \cdots + \alpha_r \cdot 2^r$ et donc αP se calcule de la manière suivante : $\alpha P = \alpha_0 P + \alpha_1 \cdot 2P + \cdots + \alpha_r \cdot 2^r P$.
- Donc le calcul de αP nécessite en moyenne $\log_2(\alpha)$ doublements et $\frac{1}{2}\log_2(\alpha)$ additions.
- On peut affiner l'algorithme précédent et ainsi réduire le nombre d'additions en moyenne à $\frac{1}{3}\log_2(\alpha)$.
- Ce qu'il faut retenir c'est que le calcul de αP est rapide même si α est "très grand", ce qui est le cas en pratique.

Calcul de αP

- Le calcul de $\alpha P = \overbrace{P + \cdots + P}^{\alpha \text{ fois}}$ s'effectue comme précédemment avec l'algorithme *d'exponentiation par carrés*, la seule différence étant que la notation est ici additive au lieu d'être multiplicative.
- On écrit α en binaire : $\alpha = \alpha_0 + \alpha_1 \cdot 2 + \cdots + \alpha_r \cdot 2^r$ et donc αP se calcule de la manière suivante : $\alpha P = \alpha_0 P + \alpha_1 \cdot 2P + \cdots + \alpha_r \cdot 2^r P$.
- Donc le calcul de αP nécessite en moyenne $\log_2(\alpha)$ doublements et $\frac{1}{2}\log_2(\alpha)$ additions.
- On peut affiner l'algorithme précédent et ainsi réduire le nombre d'additions en moyenne à $\frac{1}{3}\log_2(\alpha)$.
- Ce qu'il faut retenir c'est que le calcul de αP est rapide même si α est "très grand", ce qui est le cas en pratique.

Nombre de points

Si E est une courbe elliptique sur $\mathbb{F}_q$, alors $E(\mathbb{F}_q)$ est bien évidemment fini. On peut alors se demander quel est le nombre de points de E. C'est une question en général très difficile. Néanmoins, nous avons le résulat suivant :

Théorème (Hasse, 1933)

Soit E une courbe elliptique sur $\mathbb{F}_q$ ($q = p^n$, p premier et $n > 0$), alors on a :

$$|\#(E(\mathbb{F}_q)) - (q+1)| \leq 2\sqrt{q}$$

De plus si p est premier, alors pour tout t entier dans l'intervalle $[-2\sqrt{p}, 2\sqrt{p}]$, il existe une courbe elliptique E définie sur $\mathbb{F}_p$ telle que $\#(E(\mathbb{F}_p)) = p + 1 - t$.

Il existe aujourd'hui des méthodes particulièrement sophistiquées pour calculer efficacement $\#(E(\mathbb{F}_q))$.

Nombre de points

Si E est une courbe elliptique sur $\mathbb{F}_q$, alors $E(\mathbb{F}_q)$ est bien évidemment fini. On peut alors se demander quel est le nombre de points de E. C'est une question en général très difficile. Néanmoins, nous avons le résulat suivant :

Théorème (Hasse, 1933)

Soit E une courbe elliptique sur $\mathbb{F}_q$ ($q = p^n$, p premier et $n > 0$), alors on a :

$$|\#(E(\mathbb{F}_q)) - (q+1)| \leq 2\sqrt{q}$$

De plus si p est premier, alors pour tout t entier dans l'intervalle $[-2\sqrt{p}, 2\sqrt{p}]$, il existe une courbe elliptique E définie sur $\mathbb{F}_p$ telle que $\#(E(\mathbb{F}_p)) = p + 1 - t$.

Il existe aujourd'hui des méthodes particulièrement sophistiquées pour calculer efficacement $\#(E(\mathbb{F}_q))$.

Nombre de points

Si E est une courbe elliptique sur $\mathbb{F}_q$, alors $E(\mathbb{F}_q)$ est bien évidemment fini. On peut alors se demander quel est le nombre de points de E. C'est une question en général très difficile. Néanmoins, nous avons le résulat suivant :

Théorème (Hasse, 1933)

Soit E une courbe elliptique sur $\mathbb{F}_q$ ($q = p^n$, p premier et $n > 0$), alors on a :

$$|\#(E(\mathbb{F}_q)) - (q+1)| \leq 2\sqrt{q}$$

De plus si p est premier, alors pour tout t entier dans l'intervalle $[-2\sqrt{p}, 2\sqrt{p}]$, il existe une courbe elliptique E définie sur $\mathbb{F}_p$ telle que $\#(E(\mathbb{F}_p)) = p + 1 - t$.

Il existe aujourd'hui des méthodes particulièrement sophistiquées pour calculer efficacement $\#(E(\mathbb{F}_q))$.

Nombre de points

Si E est une courbe elliptique sur $\mathbb{F}_q$, alors $E(\mathbb{F}_q)$ est bien évidemment fini. On peut alors se demander quel est le nombre de points de E. C'est une question en général très difficile. Néanmoins, nous avons le résulat suivant :

Théorème (Hasse, 1933)

Soit E une courbe elliptique sur $\mathbb{F}_q$ ($q = p^n$, p premier et $n > 0$), alors on a :

$$|\#(E(\mathbb{F}_q)) - (q+1)| \leq 2\sqrt{q}$$

De plus si p est premier, alors pour tout t entier dans l'intervalle $[-2\sqrt{p}, 2\sqrt{p}]$, il existe une courbe elliptique E définie sur $\mathbb{F}_p$ telle que $\#(E(\mathbb{F}_p)) = p + 1 - t$.

Il existe aujourd'hui des méthodes particulièrement sophistiquées pour calculer efficacement $\#(E(\mathbb{F}_q))$.

Structure du groupe $E(\mathbb{F}_q)$

Théorème

Le groupe $E(\mathbb{F}_q)$ est soit un groupe cyclique soit un produit de deux groupes cycliques. Plus précisément, on a dans le premier cas :

$$E(\mathbb{F}_q) \cong \mathbb{Z}/n\mathbb{Z} \quad \text{où } n = \#(E(\mathbb{F}_q)),$$

et dans le second cas :

$$E(\mathbb{F}_q) \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \quad \text{où } m \mid n \text{ et } m \mid q-1.$$

Si on reprend les exemples précédents :

- $q = 23$, $E : y^2 = x^3 + x + 4$, $\#(E(\mathbb{F}_{23})) = 29$. L'entier 29 est premier donc $E(\mathbb{F}_{23})$ est cyclique et isomorphe à $\mathbb{Z}/29\mathbb{Z}$.
- $q = 23$, $E : y^2 = x^3 + 5x$, $\#(E(\mathbb{F}_{23})) = 24$. $E(\mathbb{F}_{23})$ n'est pas cyclique. On a $m \mid 24$ et $m \mid 22$ donc $m = 2$. Par conséquent : $E(\mathbb{F}_{23}) \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/12\mathbb{Z}$.

Structure du groupe $E(\mathbb{F}_q)$

Théorème

Le groupe $E(\mathbb{F}_q)$ est soit un groupe cyclique soit un produit de deux groupes cycliques. Plus précisément, on a dans le premier cas :

$$E(\mathbb{F}_q) \cong \mathbb{Z}/n\mathbb{Z} \quad \text{où } n = \#(E(\mathbb{F}_q)),$$

et dans le second cas :

$$E(\mathbb{F}_q) \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \quad \text{où } m \mid n \text{ et } m \mid q-1.$$

Si on reprend les exemples précédents :

- $q = 23$, $E : y^2 = x^3 + x + 4$, $\#(E(\mathbb{F}_{23})) = 29$. L'entier 29 est premier donc $E(\mathbb{F}_{23})$ est cyclique et isomorphe à $\mathbb{Z}/29\mathbb{Z}$.
- $q = 23$, $E : y^2 = x^3 + 5x$, $\#(E(\mathbb{F}_{23})) = 24$. $E(\mathbb{F}_{23})$ n'est pas cyclique. On a $m \mid 24$ et $m \mid 22$ donc $m = 2$. Par conséquent : $E(\mathbb{F}_{23}) \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/12\mathbb{Z}$.

Structure du groupe $E(\mathbb{F}_q)$

Théorème

Le groupe $E(\mathbb{F}_q)$ est soit un groupe cyclique soit un produit de deux groupes cycliques. Plus précisément, on a dans le premier cas :

$$E(\mathbb{F}_q) \cong \mathbb{Z}/n\mathbb{Z} \quad \text{où } n = \#(E(\mathbb{F}_q)),$$

et dans le second cas :

$$E(\mathbb{F}_q) \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \quad \text{où } m \mid n \text{ et } m \mid q-1.$$

Si on reprend les exemples précédents :

- $q = 23$, $E : y^2 = x^3 + x + 4$, $\#(E(\mathbb{F}_{23})) = 29$. L'entier 29 est premier donc $E(\mathbb{F}_{23})$ est cyclique et isomorphe à $\mathbb{Z}/29\mathbb{Z}$.
- $q = 23$, $E : y^2 = x^3 + 5x$, $\#(E(\mathbb{F}_{23})) = 24$. $E(\mathbb{F}_{23})$ n'est pas cyclique. On a $m \mid 24$ et $m \mid 22$ donc $m = 2$. Par conséquent : $E(\mathbb{F}_{23}) \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/12\mathbb{Z}$.

Structure du groupe $E(\mathbb{F}_q)$

Théorème

Le groupe $E(\mathbb{F}_q)$ est soit un groupe cyclique soit un produit de deux groupes cycliques. Plus précisément, on a dans le premier cas :

$$E(\mathbb{F}_q) \cong \mathbb{Z}/n\mathbb{Z} \quad \text{où } n = \#(E(\mathbb{F}_q)),$$

et dans le second cas :

$$E(\mathbb{F}_q) \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \quad \text{où } m \mid n \text{ et } m \mid q-1.$$

Si on reprend les exemples précédents :

- $q = 23$, $E : y^2 = x^3 + x + 4$, $\#(E(\mathbb{F}_{23})) = 29$. L'entier 29 est premier donc $E(\mathbb{F}_{23})$ est cyclique et isomorphe à $\mathbb{Z}/29\mathbb{Z}$.
- $q = 23$, $E : y^2 = x^3 + 5x$, $\#(E(\mathbb{F}_{23})) = 24$. $E(\mathbb{F}_{23})$ n'est pas cyclique. On a $m \mid 24$ et $m \mid 22$ donc $m = 2$. Par conséquent : $E(\mathbb{F}_{23}) \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/12\mathbb{Z}$.

Robustesse du PLDCE

- Dans le cas général, les meilleurs algorithmes de résolution du PLDCE (méthode ρ de *Pollard*, *baby-step giant-step*), connus à ce jour sont en temps en $\mathcal{O}(\sqrt{n})$ où n est, comme précédemment, l'ordre du sous-groupe de $E(\mathbb{F}_q)$.
- Donc, si n est suffisamment grand, le PLDCE n'est pas soluble en un temps raisonnable.
- Attention : il y a des cas particuliers pour lesquels il existe un algorithme (beaucoup) plus performant. Par exemple :
  - Si $\#(E(\mathbb{F}_q)) = q$, alors il existe un algorithme en $\mathcal{O}(\log n)$.
  - Si $q = 2^k$ et k composé, il y a la méthode de descente infinie de *Weil* étendue par *Galbraith*, *Hesse* et *Smart*.
- Si on envisage de chiffrer avec AES-128 par exemple, il faudra, si l'on veut une robustesse équivalente pour l'échange de clés, choisir $q \approx 2^{256}$. C'est à comparer avec RSA qui nécessiterait lui, pour le module n et la clé privée, une taille d'environ 3072 bits.

Robustesse du PLDCE

- Dans le cas général, les meilleurs algorithmes de résolution du PLDCE (méthode ρ de *Pollard*, *baby-step giant-step*), connus à ce jour sont en temps en $\mathcal{O}(\sqrt{n})$ où n est, comme précédemment, l'ordre du sous-groupe de $E(\mathbb{F}_q)$.
- Donc, si n est suffisamment grand, le PLDCE n'est pas soluble en un temps raisonnable.
- Attention : il y a des cas particuliers pour lesquels il existe un algorithme (beaucoup) plus performant. Par exemple :
  - Si $\#(E(\mathbb{F}_q)) = q$, alors il existe un algorithme en $\mathcal{O}(\log n)$.
  - Si $q = 2^k$ et k composé, il y a la méthode de descente infinie de *Weil* étendue par *Galbraith*, *Hesse* et *Smart*.
- Si on envisage de chiffrer avec AES-128 par exemple, il faudra, si l'on veut une robustesse équivalente pour l'échange de clés, choisir $q \approx 2^{256}$. C'est à comparer avec RSA qui nécessiterait lui, pour le module n et la clé privée, une taille d'environ 3072 bits.

Robustesse du PLDCE

- Dans le cas général, les meilleurs algorithmes de résolution du PLDCE (méthode ρ de *Pollard*, *baby-step giant-step*), connus à ce jour sont en temps en $\mathcal{O}(\sqrt{n})$ où n est, comme précédemment, l'ordre du sous-groupe de $E(\mathbb{F}_q)$.
- Donc, si n est suffisamment grand, le PLDCE n'est pas soluble en un temps raisonnable.
- Attention : il y a des cas particuliers pour lesquels il existe un algorithme (beaucoup) plus performant. Par exemple :
  - Si $\#(E(\mathbb{F}_q)) = q$, alors il existe un algorithme en $\mathcal{O}(\log n)$.
  - Si $q = 2^k$ et k composé, il y a la méthode de descente infinie de *Weil* étendue par *Galbraith*, *Hesse* et *Smart*.
- Si on envisage de chiffrer avec AES-128 par exemple, il faudra, si l'on veut une robustesse équivalente pour l'échange de clés, choisir $q \approx 2^{256}$. C'est à comparer avec RSA qui nécessiterait lui, pour le module n et la clé privée, une taille d'environ 3072 bits.

Robustesse du PLDCE

- Dans le cas général, les meilleurs algorithmes de résolution du PLDCE (méthode ρ de *Pollard*, *baby-step giant-step*), connus à ce jour sont en temps en $\mathcal{O}(\sqrt{n})$ où n est, comme précédemment, l'ordre du sous-groupe de $E(\mathbb{F}_q)$.
- Donc, si n est suffisamment grand, le PLDCE n'est pas soluble en un temps raisonnable.
- Attention : il y a des cas particuliers pour lesquels il existe un algorithme (beaucoup) plus performant. Par exemple :
  - Si $\#(E(\mathbb{F}_q)) = q$, alors il existe un algorithme en $\mathcal{O}(\log n)$.
  - Si $q = 2^k$ et k composé, il y a la méthode de descente infinie de *Weil* étendue par *Galbraith*, *Hesse* et *Smart*.
- Si on envisage de chiffrer avec AES-128 par exemple, il faudra, si l'on veut une robustesse équivalente pour l'échange de clés, choisir $q \approx 2^{256}$. C'est à comparer avec RSA qui nécessiterait lui, pour le module n et la clé privée, une taille d'environ 3072 bits.

Robustesse du PLDCE

- Dans le cas général, les meilleurs algorithmes de résolution du PLDCE (méthode ρ de *Pollard*, *baby-step giant-step*), connus à ce jour sont en temps en $\mathcal{O}(\sqrt{n})$ où n est, comme précédemment, l'ordre du sous-groupe de $E(\mathbb{F}_q)$.
- Donc, si n est suffisamment grand, le PLDCE n'est pas soluble en un temps raisonnable.
- Attention : il y a des cas particuliers pour lesquels il existe un algorithme (beaucoup) plus performant. Par exemple :
  - Si $\#(E(\mathbb{F}_q)) = q$, alors il existe un algorithme en $\mathcal{O}(\log n)$.
  - Si $q = 2^k$ et k composé, il y a la méthode de descente infinie de *Weil* étendue par *Galbraith*, *Hesse* et *Smart*.
- Si on envisage de chiffrer avec AES-128 par exemple, il faudra, si l'on veut une robustesse équivalente pour l'échange de clés, choisir $q \approx 2^{256}$. C'est à comparer avec RSA qui nécessiterait lui, pour le module n et la clé privée, une taille d'environ 3072 bits.

Courbes de *Koblitz*

- En cryptographie pratique, on utilisera plus volontiers les corps finis $\mathbb{F}_{2^k}$ pour $k > 0$. Or, en caractéristique 2, toutes les courbes $E : y^2 = x^3 + ax + b$ sont singulières. Il faut donc revenir à l'équation non réduite : $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$.
- Une courbe de *Koblitz* est une courbe particulière d'équation :

$$y^2 + xy = x^3 + ax^2 + 1 \quad \text{avec } a = 0 \text{ ou } a = 1.$$

- Un intérêt de ces courbes est que le calcul du nombre de points est donné explicitement par la formule :

$$\#(E(\mathbb{F}_{2^k})) = 2^k - \left(\frac{-1+\sqrt{-7}}{2}\right)^k - \left(\frac{-1-\sqrt{-7}}{2}\right)^k + 1$$

Courbes de *Koblitz*

- En cryptographie pratique, on utilisera plus volontiers les corps finis $\mathbb{F}_{2^k}$ pour $k > 0$. Or, en caractéristique 2, toutes les courbes $E : y^2 = x^3 + ax + b$ sont singulières. Il faut donc revenir à l'équation non réduite : $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$.
- Une courbe de *Koblitz* est une courbe particulière d'équation :

$$y^2 + xy = x^3 + ax^2 + 1 \quad \text{avec } a = 0 \text{ ou } a = 1.$$

- Un intérêt de ces courbes est que le calcul du nombre de points est donné explicitement par la formule :

$$\#(E(\mathbb{F}_{2^k})) = 2^k - \left(\frac{-1+\sqrt{-7}}{2}\right)^k - \left(\frac{-1-\sqrt{-7}}{2}\right)^k + 1$$

Courbes de *Koblitz*

- En cryptographie pratique, on utilisera plus volontiers les corps finis $\mathbb{F}_{2^k}$ pour $k > 0$. Or, en caractéristique 2, toutes les courbes $E : y^2 = x^3 + ax + b$ sont singulières. Il faut donc revenir à l'équation non réduite : $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$.
- Une courbe de *Koblitz* est une courbe particulière d'équation :

$$y^2 + xy = x^3 + ax^2 + 1 \quad \text{avec } a = 0 \text{ ou } a = 1.$$

- Un intérêt de ces courbes est que le calcul du nombre de points est donné explicitement par la formule :

$$\#(E(\mathbb{F}_{2^k})) = 2^k - \left(\frac{-1+\sqrt{-7}}{2}\right)^k - \left(\frac{-1-\sqrt{-7}}{2}\right)^k + 1$$

Courbes de *Koblitz*

- Un autre intérêt calculatoire des courbes de *Koblitz* est l'existence de l'homomorphisme de groupes $\tau : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$, $\tau(x, y) = (x^2, y^2)$.
- L'application τ a donc la propriété : $\tau(P + Q) = \tau(P) + \tau(Q)$.
- Elle satisfait également l'identité : $\tau^2(P) + \tau(P) + 1 = O$.
- Donc si on écrit :

$$\alpha = \alpha_0 + \alpha_1\tau + \alpha_2\tau^2 + \cdots + \alpha_r\tau^r \quad \text{avec } \alpha_0, \ldots, \alpha_r \in \{0, \pm 1\},$$

on a :

$$\alpha P = \alpha_0 P + \alpha_1\tau(P) + \alpha_2\tau^2(P) + \cdots + \alpha_r\tau^r(P).$$

Le calcul de αP s'effectue ainsi **sans doublement**.

Courbes de *Koblitz*

- Un autre intérêt calculatoire des courbes de *Koblitz* est l'existence de l'homomorphisme de groupes $\tau : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$, $\tau(x, y) = (x^2, y^2)$.
- L'application τ a donc la propriété : $\tau(P + Q) = \tau(P) + \tau(Q)$.
- Elle satisfait également l'identité : $\tau^2(P) + \tau(P) + 1 = O$.
- Donc si on écrit :

$$\alpha = \alpha_0 + \alpha_1\tau + \alpha_2\tau^2 + \cdots + \alpha_r\tau^r \quad \text{avec } \alpha_0, \ldots, \alpha_r \in \{0, \pm 1\},$$

on a :

$$\alpha P = \alpha_0 P + \alpha_1\tau(P) + \alpha_2\tau^2(P) + \cdots + \alpha_r\tau^r(P).$$

Le calcul de αP s'effectue ainsi **sans doublement**.

Courbes de *Koblitz*

- Un autre intérêt calculatoire des courbes de *Koblitz* est l'existence de l'homomorphisme de groupes $\tau : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$, $\tau(x, y) = (x^2, y^2)$.
- L'application τ a donc la propriété : $\tau(P + Q) = \tau(P) + \tau(Q)$.
- Elle satisfait également l'identité : $\tau^2(P) + \tau(P) + 1 = O$.
- Donc si on écrit :

$$\alpha = \alpha_0 + \alpha_1\tau + \alpha_2\tau^2 + \cdots + \alpha_r\tau^r \quad \text{avec } \alpha_0, \ldots, \alpha_r \in \{0, \pm 1\},$$

on a :

$$\alpha P = \alpha_0 P + \alpha_1\tau(P) + \alpha_2\tau^2(P) + \cdots + \alpha_r\tau^r(P).$$

Le calcul de αP s'effectue ainsi **sans doublement**.

Courbes de *Koblitz*

- Un autre intérêt calculatoire des courbes de *Koblitz* est l'existence de l'homomorphisme de groupes $\tau : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$, $\tau(x, y) = (x^2, y^2)$.
- L'application τ a donc la propriété : $\tau(P + Q) = \tau(P) + \tau(Q)$.
- Elle satisfait également l'identité : $\tau^2(P) + \tau(P) + 1 = O$.
- Donc si on écrit :

$$\alpha = \alpha_0 + \alpha_1\tau + \alpha_2\tau^2 + \cdots + \alpha_r\tau^r \quad \text{avec } \alpha_0, \ldots, \alpha_r \in \{0, \pm 1\},$$

on a :

$$\alpha P = \alpha_0 P + \alpha_1\tau(P) + \alpha_2\tau^2(P) + \cdots + \alpha_r\tau^r(P).$$

Le calcul de αP s'effectue ainsi **sans doublement**.

Well-Known Group 3 - RFC 2412 (1998)

- Corps $k = \mathbb{F}_{2^{155}}$, polynôme irréductible : $X^{155} + X^{62} + 1$.
- Courbe elliptique $E : y^2 + xy = x^3 + ax^2 + b$ avec $a = 0$ et $b = X^{18} + X^{17} + X^{16} + X^{13} + X^{12} + X^9 + X^8 + X^7 + X^3 + X^2 + X + 1$. En binaire, $b = 1110011001110001111$.
- Le générateur est $P(x, y)$ avec $x = X^6 + X^5 + X^4 + X^3 + X + 1$ et $y = X^8 + X^7 + X^6 + X^3$. En binaire, $x = 1111011$ et $y = 111001000$.
- $\#(E(\mathbb{F}_{2^{155}})) = 12p$ où p est l'entier premier :

$$p = 3805993847215893016155463826195386266397436443$$

- L'ordre du sous-groupe engendré par $P(x, y)$ est égal à $4p$.
- On notera que $155 = 5 \times 31$ n'est pas premier.

Well-Known Group 3 - RFC 2412 (1998)

- Corps $k = \mathbb{F}_{2^{155}}$, polynôme irréductible : $X^{155} + X^{62} + 1$.
- Courbe elliptique $E : y^2 + xy = x^3 + ax^2 + b$ avec $a = 0$ et $b = X^{18} + X^{17} + X^{16} + X^{13} + X^{12} + X^9 + X^8 + X^7 + X^3 + X^2 + X + 1$. En binaire, $b = 1110011001110001111$.
- Le générateur est $P(x, y)$ avec $x = X^6 + X^5 + X^4 + X^3 + X + 1$ et $y = X^8 + X^7 + X^6 + X^3$. En binaire, $x = 1111011$ et $y = 111001000$.
- $\#(E(\mathbb{F}_{2^{155}})) = 12p$ où p est l'entier premier :

$$p = 3805993847215893016155463826195386266397436443$$

- L'ordre du sous-groupe engendré par $P(x, y)$ est égal à $4p$.
- On notera que $155 = 5 \times 31$ n'est pas premier.

Well-Known Group 3 - RFC 2412 (1998)

- Corps $k = \mathbb{F}_{2^{155}}$, polynôme irréductible : $X^{155} + X^{62} + 1$.
- Courbe elliptique $E : y^2 + xy = x^3 + ax^2 + b$ avec $a = 0$ et $b = X^{18} + X^{17} + X^{16} + X^{13} + X^{12} + X^9 + X^8 + X^7 + X^3 + X^2 + X + 1$. En binaire, $b = 1110011001110001111$.
- Le générateur est $P(x, y)$ avec $x = X^6 + X^5 + X^4 + X^3 + X + 1$ et $y = X^8 + X^7 + X^6 + X^3$. En binaire, $x = 1111011$ et $y = 111001000$.
- $\#(E(\mathbb{F}_{2^{155}})) = 12p$ où p est l'entier premier :

$$p = 3805993847215893016155463826195386266397436443$$

- L'ordre du sous-groupe engendré par $P(x, y)$ est égal à $4p$.
- On notera que $155 = 5 \times 31$ n'est pas premier.

Well-Known Group 3 - RFC 2412 (1998)

- Corps $k = \mathbb{F}_{2^{155}}$, polynôme irréductible : $X^{155} + X^{62} + 1$.
- Courbe elliptique $E : y^2 + xy = x^3 + ax^2 + b$ avec $a = 0$ et $b = X^{18} + X^{17} + X^{16} + X^{13} + X^{12} + X^9 + X^8 + X^7 + X^3 + X^2 + X + 1$. En binaire, $b = 1110011001110001111$.
- Le générateur est $P(x, y)$ avec $x = X^6 + X^5 + X^4 + X^3 + X + 1$ et $y = X^8 + X^7 + X^6 + X^3$. En binaire, $x = 1111011$ et $y = 111001000$.
- $\#(E(\mathbb{F}_{2^{155}})) = 12p$ où p est l'entier premier :

$$p = 3805993847215893016155463826195386266397436443$$

- L'ordre du sous-groupe engendré par $P(x, y)$ est égal à $4p$.
- On notera que $155 = 5 \times 31$ n'est pas premier.

Well-Known Group 3 - RFC 2412 (1998)

- Corps $k = \mathbb{F}_{2^{155}}$, polynôme irréductible : $X^{155} + X^{62} + 1$.
- Courbe elliptique $E : y^2 + xy = x^3 + ax^2 + b$ avec $a = 0$ et $b = X^{18} + X^{17} + X^{16} + X^{13} + X^{12} + X^9 + X^8 + X^7 + X^3 + X^2 + X + 1$. En binaire, $b = 1110011001110001111$.
- Le générateur est $P(x, y)$ avec $x = X^6 + X^5 + X^4 + X^3 + X + 1$ et $y = X^8 + X^7 + X^6 + X^3$. En binaire, $x = 1111011$ et $y = 111001000$.
- $\#(E(\mathbb{F}_{2^{155}})) = 12p$ où p est l'entier premier :

$$p = 3805993847215893016155463826195386266397436443$$

- L'ordre du sous-groupe engendré par $P(x, y)$ est égal à $4p$.
- On notera que $155 = 5 \times 31$ n'est pas premier.

Well-Known Group 3 - RFC 2412 (1998)

- Corps $k = \mathbb{F}_{2^{155}}$, polynôme irréductible : $X^{155} + X^{62} + 1$.
- Courbe elliptique $E : y^2 + xy = x^3 + ax^2 + b$ avec $a = 0$ et $b = X^{18} + X^{17} + X^{16} + X^{13} + X^{12} + X^9 + X^8 + X^7 + X^3 + X^2 + X + 1$. En binaire, $b = 1110011001110001111$.
- Le générateur est $P(x, y)$ avec $x = X^6 + X^5 + X^4 + X^3 + X + 1$ et $y = X^8 + X^7 + X^6 + X^3$. En binaire, $x = 1111011$ et $y = 111001000$.
- $\#(E(\mathbb{F}_{2^{155}})) = 12p$ où p est l'entier premier :

$$p = 3805993847215893016155463826195386266397436443$$

- L'ordre du sous-groupe engendré par $P(x, y)$ est égal à $4p$.
- On notera que $155 = 5 \times 31$ n'est pas premier.

Well-Known Group 25 - RFC 5114 (2008)

- Corps $k = \mathbb{F}_p$ avec $p = 2^{192} - 2^{64} - 1$ (p est premier).
- Courbe elliptique $E : y^2 = x^3 + ax + b$ avec :
  $a =$ FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC,
  $b =$ 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1.
- Le générateur est $P(x, y)$ avec :
  $x =$ 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012,
  $y =$ 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811.
- L'ordre du sous-groupe engendré par $P(x, y)$ est :
  $n =$ FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831.

Well-Known Group 25 - RFC 5114 (2008)

- Corps $k = \mathbb{F}_p$ avec $p = 2^{192} - 2^{64} - 1$ (p est premier).
- Courbe elliptique $E : y^2 = x^3 + ax + b$ avec :
  $a =$ FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC,
  $b =$ 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1.
- Le générateur est $P(x, y)$ avec :
  $x =$ 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012,
  $y =$ 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811.
- L'ordre du sous-groupe engendré par $P(x, y)$ est :
  $n =$ FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831.

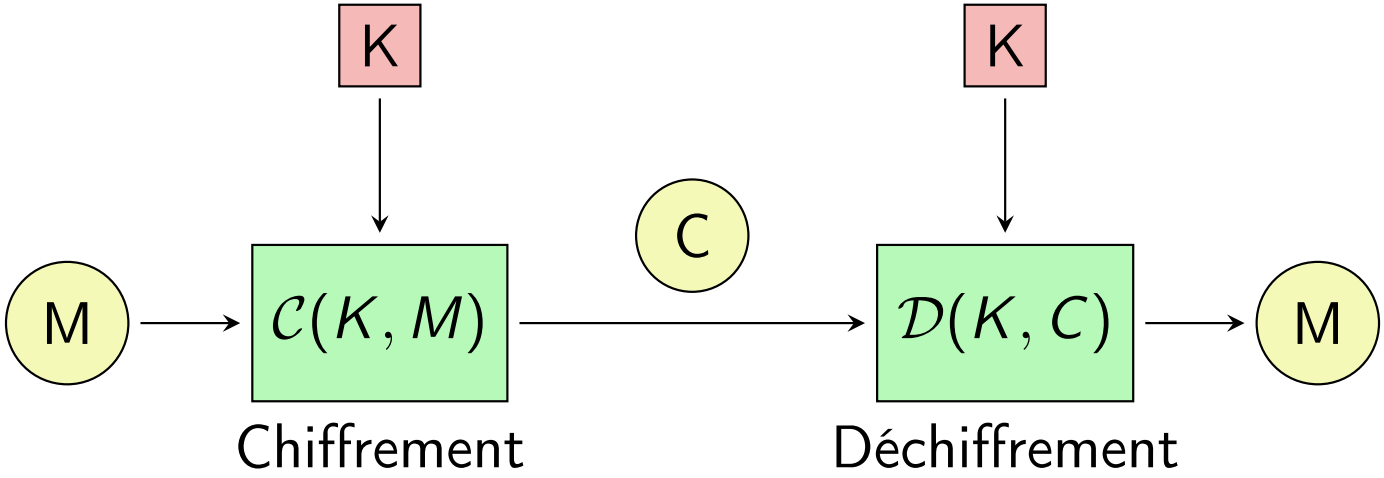
Well-Known Group 25 - RFC 5114 (2008)

- Corps $k = \mathbb{F}_p$ avec $p = 2^{192} - 2^{64} - 1$ (p est premier).
- Courbe elliptique $E : y^2 = x^3 + ax + b$ avec :
  $a =$ FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC,
  $b =$ 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1.
- Le générateur est $P(x, y)$ avec :
  $x =$ 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012,
  $y =$ 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811.
- L'ordre du sous-groupe engendré par $P(x, y)$ est :
  $n =$ FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831.

Well-Known Group 25 - RFC 5114 (2008)

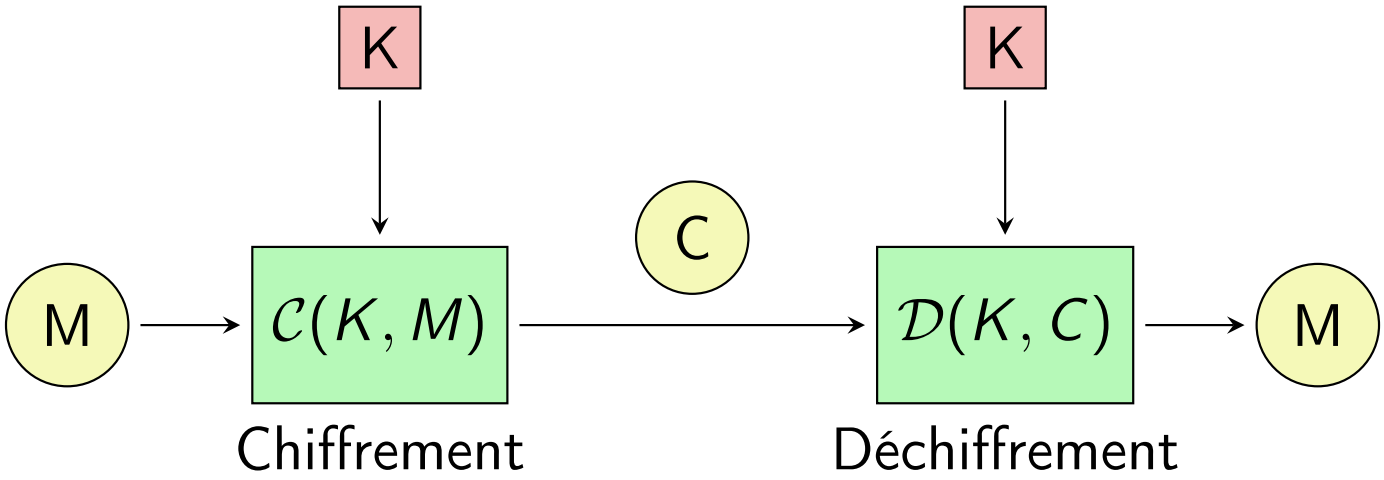
- Corps $k = \mathbb{F}_p$ avec $p = 2^{192} - 2^{64} - 1$ (p est premier).
- Courbe elliptique $E : y^2 = x^3 + ax + b$ avec :
  $a =$ FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC,
  $b =$ 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1.
- Le générateur est $P(x, y)$ avec :
  $x =$ 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012,
  $y =$ 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811.
- L'ordre du sous-groupe engendré par $P(x, y)$ est :
  $n =$ FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831.

Vue d'ensemble et caractéristiques



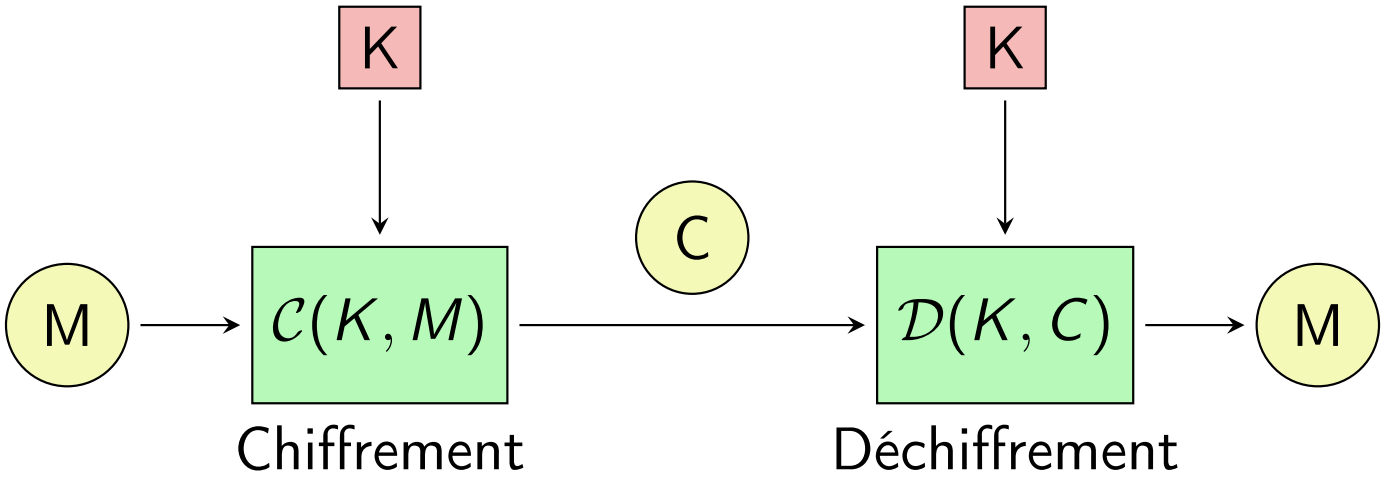
- Chiffrement rapide, voire très rapide en implantation matérielle.
- Clés plutôt courtes : 128 - 256 bits (à comparer avec RSA : 1024 - 2048 bits).
- Inconvénient majeur : partage d'un secret (la clé commune K), ce qui est toujours délicat à gérer.

Vue d'ensemble et caractéristiques



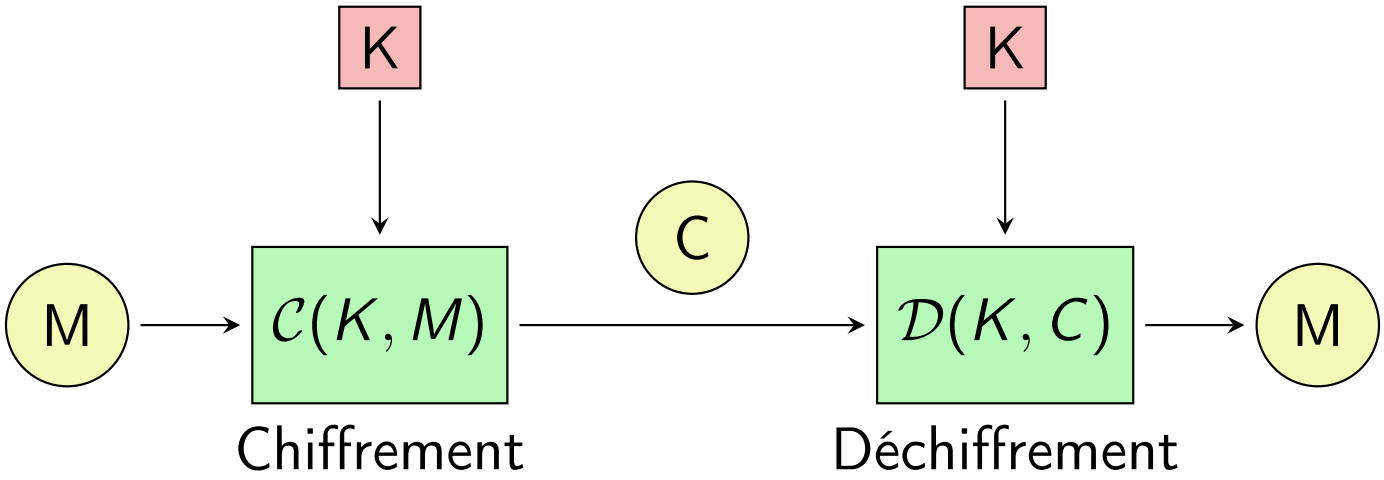
- Chiffrement rapide, voire très rapide en implantation matérielle.
- Clés plutôt courtes : 128 - 256 bits (à comparer avec RSA : 1024 - 2048 bits).
- Inconvénient majeur : partage d'un secret (la clé commune K), ce qui est toujours délicat à gérer.

Vue d'ensemble et caractéristiques



- Chiffrement rapide, voire très rapide en implantation matérielle.
- Clés plutôt courtes : 128 - 256 bits (à comparer avec RSA : 1024 - 2048 bits).
- Inconvénient majeur : partage d'un secret (la clé commune K), ce qui est toujours délicat à gérer.

Vue d'ensemble et caractéristiques



- Chiffrement rapide, voire très rapide en implantation matérielle.
- Clés plutôt courtes : 128 - 256 bits (à comparer avec RSA : 1024 - 2048 bits).
- Inconvénient majeur : partage d'un secret (la clé commune K), ce qui est toujours délicat à gérer.

Théorie de Shannon

- En 1949, *Claude Shannon*, dans son fameux article fondateur de la cryptographie moderne (*Communication Theory of Secrecy Systems*), introduit deux propriétès que devrait satisfaire un bon algorithme de chiffrement : la **diffusion** et la **confusion**.
- La propriété de *diffusion* signifie que des changements **minimes** dans les données en entrée se traduisent par des changements **importants** dans les données en sortie.
- La propriété de *confusion* mesure la **complexité** de l'interdépendance entre la *clé*, le *clair* et le *chiffré*. Plus cette complexité est grande, meilleur est l'algorithme.
- En pratique, la diffusion est un processus essentiellement **linéaire** alors qu'au contraire la confusion s'appuie sur des opérateurs **non-linéaires** comme les *S-Boxes*.

Théorie de Shannon

- En 1949, *Claude Shannon*, dans son fameux article fondateur de la cryptographie moderne (*Communication Theory of Secrecy Systems*), introduit deux propriétès que devrait satisfaire un bon algorithme de chiffrement : la **diffusion** et la **confusion**.
- La propriété de *diffusion* signifie que des changements **minimes** dans les données en entrée se traduisent par des changements **importants** dans les données en sortie.
- La propriété de *confusion* mesure la **complexité** de l'interdépendance entre la *clé*, le *clair* et le *chiffré*. Plus cette complexité est grande, meilleur est l'algorithme.
- En pratique, la diffusion est un processus essentiellement **linéaire** alors qu'au contraire la confusion s'appuie sur des opérateurs **non-linéaires** comme les *S-Boxes*.

Théorie de Shannon

- En 1949, *Claude Shannon*, dans son fameux article fondateur de la cryptographie moderne (*Communication Theory of Secrecy Systems*), introduit deux propriétès que devrait satisfaire un bon algorithme de chiffrement : la **diffusion** et la **confusion**.
- La propriété de *diffusion* signifie que des changements **minimes** dans les données en entrée se traduisent par des changements **importants** dans les données en sortie.
- La propriété de *confusion* mesure la **complexité** de l'interdépendance entre la *clé*, le *clair* et le *chiffré*. Plus cette complexité est grande, meilleur est l'algorithme.
- En pratique, la diffusion est un processus essentiellement **linéaire** alors qu'au contraire la confusion s'appuie sur des opérateurs **non-linéaires** comme les *S-Boxes*.

Théorie de Shannon

- En 1949, *Claude Shannon*, dans son fameux article fondateur de la cryptographie moderne (*Communication Theory of Secrecy Systems*), introduit deux propriétès que devrait satisfaire un bon algorithme de chiffrement : la **diffusion** et la **confusion**.
- La propriété de *diffusion* signifie que des changements **minimes** dans les données en entrée se traduisent par des changements **importants** dans les données en sortie.
- La propriété de *confusion* mesure la **complexité** de l'interdépendance entre la *clé*, le *clair* et le *chiffré*. Plus cette complexité est grande, meilleur est l'algorithme.
- En pratique, la diffusion est un processus essentiellement **linéaire** alors qu'au contraire la confusion s'appuie sur des opérateurs **non-linéaires** comme les *S-Boxes*.

Les deux types de chiffrements symétriques

Il existe deux grandes familles de chiffrements symétriques :

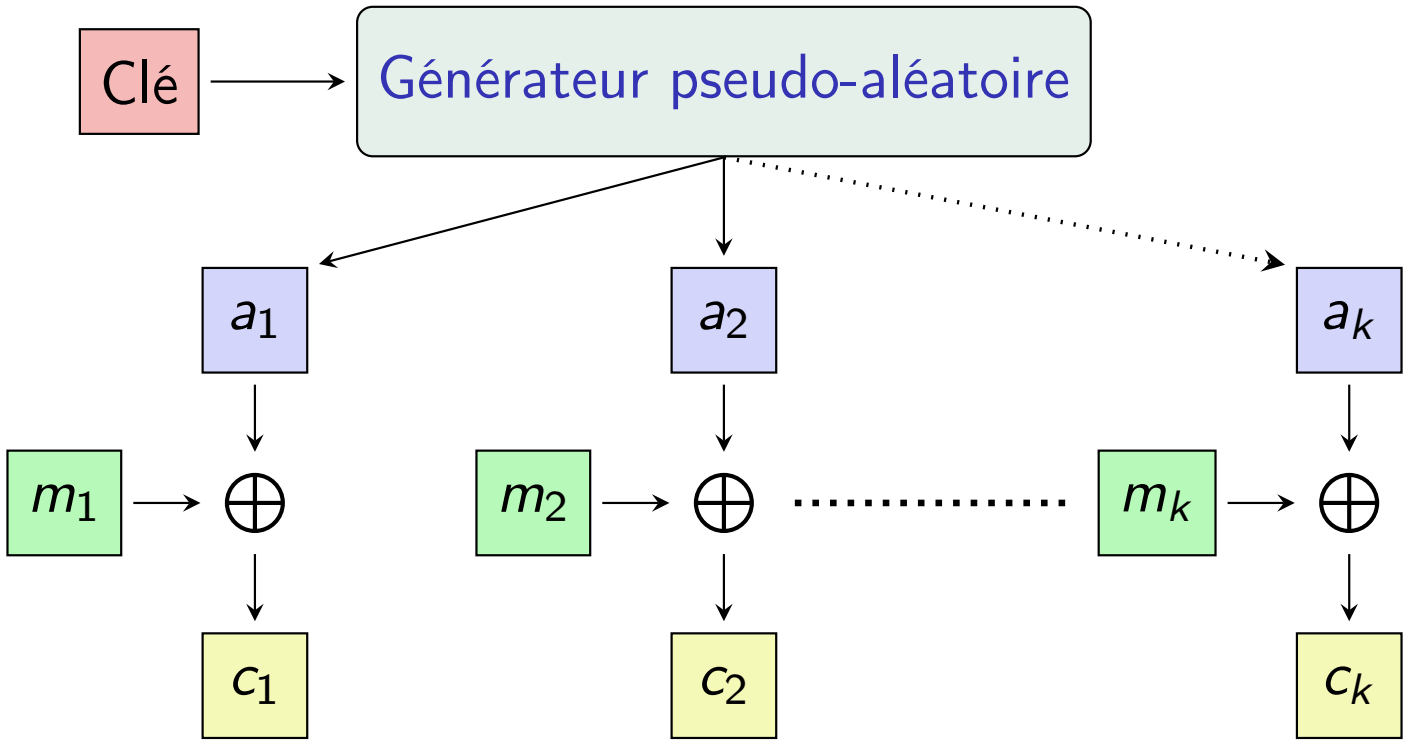
- Les **chiffrements à flot** - on génère à partir de la clé une suite chiffrante pseudo-aléatoire de même longueur que les données. On combine cette suite, par exemple avec un XOR bit-à-bit, avec les données à chiffer.
  - RC4 (Rivest, 1987) : SSL/TLS, WEP, WPA, WPA2,...
  - E0 : Bluetooth
  - A5 : GSM
- Les **chiffrements par bloc** - les données à chiffrer sont découpées en *blocs* de taille fixe (typiquement 64 ou 128 bits). Les blocs sont chiffrés séparément et ensuite combinés selon un *mode opératoire* (ECB, CBC, CTR...).
  - DES (IBM et NSA, 1975) - blocs 64 bits, clés 56 bits
  - IDEA (Lai-Massey, 1992) - blocs 64 bits, clés 128 bits
  - AES/Rijndael (Daemen-Rijmen, 1998) - blocs/clés 128, 192, 256 bits
  - et aussi : RC5, RC6, Camellia,...

Les deux types de chiffrements symétriques

Il existe deux grandes familles de chiffrements symétriques :

- Les **chiffrements à flot** - on génère à partir de la clé une suite chiffrante pseudo-aléatoire de même longueur que les données. On combine cette suite, par exemple avec un XOR bit-à-bit, avec les données à chiffer.
  - RC4 (Rivest, 1987) : SSL/TLS, WEP, WPA, WPA2,...
  - E0 : Bluetooth
  - A5 : GSM
- Les **chiffrements par bloc** - les données à chiffrer sont découpées en *blocs* de taille fixe (typiquement 64 ou 128 bits). Les blocs sont chiffrés séparément et ensuite combinés selon un *mode opératoire* (ECB, CBC, CTR...).
  - DES (IBM et NSA, 1975) - blocs 64 bits, clés 56 bits
  - IDEA (Lai-Massey, 1992) - blocs 64 bits, clés 128 bits
  - AES/Rijndael (Daemen-Rijmen, 1998) - blocs/clés 128, 192, 256 bits
  - et aussi : RC5, RC6, Camellia,...

Schéma de fonctionnement



- Des données pseudo-aléatoires, appelées **flux de clé** (*keystream*), sont générées et combinées (le plus souvent avec un XOR) aux données en clair pour produire les données chiffrées.

Caractéristiques

- Les chiffrements à flot sont **très rapides**, les implantations matérielles étant particulièrement efficaces.
- On chiffre à la volée sans attendre d'avoir lu, tout ou une partie, des données : bien adapté aux applications **temps réel**.
- Ils sont très utilisés pour la protection des données multimedia.
- Ce chiffrement est adapté du chiffrement de *Vernam* (théoriquement inviolable) sauf qu'ici on génère une suite **pseudo-aléatoire** à partir d'une clé **aléatoire**, généralement de petite taille comparée à la taille des données. Le chiffrement de *Vernam*, quant à lui, utilise une clé aléatoire **à usage unique** de même taille que les données à chiffrer.
- Le chiffrement de *Vernam* est certes sûr mais, en pratique, très difficile à mettre en œuvre. Le chiffrement à flot peut être vu comme un "pseudo-Vernam" adapté à un usage concret.

Caractéristiques

- Les chiffrements à flot sont **très rapides**, les implantations matérielles étant particulièrement efficaces.
- On chiffre à la volée sans attendre d'avoir lu, tout ou une partie, des données : bien adapté aux applications **temps réel**.
- Ils sont très utilisés pour la protection des données multimedia.
- Ce chiffrement est adapté du chiffrement de *Vernam* (théoriquement inviolable) sauf qu'ici on génère une suite **pseudo-aléatoire** à partir d'une clé **aléatoire**, généralement de petite taille comparée à la taille des données. Le chiffrement de *Vernam*, quant à lui, utilise une clé aléatoire **à usage unique** de même taille que les données à chiffrer.
- Le chiffrement de *Vernam* est certes sûr mais, en pratique, très difficile à mettre en œuvre. Le chiffrement à flot peut être vu comme un "pseudo-Vernam" adapté à un usage concret.

Caractéristiques

- Les chiffrements à flot sont **très rapides**, les implantations matérielles étant particulièrement efficaces.
- On chiffre à la volée sans attendre d'avoir lu, tout ou une partie, des données : bien adapté aux applications **temps réel**.
- Ils sont très utilisés pour la protection des données multimedia.
- Ce chiffrement est adapté du chiffrement de *Vernam* (théoriquement inviolable) sauf qu'ici on génère une suite **pseudo-aléatoire** à partir d'une clé **aléatoire**, généralement de petite taille comparée à la taille des données. Le chiffrement de *Vernam*, quant à lui, utilise une clé aléatoire **à usage unique** de même taille que les données à chiffrer.
- Le chiffrement de *Vernam* est certes sûr mais, en pratique, très difficile à mettre en œuvre. Le chiffrement à flot peut être vu comme un "pseudo-Vernam" adapté à un usage concret.

Caractéristiques

- Les chiffrements à flot sont **très rapides**, les implantations matérielles étant particulièrement efficaces.
- On chiffre à la volée sans attendre d'avoir lu, tout ou une partie, des données : bien adapté aux applications **temps réel**.
- Ils sont très utilisés pour la protection des données multimedia.
- Ce chiffrement est adapté du chiffrement de *Vernam* (théoriquement inviolable) sauf qu'ici on génère une suite **pseudo-aléatoire** à partir d'une clé **aléatoire**, généralement de petite taille comparée à la taille des données. Le chiffrement de *Vernam*, quant à lui, utilise une clé aléatoire **à usage unique** de même taille que les données à chiffrer.
- Le chiffrement de *Vernam* est certes sûr mais, en pratique, très difficile à mettre en œuvre. Le chiffrement à flot peut être vu comme un "pseudo-Vernam" adapté à un usage concret.

Caractéristiques

- Les chiffrements à flot sont **très rapides**, les implantations matérielles étant particulièrement efficaces.
- On chiffre à la volée sans attendre d'avoir lu, tout ou une partie, des données : bien adapté aux applications **temps réel**.
- Ils sont très utilisés pour la protection des données multimedia.
- Ce chiffrement est adapté du chiffrement de *Vernam* (théoriquement inviolable) sauf qu'ici on génère une suite **pseudo-aléatoire** à partir d'une clé **aléatoire**, généralement de petite taille comparée à la taille des données. Le chiffrement de *Vernam*, quant à lui, utilise une clé aléatoire **à usage unique** de même taille que les données à chiffrer.
- Le chiffrement de *Vernam* est certes sûr mais, en pratique, très difficile à mettre en œuvre. Le chiffrement à flot peut être vu comme un "pseudo-Vernam" adapté à un usage concret.

Sécurité

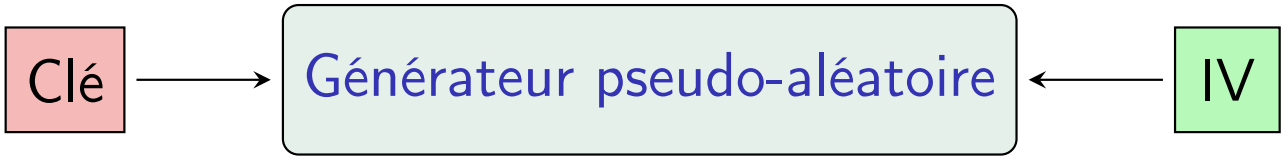
- Une première difficulté : si on chiffre plusieurs messages avec la **même clé**, on génère le **même aléa**. On ajoute donc une entrée auxiliaire, le vecteur d'initialisation (*Initialization Vector - IV*) :

- Le générateur pseudo-aléatoire doit être de bonne qualité. Idéalement :
  1. chaque bit de sortie vaut 0 ou 1 avec une probabilité $\frac{1}{2}$,
  2. aucun bit de sortie n'est corrélé avec les précédents ou les suivants,
  3. la période du générateur est astronomiquement longue.

  Problème : ces trois conditions sont extrêmement difficiles à satisfaire en pratique.
- En l'absence d'une assise théorique solide, la **sécurité** des chiffrements à flot reste **problématique**. Pour preuve de cet état de fait, le nombre important d'algorithmes proposés qui ont été cassés...

Sécurité

- Une première difficulté : si on chiffre plusieurs messages avec la **même clé**, on génère le **même aléa**. On ajoute donc une entrée auxiliaire, le vecteur d'initialisation (*Initialization Vector - IV*) :

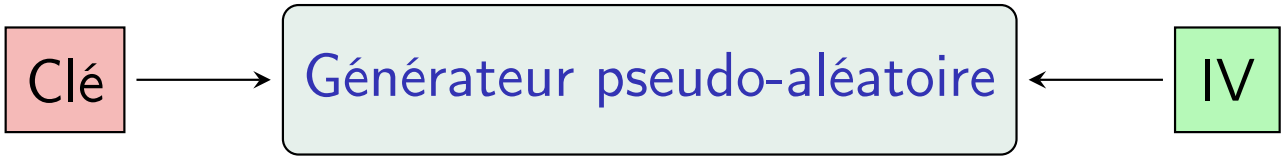


- Le générateur pseudo-aléatoire doit être de bonne qualité. Idéalement :
  1. chaque bit de sortie vaut 0 ou 1 avec une probabilité $\frac{1}{2}$,
  2. aucun bit de sortie n'est corrélé avec les précédents ou les suivants,
  3. la période du générateur est suffisamment longue.

  Problème : ces trois conditions sont extrêmement difficiles à satisfaire en pratique.
- En l'absence d'une assise théorique solide, la **sécurité** des chiffrements à flot reste **problématique**. Pour preuve de cet état de fait, le nombre important d'algorithmes proposés qui ont été cassés...

Sécurité

- Une première difficulté : si on chiffre plusieurs messages avec la **même clé**, on génère le **même aléa**. On ajoute donc une entrée auxiliaire, le vecteur d'initialisation (*Initialization Vector - IV*) :

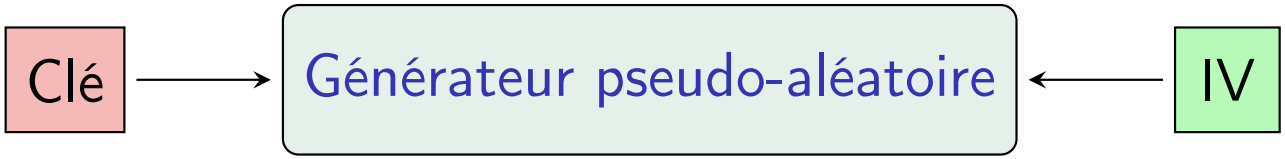


- Le générateur pseudo-aléatoire doit être de bonne qualité. Idéalement :
  1. chaque bit de sortie vaut 0 ou 1 avec une probabilité $\frac{1}{2}$,
  2. aucun bit de sortie n'est corrélé avec les précédents ou les suivants,
  3. la période du générateur est suffisamment longue.

  Problème : ces trois conditions sont extrêmement difficiles à satisfaire en pratique.
- En l'absence d'une assise théorique solide, la sécurité des chiffrements à flot reste **problématique**. Pour preuve de cet état de fait, le nombre important d'algorithmes proposés qui ont été cassés...

Sécurité

- Une première difficulté : si on chiffre plusieurs messages avec la **même clé**, on génère le **même aléa**. On ajoute donc une entrée auxiliaire, le vecteur d'initialisation (*Initialization Vector - IV*) :

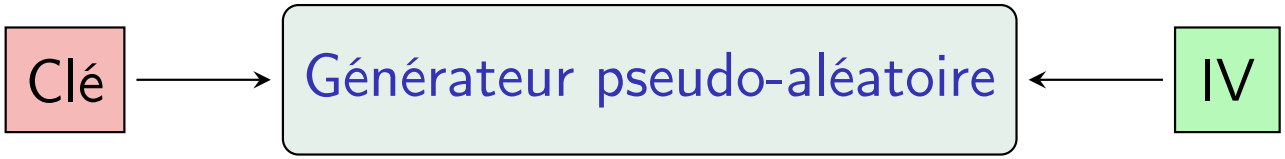


- Le générateur pseudo-aléatoire doit être de bonne qualité. Idéalement :
  1. chaque bit de sortie vaut 0 ou 1 avec une probabilité $\frac{1}{2}$,
  2. aucun bit de sortie n'est corrélé avec les précédents ou les suivants,
  3. la période du générateur est suffisamment longue.

  Problème : ces trois conditions sont extrêmement difficiles à satisfaire en pratique.
- En l'absence d'une assise théorique solide, la **sécurité** des chiffrements à flot reste **problématique**. Pour preuve de cet état de fait, le nombre important d'algorithmes proposés qui ont été cassés...

Sécurité

- Une première difficulté : si on chiffre plusieurs messages avec la **même clé**, on génère le **même aléa**. On ajoute donc une entrée auxiliaire, le vecteur d'initialisation (*Initialization Vector - IV*) :

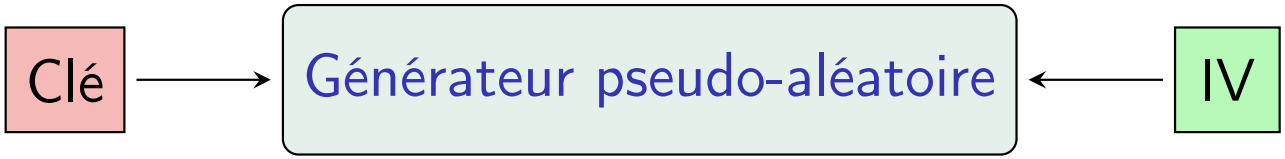


- Le générateur pseudo-aléatoire doit être de bonne qualité. Idéalement :
  1. chaque bit de sortie vaut 0 ou 1 avec une probabilité $\frac{1}{2}$,
  2. aucun bit de sortie n'est corrélé avec les précédents ou les suivants,
  3. la période du générateur est suffisamment longue.

  Problème : ces trois conditions sont extrêmement difficiles à satisfaire en pratique.
- En l'absence d'une assise théorique solide, la **sécurité** des chiffrements à flot reste **problématique**. Pour preuve de cet état de fait, le nombre important d'algorithmes proposés qui ont été cassés...

Sécurité

- Une première difficulté : si on chiffre plusieurs messages avec la **même clé**, on génère le **même aléa**. On ajoute donc une entrée auxiliaire, le vecteur d'initialisation (*Initialization Vector - IV*) :



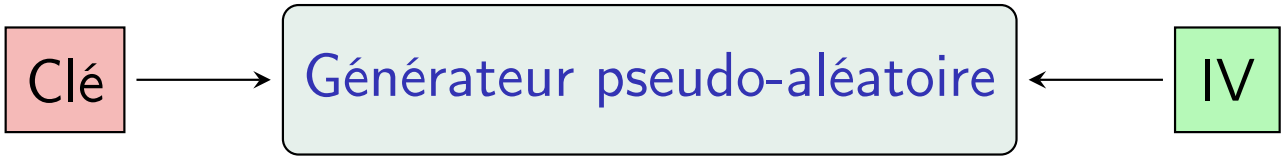
- Le générateur pseudo-aléatoire doit être de bonne qualité. Idéalement :
  1. chaque bit de sortie vaut 0 ou 1 avec une probabilité $\frac{1}{2}$,
  2. aucun bit de sortie n'est corrélé avec les précédents ou les suivants,
  3. la période du générateur est suffisamment longue.

Problème : ces trois conditions sont extrêmement difficiles à satisfaire en pratique.

- En l'absence d'une assise théorique solide, la **sécurité** des chiffrements à flot reste **problématique**. Pour preuve de cet état de fait, le nombre important d'algorithmes proposés qui ont été cassés...

Sécurité

- Une première difficulté : si on chiffre plusieurs messages avec la **même clé**, on génère le **même aléa**. On ajoute donc une entrée auxiliaire, le vecteur d'initialisation (*Initialization Vector - IV*) :

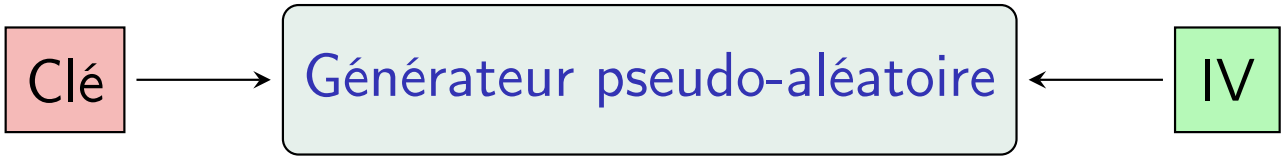


- Le générateur pseudo-aléatoire doit être de bonne qualité. Idéalement :
  1. chaque bit de sortie vaut 0 ou 1 avec une probabilité $\frac{1}{2}$,
  2. aucun bit de sortie n'est corrélé avec les précédents ou les suivants,
  3. la période du générateur est suffisamment longue.

  Problème : ces trois conditions sont extrêmement difficiles à satisfaire en pratique.
- En l'absence d'une assise théorique solide, la **sécurité** des chiffrements à flot reste **problématique**. Pour preuve de cet état de fait, le nombre important d'algorithmes proposés qui ont été cassés...

Sécurité

- Une première difficulté : si on chiffre plusieurs messages avec la **même clé**, on génère le **même aléa**. On ajoute donc une entrée auxiliaire, le vecteur d'initialisation (*Initialization Vector - IV*) :



- Le générateur pseudo-aléatoire doit être de bonne qualité. Idéalement :
  1. chaque bit de sortie vaut 0 ou 1 avec une probabilité $\frac{1}{2}$,
  2. aucun bit de sortie n'est corrélé avec les précédents ou les suivants,
  3. la période du générateur est suffisamment longue.

  Problème : ces trois conditions sont extrêmement difficiles à satisfaire en pratique.
- En l'absence d'une assise théorique solide, la **sécurité** des chiffrements à flot reste **problématique**. Pour preuve de cet état de fait, le nombre important d'algorithmes proposés qui ont été cassés...

Sécurité : un cas emblématique le WEP

- Le WEP (*Wired Equivalent Privacy*) génère (trop) simplement une clé de session k' à partir de la clé initiale k et du vecteur d'initialisation IV : $k' = IV \| k$. L'IV étant un des champs de l'en-tête d'un paquet WEP, un attaquant connaît déjà une partie de la clé de session !
- Le vecteur IV est codé sur seulement 24 bits. Avec l'attaque des anniversaires, il suffit d'environ seulement $\sqrt{2 \cdot 2^{24} \cdot \ln(2)} \approx 4822$ essais pour obtenir une collision avec une probabilité de 50%.
- Le protocole WEP n'indique pas comment est initialisé le vecteur IV. Du coup, certaines cartes WIFI l'initialisent de manière trop simple, introduisant une faiblesse supplémentaire dans le protocole.
- L'exploitation de ces faiblesses a permis (2001) de reconstituer la clé k en quelques heures. Depuis 2006, on sait pénétrer (en utilisant la fragmentation des paquets) un réseau WEP en quelques secondes !
- Aujourd'hui, pour sécuriser les sessions WIFI, on utilise le protocole WPA2 (*Wi-Fi Protected Access*, IEEE 802.11i).

Sécurité : un cas emblématique le WEP

- Le WEP (*Wired Equivalent Privacy*) génère (trop) simplement une clé de session k' à partir de la clé initiale k et du vecteur d'initialisation IV : $k' = IV \| k$. L'IV étant un des champs de l'en-tête d'un paquet WEP, un attaquant connaît déjà une partie de la clé de session !
- Le vecteur IV est codé sur seulement 24 bits. Avec l'attaque des anniversaires, il suffit d'environ seulement $\sqrt{2 \cdot 2^{24} \cdot \ln(2)} \approx 4822$ essais pour obtenir une collision avec une probabilité de 50%.
- Le protocole WEP n'indique pas comment est initialisé le vecteur IV. Du coup, certaines cartes WIFI l'initialisent de manière trop simple, introduisant une faiblesse supplémentaire dans le protocole.
- L'exploitation de ces faiblesses a permis (2001) de reconstituer la clé k en quelques heures. Depuis 2006, on sait pénétrer (en utilisant la fragmentation des paquets) un réseau WEP en quelques secondes !
- Aujourd'hui, pour sécuriser les sessions WIFI, on utilise le protocole WPA2 (*Wi-Fi Protected Access*, IEEE 802.11i).

Sécurité : un cas emblématique le WEP

- Le WEP (*Wired Equivalent Privacy*) génère (trop) simplement une clé de session k' à partir de la clé initiale k et du vecteur d'initialisation IV : $k' = IV \parallel k$. L'IV étant un des champs de l'en-tête d'un paquet WEP, un attaquant connaît déjà une partie de la clé de session !
- Le vecteur IV est codé sur seulement 24 bits. Avec l'attaque des anniversaires, il suffit d'environ seulement $\sqrt{2 \cdot 2^{24} \cdot \ln(2)} \approx 4822$ essais pour obtenir une collision avec une probabilité de 50%.
- Le protocole WEP n'indique pas comment est initialisé le vecteur IV. Du coup, certaines cartes WIFI l'initialisent de manière trop simple, introduisant une faiblesse supplémentaire dans le protocole.
- L'exploitation de ces faiblesses a permis (2001) de reconstituer la clé k en quelques heures. Depuis 2006, on sait pénétrer (en utilisant la fragmentation des paquets) un réseau WEP en quelques secondes !
- Aujourd'hui, pour sécuriser les sessions WIFI, on utilise le protocole WPA2 (*Wi-Fi Protected Access*, IEEE 802.11i).

Sécurité : un cas emblématique le WEP

- Le WEP (*Wired Equivalent Privacy*) génère (trop) simplement une clé de session k' à partir de la clé initiale k et du vecteur d'initialisation IV : $k' = IV \parallel k$. L'IV étant un des champs de l'en-tête d'un paquet WEP, un attaquant connaît déjà une partie de la clé de session !
- Le vecteur IV est codé sur seulement 24 bits. Avec l'attaque des anniversaires, il suffit d'environ seulement $\sqrt{2 \cdot 2^{24} \cdot \ln(2)} \approx 4822$ essais pour obtenir une collision avec une probabilité de 50%.
- Le protocole WEP n'indique pas comment est initialisé le vecteur IV. Du coup, certaines cartes WIFI l'initialisent de manière trop simple, introduisant une faiblesse supplémentaire dans le protocole.
- L'exploitation de ces faiblesses a permis (2001) de reconstituer la clé k en quelques heures. Depuis 2006, on sait pénétrer (en utilisant la fragmentation des paquets) un réseau WEP en quelques secondes !
- Aujourd'hui, pour sécuriser les sessions WIFI, on utilise le protocole WPA2 (*Wi-Fi Protected Access*, IEEE 802.11i).

Sécurité : un cas emblématique le WEP

- Le WEP (*Wired Equivalent Privacy*) génère (trop) simplement une clé de session k' à partir de la clé initiale k et du vecteur d'initialisation IV : $k' = IV \parallel k$. L'IV étant un des champs de l'en-tête d'un paquet WEP, un attaquant connaît déjà une partie de la clé de session !
- Le vecteur IV est codé sur seulement 24 bits. Avec l'attaque des anniversaires, il suffit d'environ seulement $\sqrt{2 \cdot 2^{24} \cdot \ln(2)} \approx 4822$ essais pour obtenir une collision avec une probabilité de 50%.
- Le protocole WEP n'indique pas comment est initialisé le vecteur IV. Du coup, certaines cartes WIFI l'initialisent de manière trop simple, introduisant une faiblesse supplémentaire dans le protocole.
- L'exploitation de ces faiblesses a permis (2001) de reconstituer la clé k en quelques heures. Depuis 2006, on sait pénétrer (en utilisant la fragmentation des paquets) un réseau WEP en quelques secondes !
- Aujourd'hui, pour sécuriser les sessions WIFI, on utilise le protocole WPA2 (*Wi-Fi Protected Access*, IEEE 802.11i).

Sécurité : l'attaque KRACK du protocole WPA2

- Le protocole WPA2 lui-même souffrait d'une vulnérabilité due à un **défaut de conception** et découverte en 2016 par deux chercheurs belges *Mathy Vanhoef* et *Frank Piessens* (publiée en octobre 2017).
- L'attaque, baptisée KRACK (*Key Reinstallation Attack*), est une **attaque par rejeu** permettant de découvrir la clé de chiffrement de la session.
- En outre, l'implémentation **wpa_supplicant** (*Android* et *Linux*) avait une faiblesse supplémentaire permettant, via une attaque de type « homme du milieu », d'installer une clé de chiffrement composée uniquement de bits à zéro !
- Bien entendu, dès la publication de cette attaque, des correctifs ont été proposés par les fournisseurs concernés (c'est à dire à peu près tout le monde, vu que c'était un défaut de conception).
- Cette attaque rendait possible le détournement de connexions TCP et plus généralement l'injection de données malveillantes dans toute session non sécurisée (HTTP par exemple).

Sécurité : l'attaque KRACK du protocole WPA2

- Le protocole WPA2 lui-même souffrait d'une vulnérabilité due à un **défaut de conception** et découverte en 2016 par deux chercheurs belges *Mathy Vanhoef* et *Frank Piessens* (publiée en octobre 2017).
- L'attaque, baptisée KRACK (*Key Reinstallation Attack*), est une **attaque par rejeu** permettant de découvrir la clé de chiffrement de la session.
- En outre, l'implémentation **wpa_supplicant** (*Android* et *Linux*) avait une faiblesse supplémentaire permettant, via une attaque de type « homme du milieu », d'installer une clé de chiffrement composée uniquement de bits à zéro !
- Bien entendu, dès la publication de cette attaque, des correctifs ont été proposés par les fournisseurs concernés (c'est à dire à peu près tout le monde, vu que c'était un défaut de conception).
- Cette attaque rendait possible le détournement de connexions TCP et plus généralement l'injection de données malveillantes dans toute session non sécurisée (HTTP par exemple).

Sécurité : l'attaque KRACK du protocole WPA2

- Le protocole WPA2 lui-même souffrait d'une vulnérabilité due à un **défaut de conception** et découverte en 2016 par deux chercheurs belges *Mathy Vanhoef* et *Frank Piessens* (publiée en octobre 2017).
- L'attaque, baptisée KRACK (*Key Reinstallation Attack*), est une **attaque par rejeu** permettant de découvrir la clé de chiffrement de la session.
- En outre, l'implémentation **wpa_supplicant** (*Android* et *Linux*) avait une faiblesse supplémentaire permettant, via une attaque de type « homme du milieu », d'installer une clé de chiffrement composée uniquement de bits à zéro !
- Bien entendu, dès la publication de cette attaque, des correctifs ont été proposés par les fournisseurs concernés (c'est à dire à peu près tout le monde, vu que c'était un défaut de conception).
- Cette attaque rendait possible le détournement de connexions TCP et plus généralement l'injection de données malveillantes dans toute session non sécurisée (HTTP par exemple).

Sécurité : l'attaque KRACK du protocole WPA2

- Le protocole WPA2 lui-même souffrait d'une vulnérabilité due à un **défaut de conception** et découverte en 2016 par deux chercheurs belges *Mathy Vanhoef* et *Frank Piessens* (publiée en octobre 2017).
- L'attaque, baptisée KRACK (*Key Reinstallation Attack*), est une **attaque par rejeu** permettant de découvrir la clé de chiffrement de la session.
- En outre, l'implémentation **wpa_supplicant** (*Android* et *Linux*) avait une faiblesse supplémentaire permettant, via une attaque de type « homme du milieu », d'installer une clé de chiffrement composée uniquement de bits à zéro !
- Bien entendu, dès la publication de cette attaque, des correctifs ont été proposés par les fournisseurs concernés (c'est à dire à peu près tout le monde, vu que c'était un défaut de conception).
- Cette attaque rendait possible le détournement de connexions TCP et plus généralement l'injection de données malveillantes dans toute session non sécurisée (HTTP par exemple).

Sécurité : l'attaque KRACK du protocole WPA2

- Le protocole WPA2 lui-même souffrait d'une vulnérabilité due à un **défaut de conception** et découverte en 2016 par deux chercheurs belges *Mathy Vanhoef* et *Frank Piessens* (publiée en octobre 2017).
- L'attaque, baptisée KRACK (*Key Reinstallation Attack*), est une **attaque par rejeu** permettant de découvrir la clé de chiffrement de la session.
- En outre, l'implémentation **wpa_supplicant** (*Android* et *Linux*) avait une faiblesse supplémentaire permettant, via une attaque de type « homme du milieu », d'installer une clé de chiffrement composée uniquement de bits à zéro !
- Bien entendu, dès la publication de cette attaque, des correctifs ont été proposés par les fournisseurs concernés (c'est à dire à peu près tout le monde, vu que c'était un défaut de conception).
- Cette attaque rendait possible le détournement de connexions TCP et plus généralement l'injection de données malveillantes dans toute session non sécurisée (HTTP par exemple).

Principe général

- L'idée maitresse : une fonction de chiffrement f_K (K étant la clé secrète) est construite par **itérations successives** d'une fonction simple g_K : $f_K = g_K^d = \underbrace{g_K \circ \cdots \circ g_K}_{d \text{ fois}}$.
- On sait depuis l'étude des *systèmes dynamiques* que le comportement de g^d peut être **imprévisible** (pour d assez grand), même si g est **très simple** (ensemble de *Mandelbrot*, $g(z) = z^2 + c$).
- Plus précisément, on dérive $K_1, \ldots, K_d$ sous-clés de la clé principale K et $f_K = g_{K_d} \circ \cdots \circ g_{K_1}$ (*algorithme de dérivation de sous-clés*).
- Chaque *fonction de tour* g_{K_i} est optimisée : opérations simples.
- Les algorithmes de chiffrement par bloc sont **performants** et leur **sécurité bien étudiée**.
- Inconvénient : nécessité d'avoir recours à l'utilisation de **modes opératoires**.

Principe général

- L'idée maitresse : une fonction de chiffrement f_K (K étant la clé secrète) est construite par **itérations successives** d'une fonction simple g_K : $f_K = g_K^d = \underbrace{g_K \circ \cdots \circ g_K}_{d \text{ fois}}$.
- On sait depuis l'étude des *systèmes dynamiques* que le comportement de g^d peut être **imprévisible** (pour d assez grand), même si g est **très simple** (ensemble de *Mandelbrot*, $g(z) = z^2 + c$).
- Plus précisément, on dérive $K_1, \ldots, K_d$ sous-clés de la clé principale K et $f_K = g_{K_d} \circ \cdots \circ g_{K_1}$ (*algorithme de dérivation de sous-clés*).
- Chaque *fonction de tour* g_{K_i} est optimisée : opérations simples.
- Les algorithmes de chiffrement par bloc sont **performants** et leur **sécurité bien étudiée**.
- Inconvénient : nécessité d'avoir recours à l'utilisation de **modes opératoires**.

Principe général

- L'idée maitresse : une fonction de chiffrement f_K (K étant la clé secrète) est construite par **itérations successives** d'une fonction simple g_K : $f_K = g_K^d = \underbrace{g_K \circ \cdots \circ g_K}_{d \text{ fois}}$.
- On sait depuis l'étude des *systèmes dynamiques* que le comportement de g^d peut être **imprévisible** (pour d assez grand), même si g est **très simple** (ensemble de *Mandelbrot*, $g(z) = z^2 + c$).
- Plus précisément, on dérive $K_1, \ldots, K_d$ sous-clés de la clé principale K et $f_K = g_{K_d} \circ \cdots \circ g_{K_1}$ (*algorithme de dérivation de sous-clés*).
- Chaque *fonction de tour* g_{K_i} est optimisée : opérations simples.
- Les algorithmes de chiffrement par bloc sont **performants** et leur **sécurité bien étudiée**.
- Inconvénient : nécessité d'avoir recours à l'utilisation de **modes opératoires**.

Principe général

- L'idée maitresse : une fonction de chiffrement f_K (K étant la clé secrète) est construite par **itérations successives** d'une fonction simple g_K : $f_K = g_K^d = \underbrace{g_K \circ \cdots \circ g_K}_{d \text{ fois}}$.
- On sait depuis l'étude des *systèmes dynamiques* que le comportement de g^d peut être **imprévisible** (pour d assez grand), même si g est **très simple** (ensemble de *Mandelbrot*, $g(z) = z^2 + c$).
- Plus précisément, on dérive $K_1, \ldots, K_d$ sous-clés de la clé principale K et $f_K = g_{K_d} \circ \cdots \circ g_{K_1}$ (*algorithme de dérivation de sous-clés*).
- Chaque *fonction de tour* g_{K_i} est optimisée : opérations simples.
- Les algorithmes de chiffrement par bloc sont **performants** et leur **sécurité bien étudiée**.
- Inconvénient : nécessité d'avoir recours à l'utilisation de **modes opératoires**.

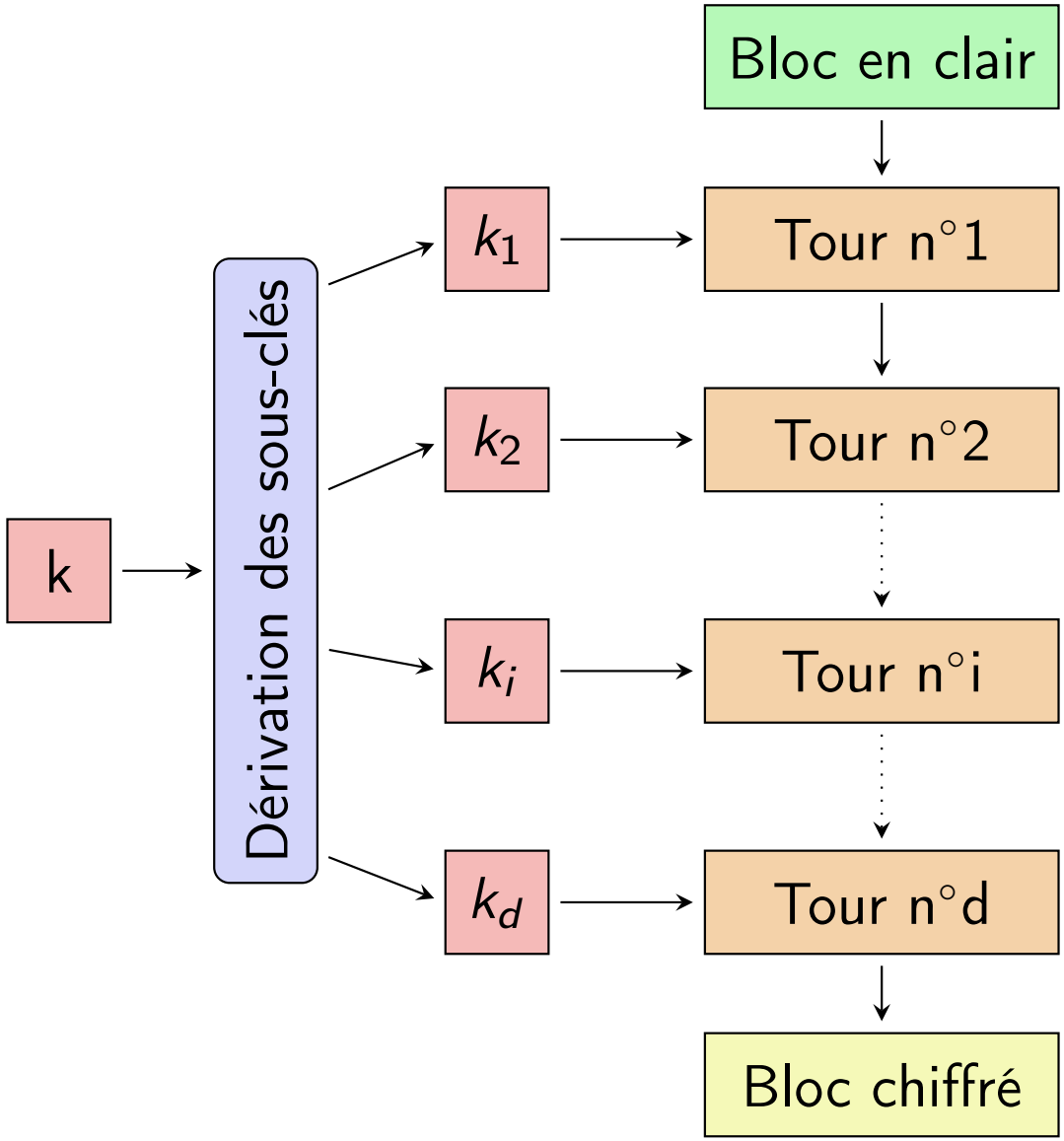
Principe général

- L'idée maitresse : une fonction de chiffrement f_K (K étant la clé secrète) est construite par **itérations successives** d'une fonction simple g_K : $f_K = g_K^d = \underbrace{g_K \circ \cdots \circ g_K}_{d \text{ fois}}$.
- On sait depuis l'étude des *systèmes dynamiques* que le comportement de g^d peut être **imprévisible** (pour d assez grand), même si g est **très simple** (ensemble de *Mandelbrot*, $g(z) = z^2 + c$).
- Plus précisément, on dérive $K_1, \ldots, K_d$ sous-clés de la clé principale K et $f_K = g_{K_d} \circ \cdots \circ g_{K_1}$ (*algorithme de dérivation de sous-clés*).
- Chaque *fonction de tour* g_{K_i} est optimisée : opérations simples.
- Les algorithmes de chiffrement par bloc sont **performants** et leur **sécurité bien étudiée**.
- Inconvénient : nécessité d'avoir recours à l'utilisation de **modes opératoires**.

Principe général

- L'idée maitresse : une fonction de chiffrement f_K (K étant la clé secrète) est construite par **itérations successives** d'une fonction simple g_K : $f_K = g_K^d = \underbrace{g_K \circ \cdots \circ g_K}_{d \text{ fois}}$.
- On sait depuis l'étude des *systèmes dynamiques* que le comportement de g^d peut être **imprévisible** (pour d assez grand), même si g est **très simple** (ensemble de *Mandelbrot*, $g(z) = z^2 + c$).
- Plus précisément, on dérive $K_1, \ldots, K_d$ sous-clés de la clé principale K et $f_K = g_{K_d} \circ \cdots \circ g_{K_1}$ (*algorithme de dérivation de sous-clés*).
- Chaque *fonction de tour* g_{K_i} est optimisée : opérations simples.
- Les algorithmes de chiffrement par bloc sont **performants** et leur **sécurité bien étudiée**.
- Inconvénient : nécessité d'avoir recours à l'utilisation de **modes opératoires**.

Architecture d'un chiffrement à d tours



Définition et propriétés

- Un réseau de Feistel, du nom de son inventeur *Horst Feistel* cryptologue chez IBM, est un dispositif général de chiffrement par blocs au cœur de nombre d'algorithmes symétriques comme : DES, 3DES, Blowfish, Twofish, Camellia, SEED, RC5, OAEP,...
- Soit $\mathscr{M} = \{0,1\}^{2n}$, l'ensemble des messages à chiffrer. On va définir une fonction de chiffrement de $\mathscr{M}$ vers l'ensemble des messages chiffrés $\mathscr{C} = \mathscr{M}$. La clé secrète est un ensemble $K = \{K_1, \ldots, K_d\}$ avec $K_i \in \{0,1\}^k$. On se donne également une fonction **quelconque** $F : \{0,1\}^{k+n} \to \{0,1\}^n$.
- Soit $M \in \mathscr{M}$ un message en clair. On le découpe en deux blocs de même longueur (il existe des variantes avec des longueurs différentes) $M = (L, R)$ avec $L, R \in \{0,1\}^n$. On définit par récurrence la suite finie $(L_i, R_i)_{0 \leq i \leq d}$ comme suit :

$$\begin{cases} (L_0, R_0) = (L, R), \\ (L_i, R_i) = (R_{i-1}, L_{i-1} \oplus F(K_i, R_{i-1})), \text{ pour } 1 \leq i \leq d. \end{cases}$$

Définition et propriétés

- Un réseau de Feistel, du nom de son inventeur *Horst Feistel* cryptologue chez IBM, est un dispositif général de chiffrement par blocs au cœur de nombre d'algorithmes symétriques comme : DES, 3DES, Blowfish, Twofish, Camellia, SEED, RC5, OAEP,...
- Soit $\mathscr{M} = \{0,1\}^{2n}$, l'ensemble des messages à chiffrer. On va définir une fonction de chiffrement de $\mathscr{M}$ vers l'ensemble des messages chiffrés $\mathscr{C} = \mathscr{M}$. La clé secrète est un ensemble $K = \{K_1, \ldots, K_d\}$ avec $K_i \in \{0,1\}^k$. On se donne également une fonction **quelconque** $F : \{0,1\}^{k+n} \to \{0,1\}^n$.
- Soit $M \in \mathscr{M}$ un message en clair. On le découpe en deux blocs de même longueur (il existe des variantes avec des longueurs différentes) $M = (L, R)$ avec $L, R \in \{0,1\}^n$. On définit par récurrence la suite finie $(L_i, R_i)_{0 \leq i \leq d}$ comme suit :

$$\begin{cases} (L_0, R_0) = (L, R), \\ (L_i, R_i) = (R_{i-1}, L_{i-1} \oplus F(K_i, R_{i-1})), \text{ pour } 1 \leq i \leq d. \end{cases}$$

Définition et propriétés

- Un réseau de Feistel, du nom de son inventeur *Horst Feistel* cryptologue chez IBM, est un dispositif général de chiffrement par blocs au cœur de nombre d'algorithmes symétriques comme : DES, 3DES, Blowfish, Twofish, Camellia, SEED, RC5, OAEP,...
- Soit $\mathscr{M} = \{0,1\}^{2n}$, l'ensemble des messages à chiffrer. On va définir une fonction de chiffrement de $\mathscr{M}$ vers l'ensemble des messages chiffrés $\mathscr{C} = \mathscr{M}$. La clé secrète est un ensemble $K = \{K_1, \ldots, K_d\}$ avec $K_i \in \{0,1\}^k$. On se donne également une fonction **quelconque** $F : \{0,1\}^{k+n} \to \{0,1\}^n$.
- Soit $M \in \mathscr{M}$ un message en clair. On le découpe en deux blocs de même longueur (il existe des variantes avec des longueurs différentes) $M = (L, R)$ avec $L, R \in \{0,1\}^n$. On définit par récurrence la suite finie $(L_i, R_i)_{0 \leq i \leq d}$ comme suit :

$$\begin{cases} (L_0, R_0) = (L, R), \\ (L_i, R_i) = (R_{i-1}, L_{i-1} \oplus F(K_i, R_{i-1})), \text{ pour } 1 \leq i \leq d. \end{cases}$$

Définition et propriétés (suite)

- Le chiffré du message $M = (L, R)$ est alors $C = (L_d, R_d)$.
- L'avantage principal d'un réseau de Feistel est que, pour déchiffrer un message, il n'est pas nécessaire de supposer inversibles les fonctions $F_i : X \to F(K_i, X)$, pour $1 \leq i \leq d$, et de savoir les inverser.
- En effet, on montre aisément que : $(R_{i-1}, L_{i-1}) = (L_i, R_i \oplus F(K_i, L_i))$.
- Un exemple : le DES (*Digital Encryption Standard*). Ici on a $n = 32$, $d = 16$ et $k = 48$. La clé secrète $K = \{K_1, \ldots, K_{16}\}$ est déduite d'une clé maître de 56 bits (cassable aujourd'hui en moins de 24h).
  Enfin on a : $F_i(X) = P(S(L(X)) \oplus K_i)$ où :
  1- L est une application linéaire de $\mathbb{F}_2^{32}$ dans $\mathbb{F}_2^{48}$,
  2- S est une application non linéaire (*S-Box*) de $\{0,1\}^{48}$ dans $\{0,1\}^{32}$,
  3- P une permutation de $\{0,1\}^{32}$.

Définition et propriétés (suite)

- Le chiffré du message $M = (L, R)$ est alors $C = (L_d, R_d)$.
- L'avantage principal d'un réseau de Feistel est que, pour déchiffrer un message, il n'est pas nécessaire de supposer inversibles les fonctions $F_i : X \to F(K_i, X)$, pour $1 \leq i \leq d$, et de savoir les inverser.
- En effet, on montre aisément que : $(R_{i-1}, L_{i-1}) = (L_i, R_i \oplus F(K_i, L_i))$.
- Un exemple : le DES (*Digital Encryption Standard*). Ici on a $n = 32$, $d = 16$ et $k = 48$. La clé secrète $K = \{K_1, \ldots, K_{16}\}$ est déduite d'une clé maître de 56 bits (cassable aujourd'hui en moins de 24h).
  Enfin on a : $F_i(X) = P(S(L(X)) \oplus K_i)$ où :
  1- L est une application linéaire de $\mathbb{F}_2^{32}$ dans $\mathbb{F}_2^{48}$,
  2- S est une application non linéaire (*S-Box*) de $\{0,1\}^{48}$ dans $\{0,1\}^{32}$,
  3- P une permutation de $\{0,1\}^{32}$.

Définition et propriétés (suite)

- Le chiffré du message $M = (L, R)$ est alors $C = (L_d, R_d)$.
- L'avantage principal d'un réseau de Feistel est que, pour déchiffrer un message, il n'est pas nécessaire de supposer inversibles les fonctions $F_i : X \to F(K_i, X)$, pour $1 \leq i \leq d$, et de savoir les inverser.
- En effet, on montre aisément que : $(R_{i-1}, L_{i-1}) = (L_i, R_i \oplus F(K_i, L_i))$.
- Un exemple : le DES (*Digital Encryption Standard*). Ici on a $n = 32$, $d = 16$ et $k = 48$. La clé secrète $K = \{K_1, \ldots, K_{16}\}$ est déduite d'une clé maître de 56 bits (cassable aujourd'hui en moins de 24h).
  Enfin on a : $F_i(X) = P(S(L(X)) \oplus K_i)$ où :
  1- L est une application linéaire de $\mathbb{F}_2^{32}$ dans $\mathbb{F}_2^{48}$,
  2- S est une application non linéaire (*S-Box*) de $\{0,1\}^{48}$ dans $\{0,1\}^{32}$,
  3- P une permutation de $\{0,1\}^{32}$.

Définition et propriétés (suite)

- Le chiffré du message $M = (L, R)$ est alors $C = (L_d, R_d)$.
- L'avantage principal d'un réseau de Feistel est que, pour déchiffrer un message, il n'est pas nécessaire de supposer inversibles les fonctions $F_i : X \to F(K_i, X)$, pour $1 \leq i \leq d$, et de savoir les inverser.
- En effet, on montre aisément que : $(R_{i-1}, L_{i-1}) = (L_i, R_i \oplus F(K_i, L_i))$.
- Un exemple : le DES (*Digital Encryption Standard*). Ici on a $n = 32$, $d = 16$ et $k = 48$. La clé secrète $K = \{K_1, \ldots, K_{16}\}$ est déduite d'une clé maître de 56 bits (cassable aujourd'hui en moins de 24h).
  Enfin on a : $F_i(X) = P(S(L(X)) \oplus K_i)$ où :
  1- L est une application linéaire de $\mathbb{F}_2^{32}$ dans $\mathbb{F}_2^{48}$,
  2- S est une application non linéaire (*S-Box*) de $\{0,1\}^{48}$ dans $\{0,1\}^{32}$,
  3- P une permutation de $\{0,1\}^{32}$.

Définition et propriétés (suite)

- Le chiffré du message $M = (L, R)$ est alors $C = (L_d, R_d)$.
- L'avantage principal d'un réseau de Feistel est que, pour déchiffrer un message, il n'est pas nécessaire de supposer inversibles les fonctions $F_i : X \to F(K_i, X)$, pour $1 \leq i \leq d$, et de savoir les inverser.
- En effet, on montre aisément que : $(R_{i-1}, L_{i-1}) = (L_i, R_i \oplus F(K_i, L_i))$.
- Un exemple : le DES (*Digital Encryption Standard*). Ici on a $n = 32$, $d = 16$ et $k = 48$. La clé secrète $K = \{K_1, \ldots, K_{16}\}$ est déduite d'une clé maître de 56 bits (cassable aujourd'hui en moins de 24h).
  Enfin on a : $F_i(X) = P(S(L(X)) \oplus K_i)$ où :
  1- L est une application linéaire de $\mathbb{F}_2^{32}$ dans $\mathbb{F}_2^{48}$,
  2- S est une application non linéaire (*S-Box*) de $\{0,1\}^{48}$ dans $\{0,1\}^{32}$,
  3- P une permutation de $\{0,1\}^{32}$.

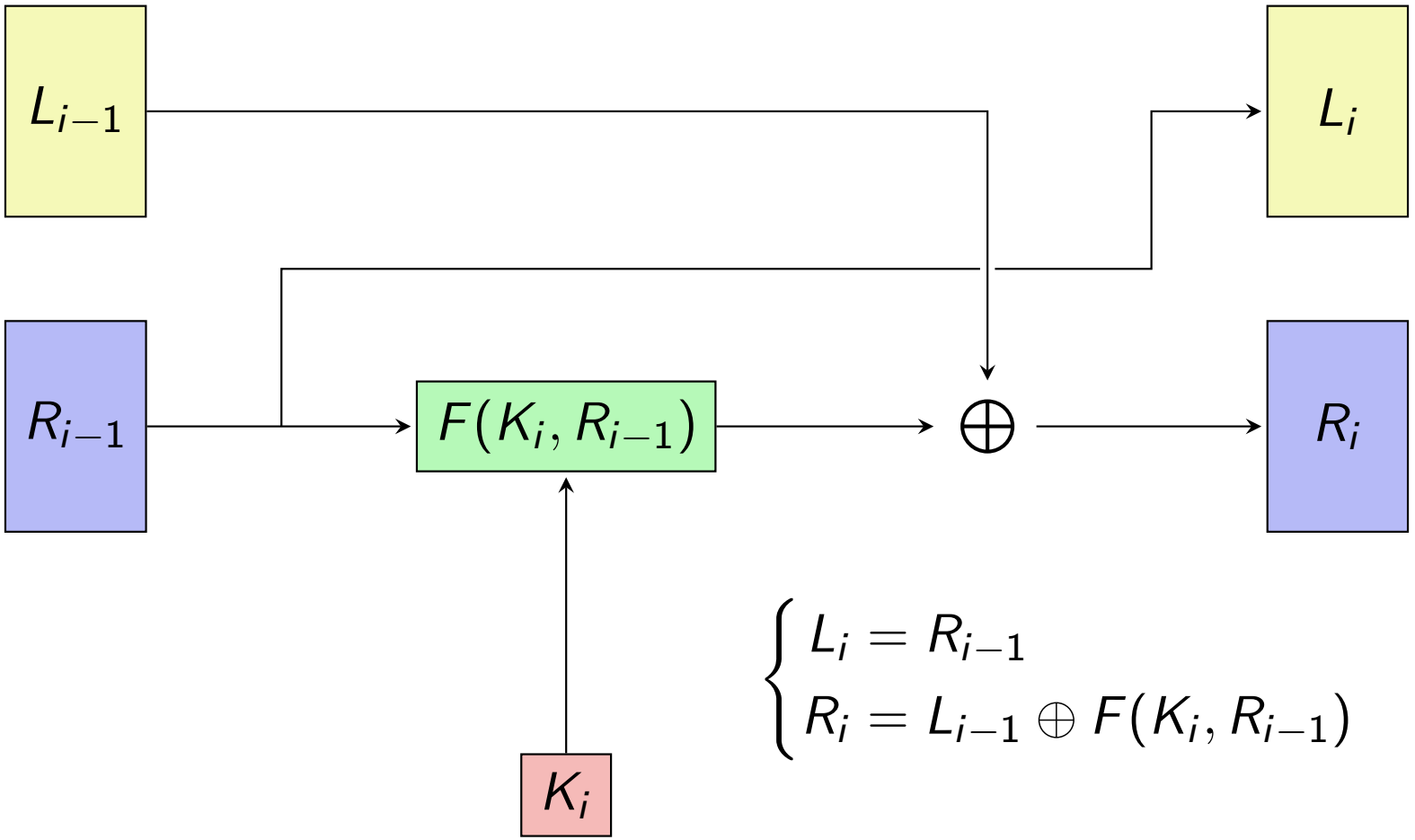
Définition et propriétés (suite)

- Le chiffré du message $M = (L, R)$ est alors $C = (L_d, R_d)$.
- L'avantage principal d'un réseau de Feistel est que, pour déchiffrer un message, il n'est pas nécessaire de supposer inversibles les fonctions $F_i : X \to F(K_i, X)$, pour $1 \leq i \leq d$, et de savoir les inverser.
- En effet, on montre aisément que : $(R_{i-1}, L_{i-1}) = (L_i, R_i \oplus F(K_i, L_i))$.
- Un exemple : le DES (*Digital Encryption Standard*). Ici on a $n = 32$, $d = 16$ et $k = 48$. La clé secrète $K = \{K_1, \ldots, K_{16}\}$ est déduite d'une clé maître de 56 bits (cassable aujourd'hui en moins de 24h).
  Enfin on a : $F_i(X) = P(S(L(X)) \oplus K_i)$ où :
  1- L est une application linéaire de $\mathbb{F}_2^{32}$ dans $\mathbb{F}_2^{48}$,
  2- S est une application non linéaire (*S-Box*) de $\{0,1\}^{48}$ dans $\{0,1\}^{32}$,
  3- P une permutation de $\{0,1\}^{32}$.

Définition et propriétés (suite)

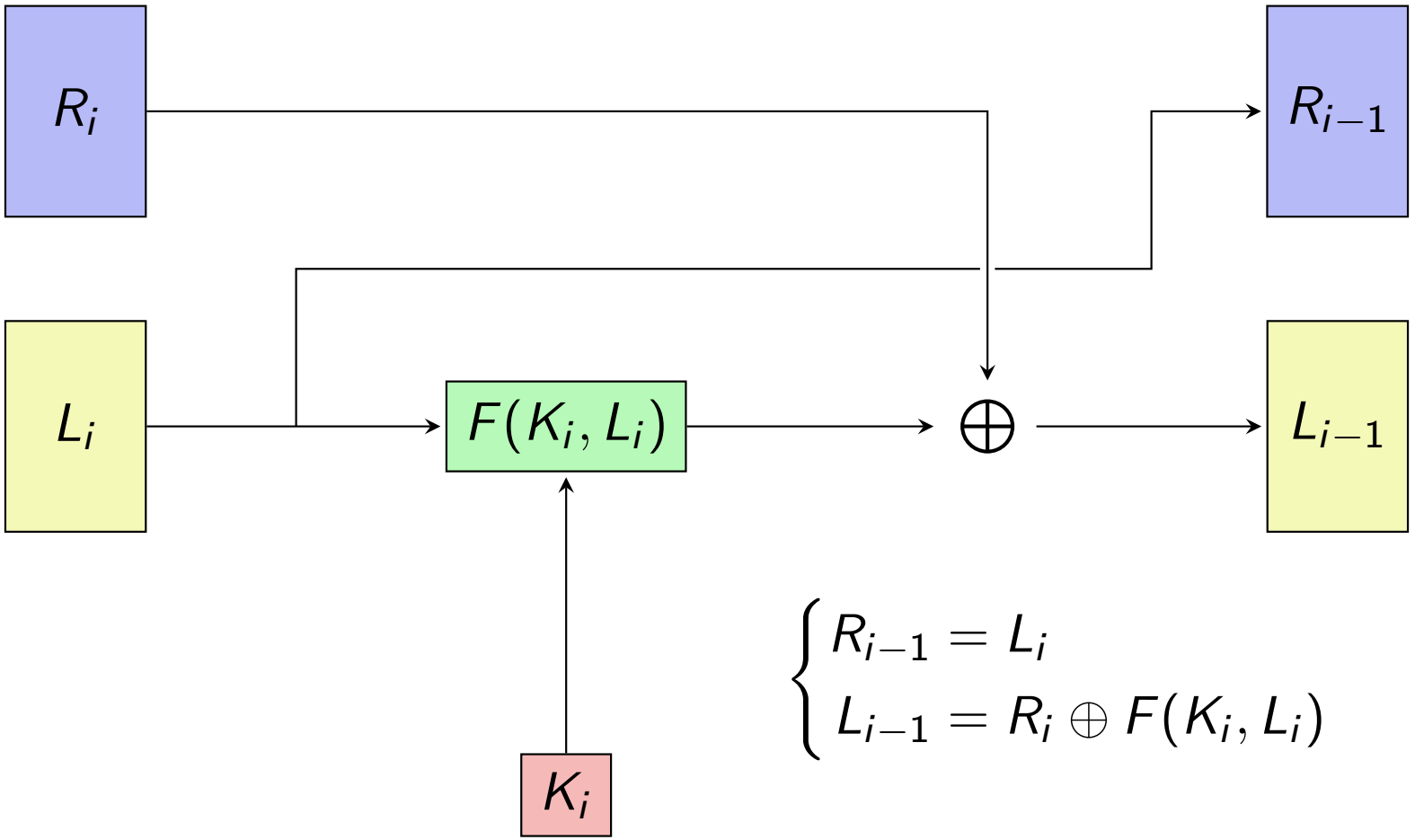
- Le chiffré du message $M = (L, R)$ est alors $C = (L_d, R_d)$.
- L'avantage principal d'un réseau de Feistel est que, pour déchiffrer un message, il n'est pas nécessaire de supposer inversibles les fonctions $F_i : X \to F(K_i, X)$, pour $1 \leq i \leq d$, et de savoir les inverser.
- En effet, on montre aisément que : $(R_{i-1}, L_{i-1}) = (L_i, R_i \oplus F(K_i, L_i))$.
- Un exemple : le DES (*Digital Encryption Standard*). Ici on a $n = 32$, $d = 16$ et $k = 48$. La clé secrète $K = \{K_1, \ldots, K_{16}\}$ est déduite d'une clé maître de 56 bits (cassable aujourd'hui en moins de 24h).
  Enfin on a : $F_i(X) = P(S(L(X)) \oplus K_i)$ où :
  1- L est une application linéaire de $\mathbb{F}_2^{32}$ dans $\mathbb{F}_2^{48}$,
  2- S est une application non linéaire (*S-Box*) de $\{0,1\}^{48}$ dans $\{0,1\}^{32}$,
  3- P une permutation de $\{0,1\}^{32}$.

Schéma d'un tour - Chiffrement



$$\begin{cases} L_i = R_{i-1} \\ R_i = L_{i-1} \oplus F(K_i, R_{i-1}) \end{cases}$$

Schéma d'un tour - Déchiffrement



$$\begin{cases} R_{i-1} = L_i \\ L_{i-1} = R_i \oplus F(K_i, L_i) \end{cases}$$

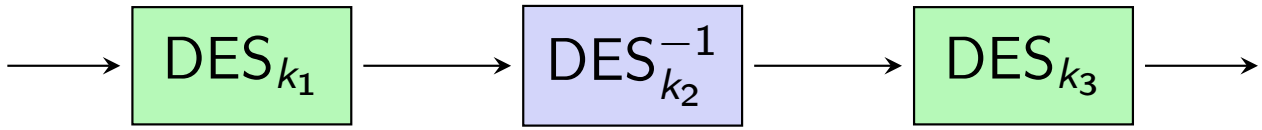
Le triple DES

- L'algorithme DES, standardisé en 1976, n'est plus utilisé aujourd'hui, non pas en raison d'une faiblesse structurelle (jamais découverte à ce jour), mais à cause de sa clé trop courte (56 bits, longueur imposée par la NSA, alors qu'initialement elle était de 64 bits).
- Son successeur est le triple DES ou 3DES (clé de 3×56 bits). Il existe plusieurs variantes dont celle recommandée par le NIST et répandue dans le monde bancaire. Elle nécessite trois (ou deux) clés k_1, k_2 et k_3 ($k_3 = k_1$ dans le cas de deux clés) :

- Inconvénient : 3DES est trois plus lent que DES.
- Alors pourquoi pas simplement un double DES ? Parce qu'il existe une attaque permettant de retrouver les deux clés secrètes, de complexité en temps 2^{57} au lieu de 2^{112} attendu (force brute).

Le triple DES

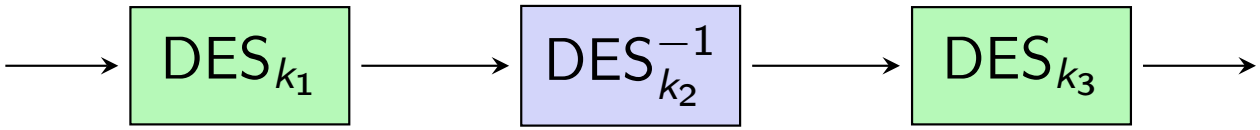
- L'algorithme DES, standardisé en 1976, n'est plus utilisé aujourd'hui, non pas en raison d'une faiblesse structurelle (jamais découverte à ce jour), mais à cause de sa clé trop courte (56 bits, longueur imposée par la NSA, alors qu'initialement elle était de 64 bits).
- Son successeur est le triple DES ou 3DES (clé de 3×56 bits). Il existe plusieurs variantes dont celle recommandée par le NIST et répandue dans le monde bancaire. Elle nécessite trois (ou deux) clés k_1, k_2 et k_3 ($k_3 = k_1$ dans le cas de deux clés) :



- Inconvénient : 3DES est trois plus lent que DES.
- Alors pourquoi pas simplement un double DES ? Parce qu'il existe une attaque permettant de retrouver les deux clés secrètes, de complexité en temps 2^{57} au lieu de 2^{112} attendu (force brute).

Le triple DES

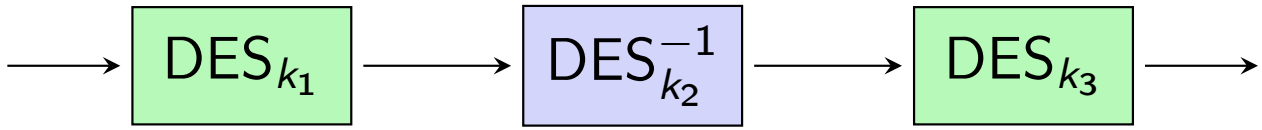
- L'algorithme DES, standardisé en 1976, n'est plus utilisé aujourd'hui, non pas en raison d'une faiblesse structurelle (jamais découverte à ce jour), mais à cause de sa clé trop courte (56 bits, longueur imposée par la NSA, alors qu'initialement elle était de 64 bits).
- Son successeur est le triple DES ou 3DES (clé de 3×56 bits). Il existe plusieurs variantes dont celle recommandée par le NIST et répandue dans le monde bancaire. Elle nécessite trois (ou deux) clés k_1, k_2 et k_3 ($k_3 = k_1$ dans le cas de deux clés) :



- Inconvénient : 3DES est trois plus lent que DES.
- Alors pourquoi pas simplement un double DES ? Parce qu'il existe une attaque permettant de retrouver les deux clés secrètes, de complexité en temps 2^{57} au lieu de 2^{112} attendu (force brute).

Le triple DES

- L'algorithme DES, standardisé en 1976, n'est plus utilisé aujourd'hui, non pas en raison d'une faiblesse structurelle (jamais découverte à ce jour), mais à cause de sa clé trop courte (56 bits, longueur imposée par la NSA, alors qu'initialement elle était de 64 bits).
- Son successeur est le triple DES ou 3DES (clé de 3×56 bits). Il existe plusieurs variantes dont celle recommandée par le NIST et répandue dans le monde bancaire. Elle nécessite trois (ou deux) clés k_1, k_2 et k_3 ($k_3 = k_1$ dans le cas de deux clés) :



- Inconvénient : 3DES est trois plus lent que DES.
- Alors pourquoi pas simplement un double DES ? Parce qu'il existe une attaque permettant de retrouver les deux clés secrètes, de complexité en temps 2^{57} au lieu de 2^{112} attendu (force brute).

Historique & Description

- AES est un algorithme de **chiffrement symétrique par bloc**. Il a remporté en 2000 le concours lancé par le NIST en 1997.
- Il y avait 15 participants à ce concours et c'est la proposition *Rijndael*, du nom de ses concepteurs belges *Joan Daemen* et *Vincent Rijmen*, qui a été retenue pour succéder au DES.
- AES est un sous-ensemble de la proposition initiale *Rijndael* : la taille des blocs est **fixée à 128 bits** (au lieu d'une taille variable multiple de 32 bits et comprise entre 128 bits et 256 bits).
- AES se décline en trois versions :
  - **AES-128** $\Rightarrow$ clé de 128 bits, 10 tours
  - **AES-192** $\Rightarrow$ clé de 192 bits, 12 tours
  - **AES-256** $\Rightarrow$ clé de 256 bits, 14 tours
- L'originalité d'AES est que la plupart des opérations se font dans le corps fini $\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$.

Historique & Description

- AES est un algorithme de **chiffrement symétrique par bloc**. Il a remporté en 2000 le concours lancé par le NIST en 1997.
- Il y avait 15 participants à ce concours et c'est la proposition *Rijndael*, du nom de ses concepteurs belges *Joan Daemen* et *Vincent Rijmen*, qui a été retenue pour succéder au DES.
- AES est un sous-ensemble de la proposition initiale *Rijndael* : la taille des blocs est **fixée à 128 bits** (au lieu d'une taille variable multiple de 32 bits et comprise entre 128 bits et 256 bits).
- AES se décline en trois versions :
  - **AES-128** $\Rightarrow$ clé de 128 bits, 10 tours
  - **AES-192** $\Rightarrow$ clé de 192 bits, 12 tours
  - **AES-256** $\Rightarrow$ clé de 256 bits, 14 tours
- L'originalité d'AES est que la plupart des opérations se font dans le corps fini $\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$.

Historique & Description

- AES est un algorithme de **chiffrement symétrique par bloc**. Il a remporté en 2000 le concours lancé par le NIST en 1997.
- Il y avait 15 participants à ce concours et c'est la proposition *Rijndael*, du nom de ses concepteurs belges *Joan Daemen* et *Vincent Rijmen*, qui a été retenue pour succéder au DES.
- AES est un sous-ensemble de la proposition initiale *Rijndael* : la taille des blocs est **fixée à 128 bits** (au lieu d'une taille variable multiple de 32 bits et comprise entre 128 bits et 256 bits).
- AES se décline en trois versions :
  - **AES-128** $\Rightarrow$ clé de 128 bits, 10 tours
  - **AES-192** $\Rightarrow$ clé de 192 bits, 12 tours
  - **AES-256** $\Rightarrow$ clé de 256 bits, 14 tours
- L'originalité d'AES est que la plupart des opérations se font dans le corps fini $\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$.

Historique & Description

- AES est un algorithme de **chiffrement symétrique par bloc**. Il a remporté en 2000 le concours lancé par le NIST en 1997.
- Il y avait 15 participants à ce concours et c'est la proposition *Rijndael*, du nom de ses concepteurs belges *Joan Daemen* et *Vincent Rijmen*, qui a été retenue pour succéder au DES.
- AES est un sous-ensemble de la proposition initiale *Rijndael* : la taille des blocs est **fixée à 128 bits** (au lieu d'une taille variable multiple de 32 bits et comprise entre 128 bits et 256 bits).
- AES se décline en trois versions :
  - **AES-128** $\Rightarrow$ clé de 128 bits, 10 tours
  - **AES-192** $\Rightarrow$ clé de 192 bits, 12 tours
  - **AES-256** $\Rightarrow$ clé de 256 bits, 14 tours
- L'originalité d'AES est que la plupart des opérations se font dans le corps fini $\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$.

Historique & Description

- AES est un algorithme de **chiffrement symétrique par bloc**. Il a remporté en 2000 le concours lancé par le NIST en 1997.
- Il y avait 15 participants à ce concours et c'est la proposition *Rijndael*, du nom de ses concepteurs belges *Joan Daemen* et *Vincent Rijmen*, qui a été retenue pour succéder au DES.
- AES est un sous-ensemble de la proposition initiale *Rijndael* : la taille des blocs est **fixée à 128 bits** (au lieu d'une taille variable multiple de 32 bits et comprise entre 128 bits et 256 bits).
- AES se décline en trois versions :
  - **AES-128** $\Rightarrow$ clé de 128 bits, 10 tours
  - **AES-192** $\Rightarrow$ clé de 192 bits, 12 tours
  - **AES-256** $\Rightarrow$ clé de 256 bits, 14 tours
- L'originalité d'AES est que la plupart des opérations se font dans le corps fini $\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$.

Historique & Description

- AES est un algorithme de **chiffrement symétrique par bloc**. Il a remporté en 2000 le concours lancé par le NIST en 1997.
- Il y avait 15 participants à ce concours et c'est la proposition *Rijndael*, du nom de ses concepteurs belges *Joan Daemen* et *Vincent Rijmen*, qui a été retenue pour succéder au DES.
- AES est un sous-ensemble de la proposition initiale *Rijndael* : la taille des blocs est **fixée à 128 bits** (au lieu d'une taille variable multiple de 32 bits et comprise entre 128 bits et 256 bits).
- AES se décline en trois versions :
  - **AES-128** $\Rightarrow$ clé de 128 bits, 10 tours
  - **AES-192** $\Rightarrow$ clé de 192 bits, 12 tours
  - **AES-256** $\Rightarrow$ clé de 256 bits, 14 tours
- L'originalité d'AES est que la plupart des opérations se font dans le corps fini $\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$.

Historique & Description

- AES est un algorithme de **chiffrement symétrique par bloc**. Il a remporté en 2000 le concours lancé par le NIST en 1997.
- Il y avait 15 participants à ce concours et c'est la proposition *Rijndael*, du nom de ses concepteurs belges *Joan Daemen* et *Vincent Rijmen*, qui a été retenue pour succéder au DES.
- AES est un sous-ensemble de la proposition initiale *Rijndael* : la taille des blocs est **fixée à 128 bits** (au lieu d'une taille variable multiple de 32 bits et comprise entre 128 bits et 256 bits).
- AES se décline en trois versions :
  - **AES-128** $\Rightarrow$ clé de 128 bits, 10 tours
  - **AES-192** $\Rightarrow$ clé de 192 bits, 12 tours
  - **AES-256** $\Rightarrow$ clé de 256 bits, 14 tours
- L'originalité d'AES est que la plupart des opérations se font dans le corps fini $\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$.

Historique & Description

- AES est un algorithme de **chiffrement symétrique par bloc**. Il a remporté en 2000 le concours lancé par le NIST en 1997.
- Il y avait 15 participants à ce concours et c'est la proposition *Rijndael*, du nom de ses concepteurs belges *Joan Daemen* et *Vincent Rijmen*, qui a été retenue pour succéder au DES.
- AES est un sous-ensemble de la proposition initiale *Rijndael* : la taille des blocs est **fixée à 128 bits** (au lieu d'une taille variable multiple de 32 bits et comprise entre 128 bits et 256 bits).
- AES se décline en trois versions :
  - **AES-128** $\Rightarrow$ clé de 128 bits, 10 tours
  - **AES-192** $\Rightarrow$ clé de 192 bits, 12 tours
  - **AES-256** $\Rightarrow$ clé de 256 bits, 14 tours
- L'originalité d'AES est que la plupart des opérations se font dans le corps fini $\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$.

Représentation d'un bloc

- Chaque bloc de 128 bits est découpé en 16 octets $b_0 b_1 \cdots b_{15}$.

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

Représentation d'un bloc

- Chaque bloc de 128 bits est découpé en 16 octets $b_0 b_1 \cdots b_{15}$.
- Les 16 octets sont ensuite placés de la manière suivante dans une matrice 4×4 $(a_{i,j})_{0 \leq i,j \leq 3}$:

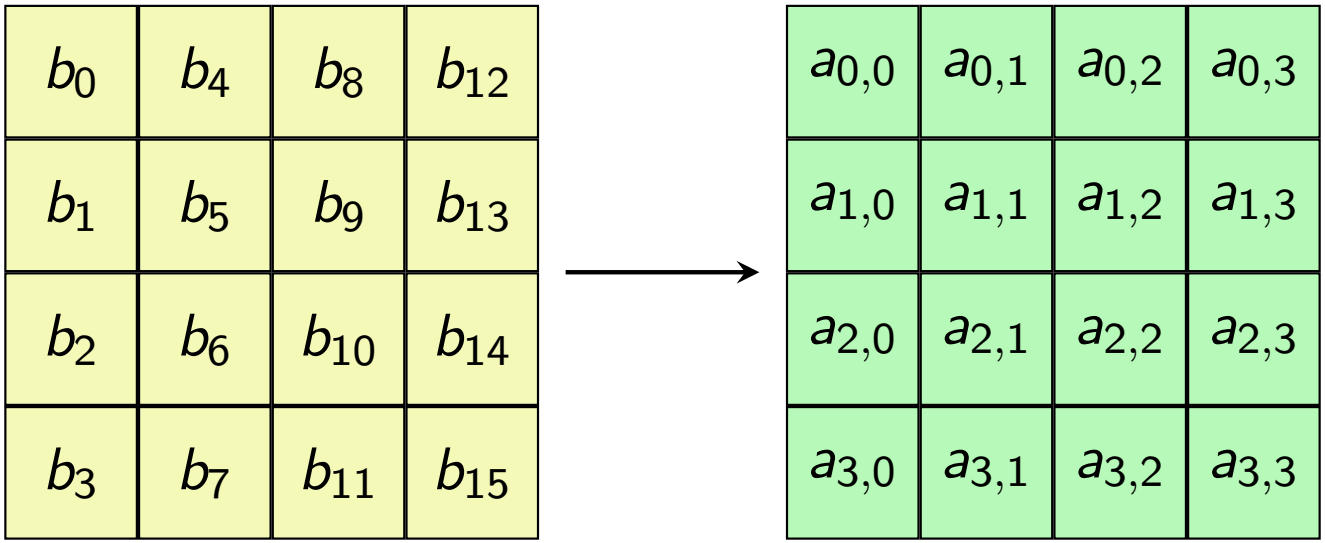


Schéma de fonctionnement d'un tour

A chaque tour, on effectue quatre opérations sur la matrice 4×4 :

Schéma de fonctionnement d'un tour

A chaque tour, on effectue quatre opérations sur la matrice 4×4 :

XOR du bloc entrant avec la sous-clé k_i

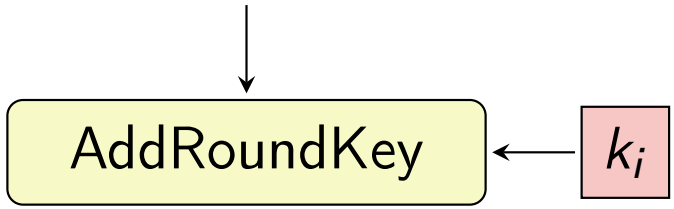


Schéma de fonctionnement d'un tour

A chaque tour, on effectue quatre opérations sur la matrice 4×4 :

XOR du bloc entrant avec la sous-clé k_i

Substitution (*S-Box*) (*confusion*)

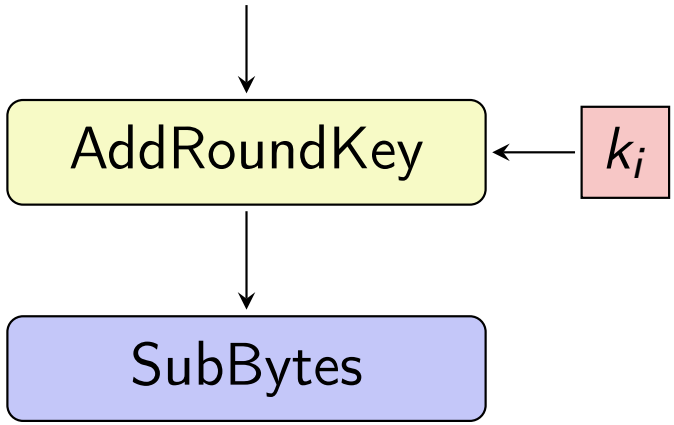


Schéma de fonctionnement d'un tour

A chaque tour, on effectue quatre opérations sur la matrice 4×4 :

XOR du bloc entrant avec la sous-clé k_i

Substitution (*S-Box*) (*confusion*)

Décalage de lignes (*diffusion*)

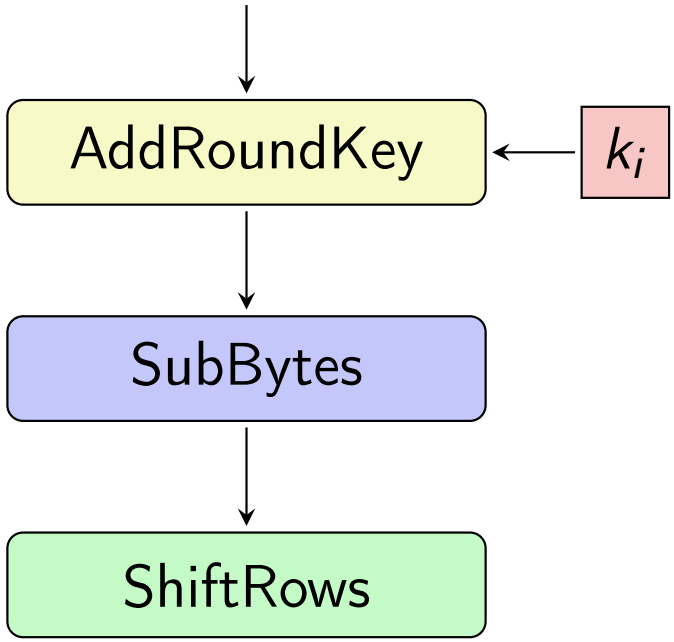


Schéma de fonctionnement d'un tour

A chaque tour, on effectue quatre opérations sur la matrice 4×4 :

XOR du bloc entrant avec la sous-clé k_i

Substitution (*S-Box*) (*confusion*)

Décalage de lignes (*diffusion*)

Mélange des colonnes (*diffusion*)

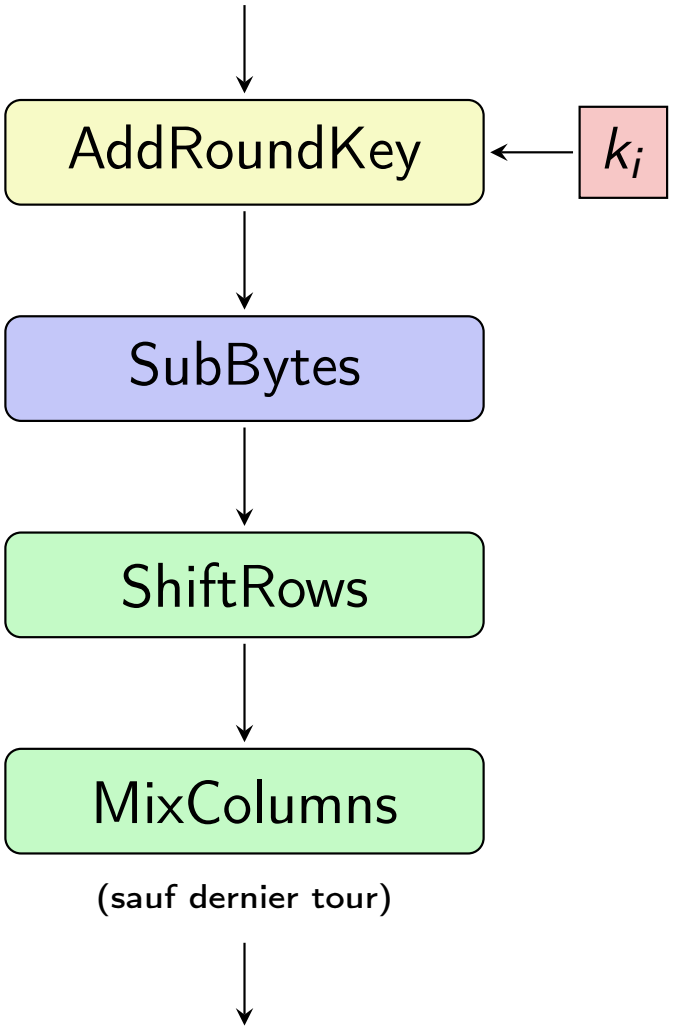


Schéma de fonctionnement d'un tour

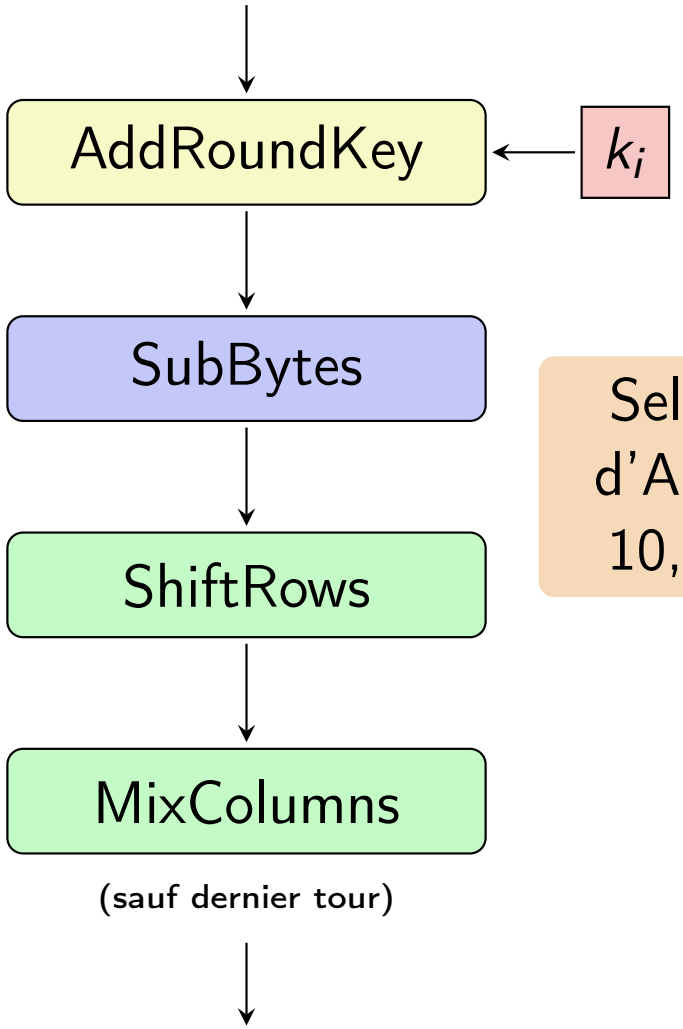
A chaque tour, on effectue quatre opérations sur la matrice 4×4 :

XOR du bloc entrant avec la sous-clé k_i

Substitution (*S-Box*) (*confusion*)

Décalage de lignes (*diffusion*)

Mélange des colonnes (*diffusion*)



Selon les variantes d'AES, on enchaîne 10, 12 ou 14 tours

L'opérateur *AddRoundKey*

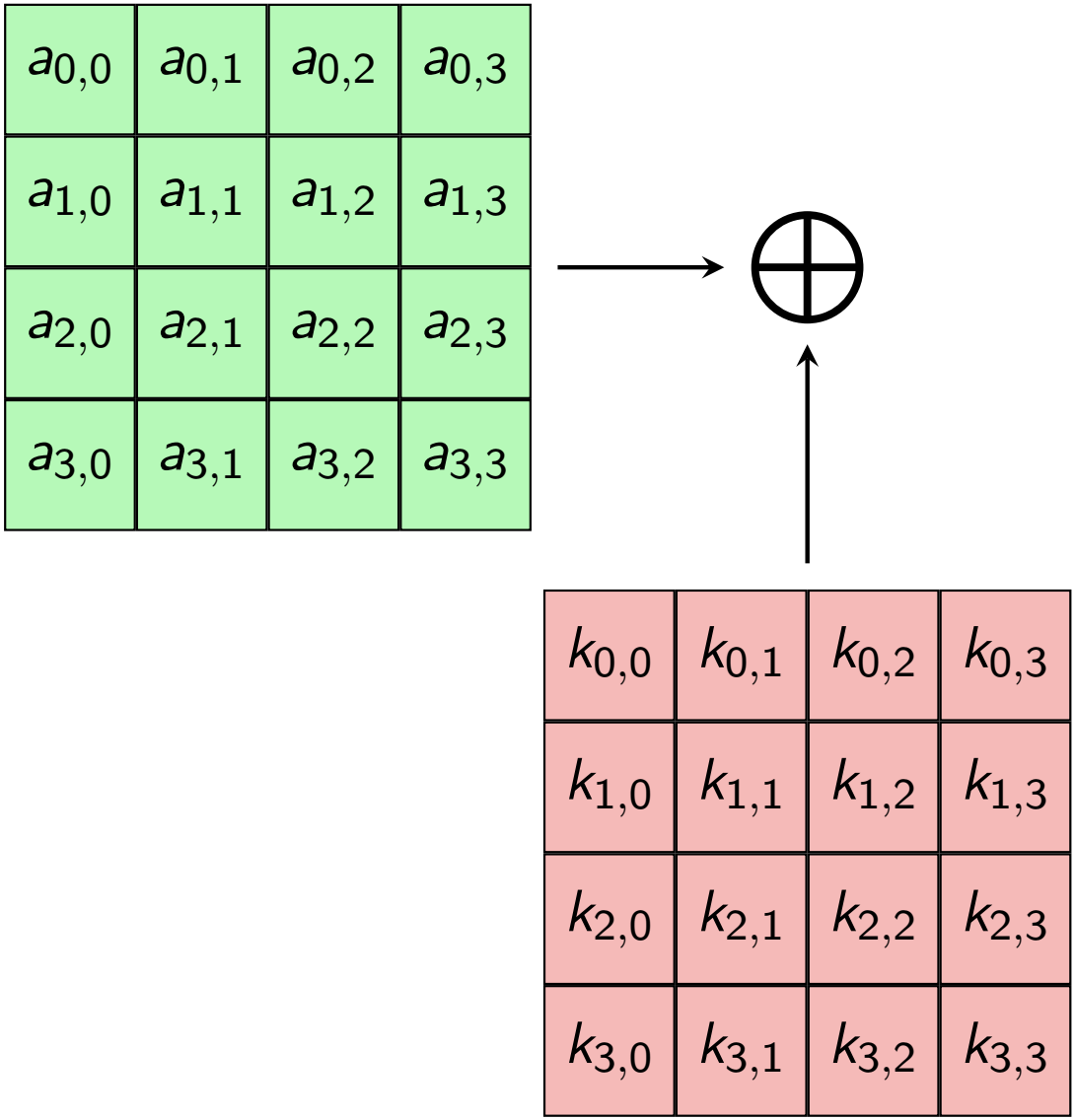
$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

L'opérateur *AddRoundKey*

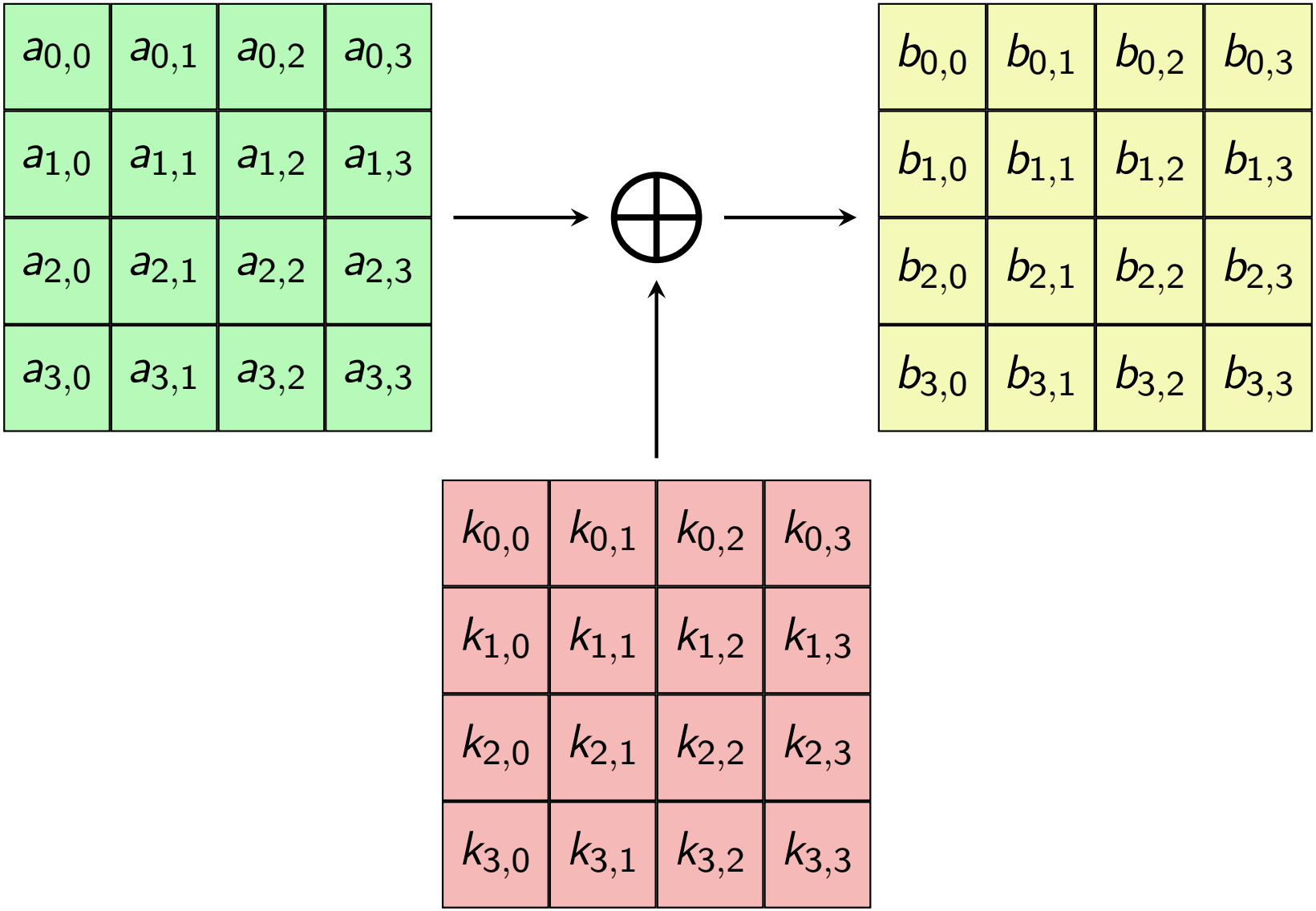
$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

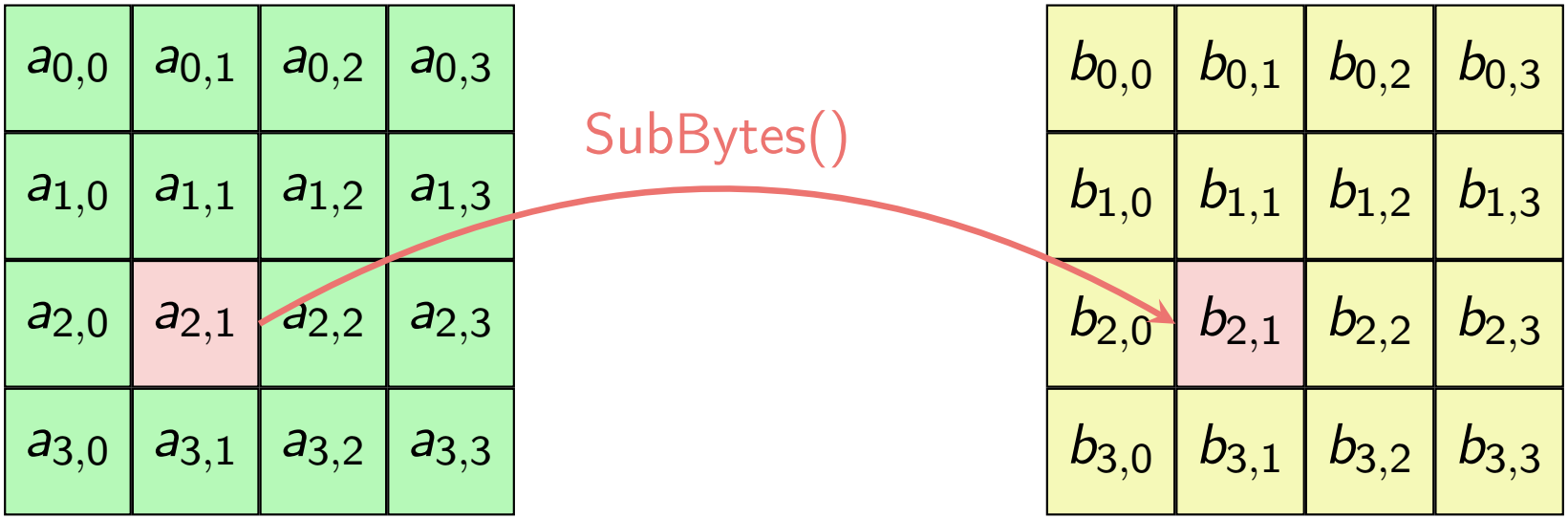
L'opérateur *AddRoundKey*



L'opérateur *AddRoundKey*

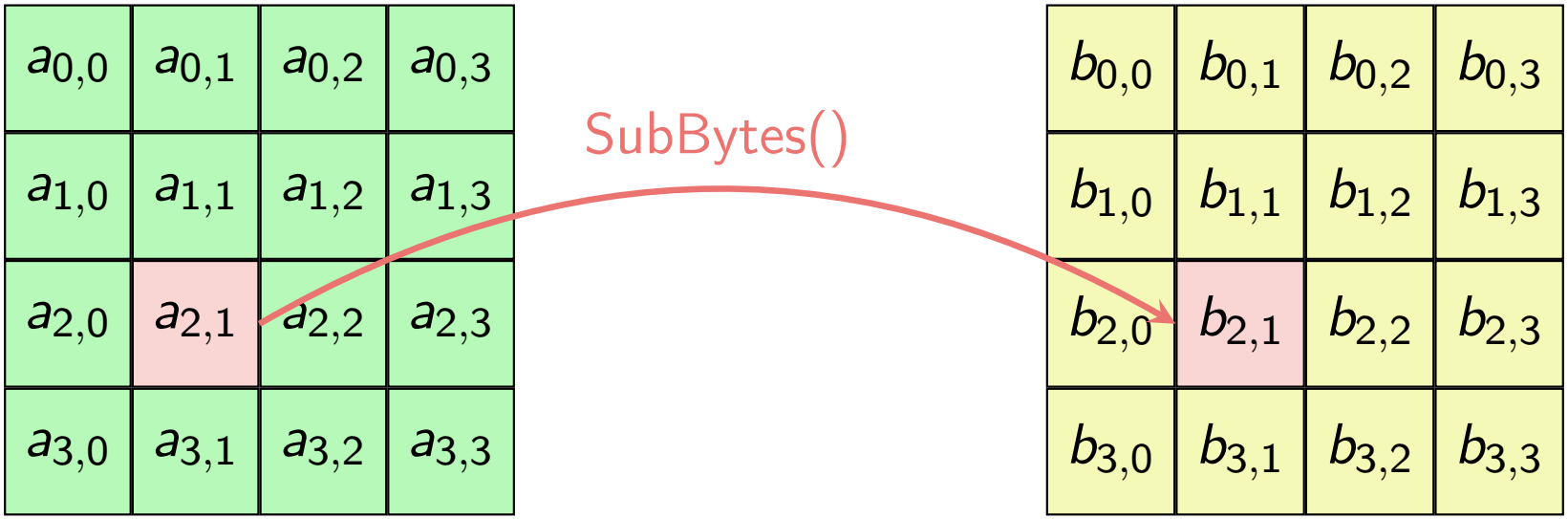


L'opérateur *SubBytes*



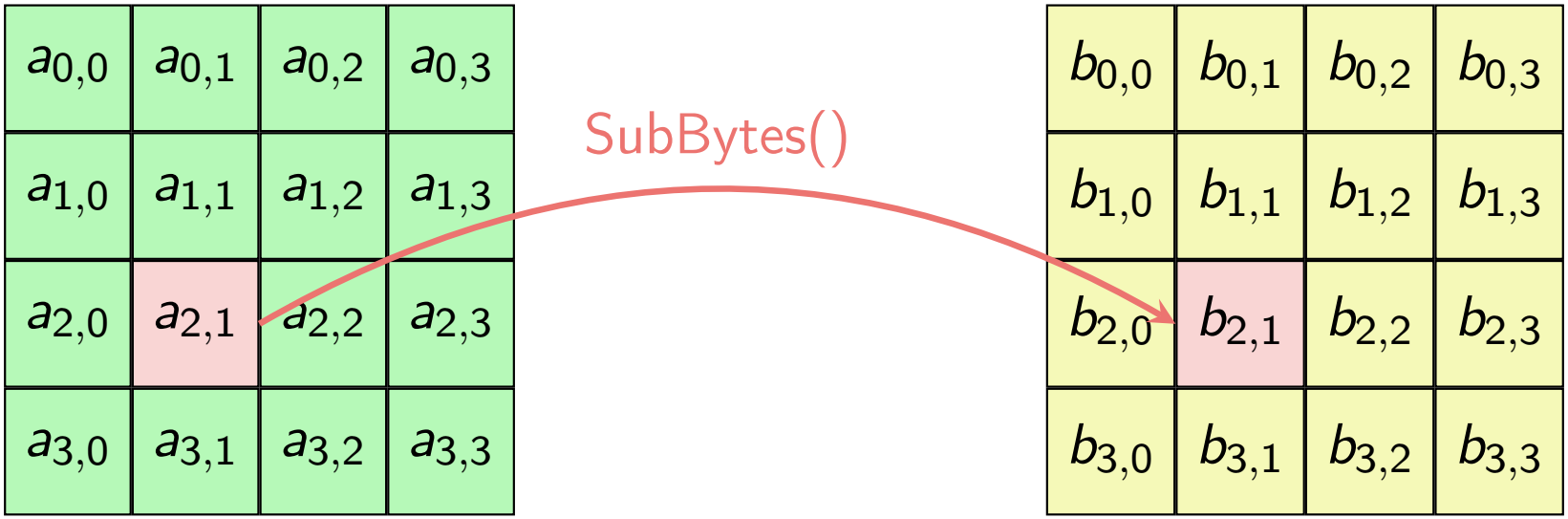
- *SubBytes* opère **indépendamment** sur chaque octet. C'est donc une application de $\{0, \ldots, 255\}$ dans lui-même.
- C'est un opérateur non-linéaire $\Longrightarrow$ *confusion*.

L'opérateur *SubBytes*



- *SubBytes* opère **indépendamment** sur chaque octet. C'est donc une application de $\{0, \ldots, 255\}$ dans lui-même.
- C'est un opérateur non-linéaire $\Longrightarrow$ *confusion*.

L'opérateur *SubBytes*



- *SubBytes* opère **indépendamment** sur chaque octet. C'est donc une application de $\{0, \ldots, 255\}$ dans lui-même.
- C'est un opérateur non-linéaire $\Longrightarrow$ *confusion*.

L'opérateur *SubBytes* (détails)

- Soit $X = (x_7, \ldots, x_0)$ un octet que l'on va considérer comme un élément du corps $\mathbb{F}_{2^8}$. On calcule son inverse $1/X = (y_7, \ldots, y_0)$ si $X \neq 0$. Lorsque X est nul, X reste inchangé.
- On applique ensuite une transformation affine, ce qui donne au final :

$$SubBytes(X) = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

La transformation affine a été choisie de façon à éviter les points fixes.

L'opérateur *SubBytes* (détails)

- Soit $X = (x_7, \ldots, x_0)$ un octet que l'on va considérer comme un élément du corps $\mathbb{F}_{2^8}$. On calcule son inverse $1/X = (y_7, \ldots, y_0)$ si $X \neq 0$. Lorsque X est nul, X reste inchangé.
- On applique ensuite une transformation affine, ce qui donne au final :

$$SubBytes(X) = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

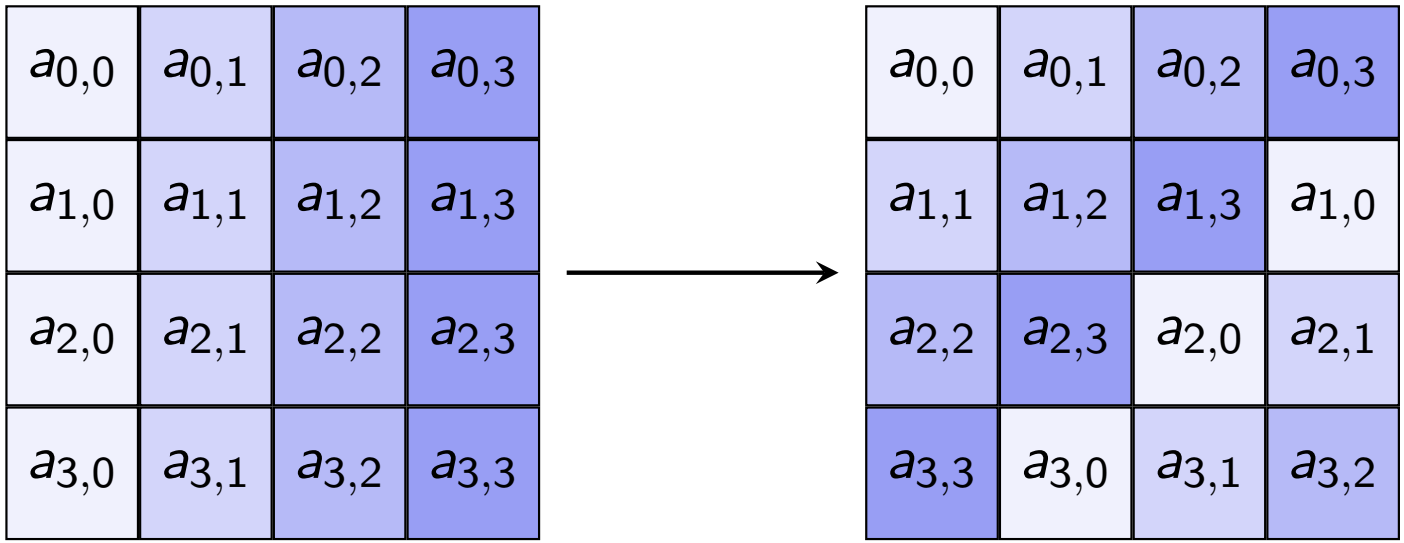
La transformation affine a été choisie de façon à éviter les points fixes.

L'opérateur *ShiftRows*

- Cet opérateur décale cycliquement les lignes d'un bloc. Chaque ligne est décalée d'une valeur différente suivant la taille du bloc sauf la ligne 0 qui reste toujours inchangée. C'est une opération de *diffusion*.
- Dans le cas du chiffrement AES-128, la ligne i est décalée de i octets vers la gauche pour $i = 1, 2, 3$:

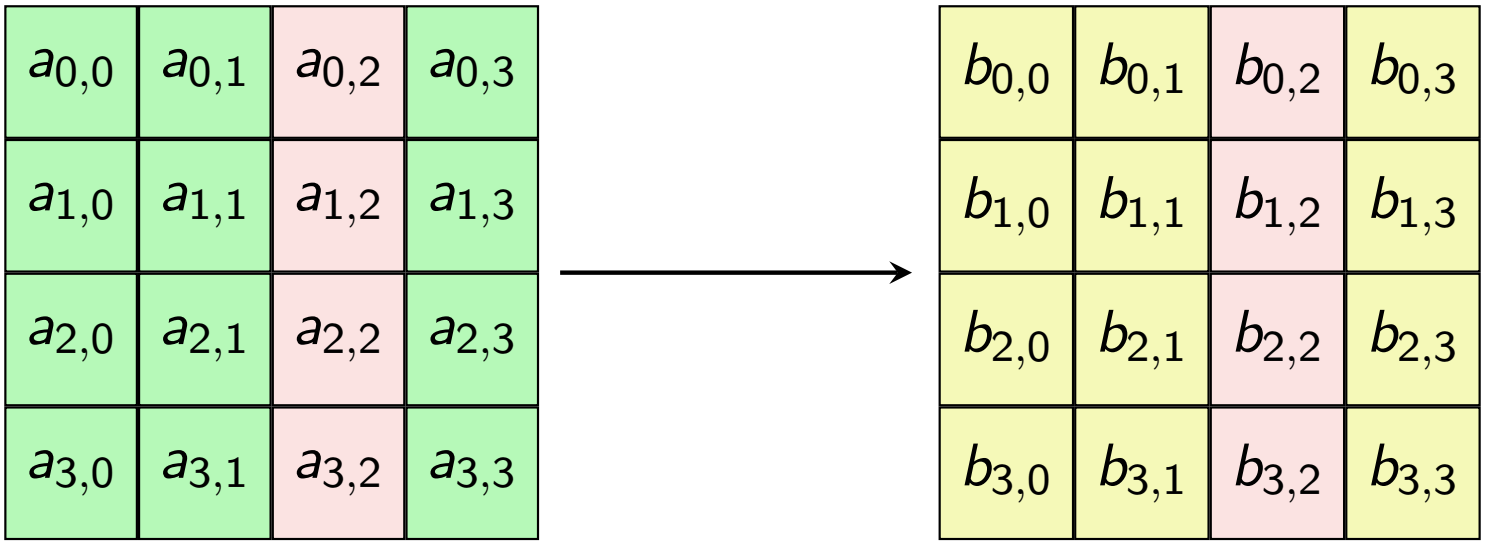
L'opérateur *ShiftRows*

- Cet opérateur décale cycliquement les lignes d'un bloc. Chaque ligne est décalée d'une valeur différente suivant la taille du bloc sauf la ligne 0 qui reste toujours inchangée. C'est une opération de *diffusion*.
- Dans le cas du chiffrement AES-128, la ligne i est décalée de i octets vers la gauche pour $i = 1, 2, 3$:



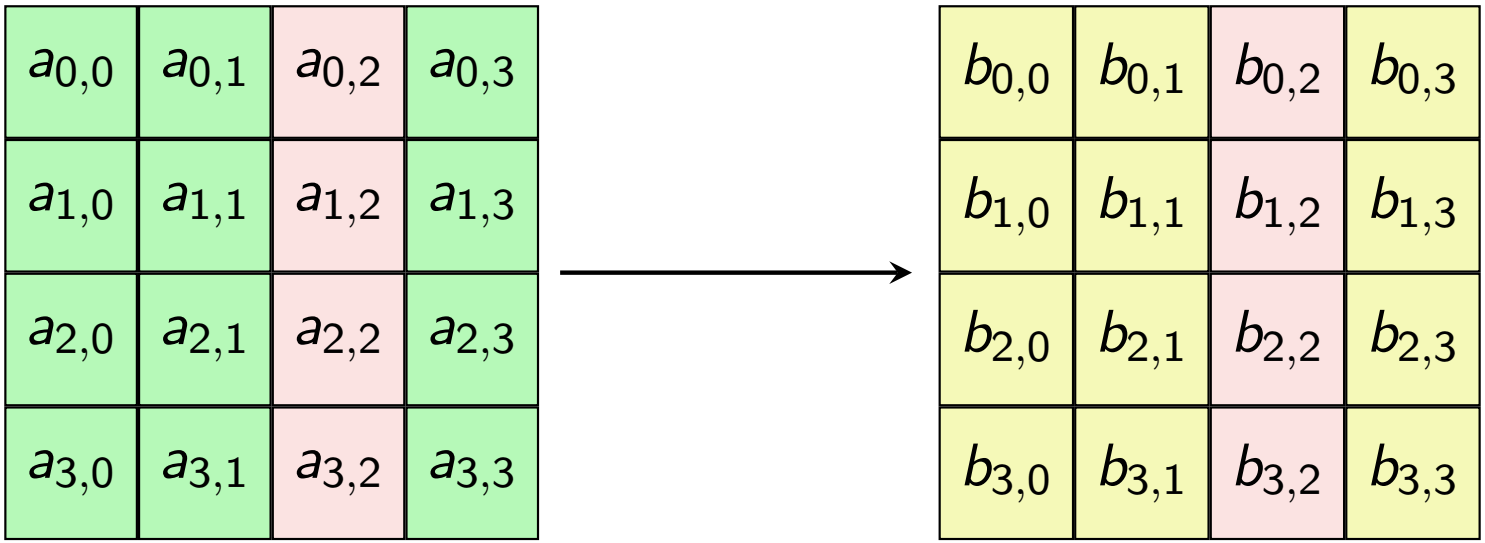
L'opérateur *MixColumns*

Cet opérateur est une transformation linéaire (*diffusion*) agissant sur les colonnes de la matrice 4×4 :



L'opérateur *MixColumns*

Cet opérateur est une transformation linéaire (*diffusion*) agissant sur les colonnes de la matrice 4×4 :



Cette transformation s'écrit (multiplication dans $\mathbb{F}_{2^8}$) :

$$\begin{pmatrix} b_{0,2} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} a_{0,2} \\ a_{1,2} \\ a_{2,2} \\ a_{3,2} \end{pmatrix}$$

L'opérateur *MixColumns* (suite)

- On peut voir aussi cet opérateur comme une multiplication de chaque colonne, considérée comme un polynôme de degré 3 à coefficients dans $\mathbb{F}_{2^8}[X]$, par le polynôme $3X^3 + X^2 + X + 2$, produit ensuite réduit modulo $X^4 + 1$.

L'opérateur *MixColumns* (suite)

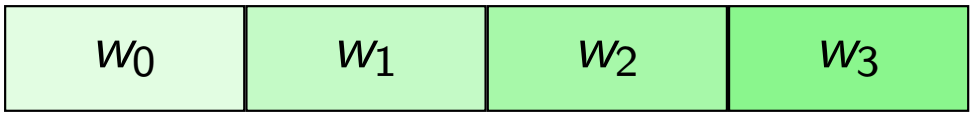
- On peut voir aussi cet opérateur comme une multiplication de chaque colonne, considérée comme un polynôme de degré 3 à coefficients dans $\mathbb{F}_{2^8}[X]$, par le polynôme $3X^3 + X^2 + X + 2$, produit ensuite réduit modulo $X^4 + 1$.
- L'opérateur inverse $MixColumns^{-1}()$:
  (i) *transformation linéaire* :

$$\begin{pmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{pmatrix} = \begin{pmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{pmatrix} \begin{pmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{pmatrix}, \text{ avec } j = 0, 1, 2, 3.$$

  (ii) *multiplication par un polynôme* : l'inverse du polynôme $3X^3 + X^2 + X + 2$ est le polynôme $11X^3 + 13X^2 + 9X + 14$.

Dérivation des sous-clés - AES-128

- On découpe la clé de 128 bits en quatre mots de 32 bits :



- On construit ensuite 10×4 mots supplémentaires (correspondant aux 10 tours d'AES-128) de la manière suivante :
  - (i) $w_i = w_{i-1}^{+} \oplus w_{i-4}$, pour $i = 4, \ldots, 43$,
  - (ii) le terme w_{i-1}^{+} est défini comme suit :

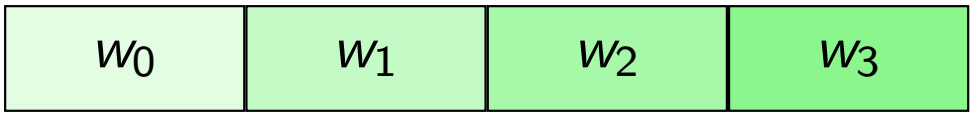
$$w_{i-1}^{+} = \begin{cases} \text{SubBytes}(\text{RotByte}(w_{i-1})) \oplus R[i/4] & \text{si } i \equiv 0 \ (\bmod 4), \\ w_{i-1} & \text{sinon.} \end{cases}$$

  - (iii) SubBytes() est l'opérateur de substitution déjà rencontré. RotByte() est l'opérateur qui décale d'un octet vers la gauche chaque mot :

  - (iv) $R[i] = (X^i, 0, 0, 0)$ où X^i est l'octet correspondant au polynôme X^i du corps $\mathbb{F}_{2^8}$.

Dérivation des sous-clés - AES-128

- On découpe la clé de 128 bits en quatre mots de 32 bits :



- On construit ensuite 10×4 mots supplémentaires (correspondant aux 10 tours d'AES-128) de la manière suivante :
  - (i) $w_i = w_{i-1}^{+} \oplus w_{i-4}$, pour $i = 4, \ldots, 43$,
  - (ii) le terme w_{i-1}^{+} est défini comme suit :

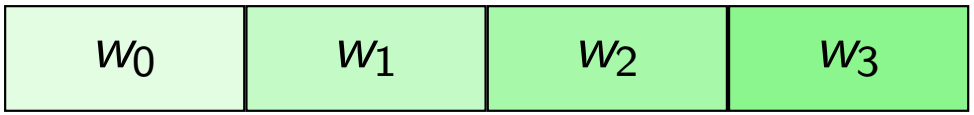
$$w_{i-1}^{+} = \begin{cases} \text{SubBytes}(\text{RotByte}(w_{i-1})) \oplus R[i/4] & \text{si } i \equiv 0 \ (\bmod 4), \\ w_{i-1} & \text{sinon.} \end{cases}$$

  - (iii) SubBytes() est l'opérateur de substitution déjà rencontré. RotByte() est l'opérateur qui décale d'un octet vers la gauche chaque mot :

  - (iv) $R[i] = (X^i, 0, 0, 0)$ où X^i est l'octet correspondant au polynôme X^i du corps $\mathbb{F}_{2^8}$.

Dérivation des sous-clés - AES-128

- On découpe la clé de 128 bits en quatre mots de 32 bits :



- On construit ensuite 10×4 mots supplémentaires (correspondant aux 10 tours d'AES-128) de la manière suivante :
  - (i) $w_i = w_{i-1}^{+} \oplus w_{i-4}$, pour $i = 4, \ldots, 43$,
  - (ii) le terme w_{i-1}^{+} est défini comme suit :

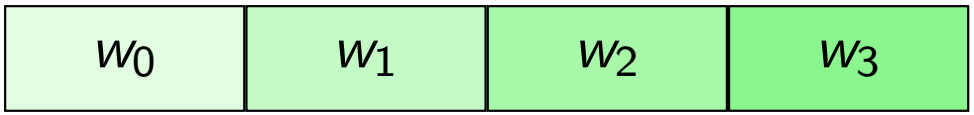
$$w_{i-1}^{+} = \begin{cases} \text{SubBytes}(\text{RotByte}(w_{i-1})) \oplus R[i/4] & \text{si } i \equiv 0 \ (\bmod 4), \\ w_{i-1} & \text{sinon.} \end{cases}$$

  - (iii) SubBytes() est l'opérateur de substitution déjà rencontré. RotByte() est l'opérateur qui décale d'un octet vers la gauche chaque mot :

  - (iv) $R[i] = (X^i, 0, 0, 0)$ où X^i est l'octet correspondant au polynôme X^i du corps $\mathbb{F}_{2^8}$.

Dérivation des sous-clés - AES-128

- On découpe la clé de 128 bits en quatre mots de 32 bits :



- On construit ensuite 10×4 mots supplémentaires (correspondant aux 10 tours d'AES-128) de la manière suivante :
  - (i) $w_i = w_{i-1}^{+} \oplus w_{i-4}$, pour $i = 4, \ldots, 43$,
  - (ii) le terme w_{i-1}^{+} est défini comme suit :

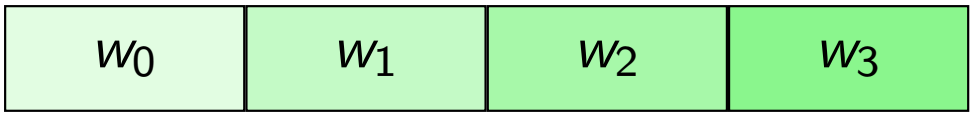
$$w_{i-1}^{+} = \begin{cases} \text{SubBytes}(\text{RotByte}(w_{i-1})) \oplus R[i/4] & \text{si } i \equiv 0 \ (\bmod 4), \\ w_{i-1} & \text{sinon.} \end{cases}$$

  - (iii) SubBytes() est l'opérateur de substitution déjà rencontré. RotByte() est l'opérateur qui décale d'un octet vers la gauche chaque mot :

  - (iv) $R[i] = (X^i, 0, 0, 0)$ où X^i est l'octet correspondant au polynôme X^i du corps $\mathbb{F}_{2^8}$.

Dérivation des sous-clés - AES-128

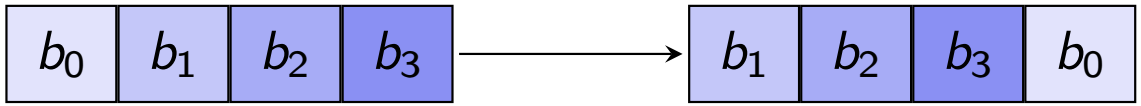
- On découpe la clé de 128 bits en quatre mots de 32 bits :



- On construit ensuite 10×4 mots supplémentaires (correspondant aux 10 tours d'AES-128) de la manière suivante :
  - (i) $w_i = w_{i-1}^{+} \oplus w_{i-4}$, pour $i = 4, \ldots, 43$,
  - (ii) le terme w_{i-1}^{+} est défini comme suit :

$$w_{i-1}^{+} = \begin{cases} \text{SubBytes}(\text{RotByte}(w_{i-1})) \oplus R[i/4] & \text{si } i \equiv 0 \ (\bmod 4), \\ w_{i-1} & \text{sinon.} \end{cases}$$

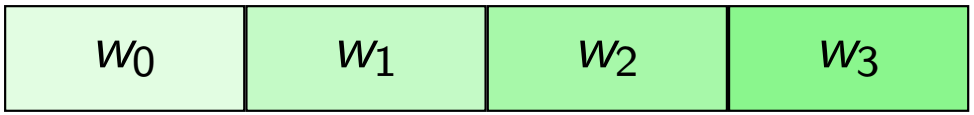
  - (iii) SubBytes() est l'opérateur de substitution déjà rencontré. RotByte() est l'opérateur qui décale d'un octet vers la gauche chaque mot :



  - (iv) $R[i] = (X^i, 0, 0, 0)$ où X^i est l'octet correspondant au polynôme X^i du corps $\mathbb{F}_{2^8}$.

Dérivation des sous-clés - AES-128

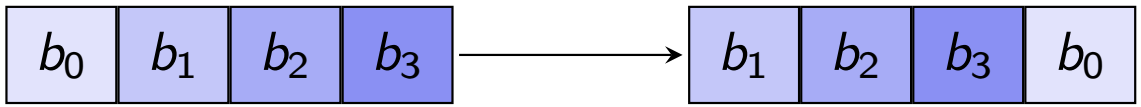
- On découpe la clé de 128 bits en quatre mots de 32 bits :



- On construit ensuite 10×4 mots supplémentaires (correspondant aux 10 tours d'AES-128) de la manière suivante :
  - (i) $w_i = w_{i-1}^{+} \oplus w_{i-4}$, pour $i = 4, \ldots, 43$,
  - (ii) le terme w_{i-1}^{+} est défini comme suit :

$$w_{i-1}^{+} = \begin{cases} \text{SubBytes}(\text{RotByte}(w_{i-1})) \oplus R[i/4] & \text{si } i \equiv 0 \ (\bmod 4), \\ w_{i-1} & \text{sinon.} \end{cases}$$

  - (iii) SubBytes() est l'opérateur de substitution déjà rencontré. RotByte() est l'opérateur qui décale d'un octet vers la gauche chaque mot :



  - (iv) $R[i] = (X^i, 0, 0, 0)$ où X^i est l'octet correspondant au polynôme X^i du corps $\mathbb{F}_{2^8}$.

Performance & Sécurité

- AES-128 est presque aussi rapide que DES. Précisément, AES-128 est 2.7 fois plus rapide que 3DES, lui-même 3 fois plus lent que DES.
- AES est donc **très rapide** et en outre **peu gourmand en mémoire**. Cela lui permet d'être efficace sur une grande variété de matériels.
- AES a été conçu pour résister aux attaques classiques comme la cryptanalyse *linéaire* ou *différentielle*.
- La meilleure **attaque**, due à des chercheurs de Microsoft en 2011, ne permet de gagner que 2 bits, soit 2^{126} opérations au lieu de 2^{128} pour une attaque par force brute. Cette attaque est donc **impraticable**.
- Attention : comme pour tout algorithme de chiffrement, AES n'est pas immunisé contre les attaques par canal auxiliaire (*side channel attack*). Ces attaques exploitent les failles et faiblesses, non pas de l'algorithme, mais de son implémentation logicielle et/ou matérielle.

Performance & Sécurité

- AES-128 est presque aussi rapide que DES. Précisément, AES-128 est 2.7 fois plus rapide que 3DES, lui-même 3 fois plus lent que DES.
- AES est donc **très rapide** et en outre **peu gourmand en mémoire**. Cela lui permet d'être efficace sur une grande variété de matériels.
- AES a été conçu pour résister aux attaques classiques comme la cryptanalyse *linéaire* ou *différentielle*.
- La meilleure **attaque**, due à des chercheurs de Microsoft en 2011, ne permet de gagner que 2 bits, soit 2^{126} opérations au lieu de 2^{128} pour une attaque par force brute. Cette attaque est donc **impraticable**.
- Attention : comme pour tout algorithme de chiffrement, AES n'est pas immunisé contre les attaques par canal auxiliaire (*side channel attack*). Ces attaques exploitent les failles et faiblesses, non pas de l'algorithme, mais de son implémentation logicielle et/ou matérielle.

Performance & Sécurité

- AES-128 est presque aussi rapide que DES. Précisément, AES-128 est 2.7 fois plus rapide que 3DES, lui-même 3 fois plus lent que DES.
- AES est donc **très rapide** et en outre **peu gourmand en mémoire**. Cela lui permet d'être efficace sur une grande variété de matériels.
- AES a été conçu pour résister aux attaques classiques comme la cryptanalyse *linéaire* ou *différentielle*.
- La meilleure **attaque**, due à des chercheurs de Microsoft en 2011, ne permet de gagner que 2 bits, soit 2^{126} opérations au lieu de 2^{128} pour une attaque par force brute. Cette attaque est donc **impraticable**.
- Attention : comme pour tout algorithme de chiffrement, AES n'est pas immunisé contre les attaques par canal auxiliaire (*side channel attack*). Ces attaques exploitent les failles et faiblesses, non pas de l'algorithme, mais de son implémentation logicielle et/ou matérielle.

Performance & Sécurité

- AES-128 est presque aussi rapide que DES. Précisément, AES-128 est 2.7 fois plus rapide que 3DES, lui-même 3 fois plus lent que DES.
- AES est donc **très rapide** et en outre **peu gourmand en mémoire**. Cela lui permet d'être efficace sur une grande variété de matériels.
- AES a été conçu pour résister aux attaques classiques comme la cryptanalyse *linéaire* ou *différentielle*.
- La meilleure **attaque**, due à des chercheurs de Microsoft en 2011, ne permet de gagner que 2 bits, soit 2^{126} opérations au lieu de 2^{128} pour une attaque par force brute. Cette attaque est donc **impraticable**.
- Attention : comme pour tout algorithme de chiffrement, AES n'est pas immunisé contre les attaques par canal auxiliaire (*side channel attack*). Ces attaques exploitent les failles et faiblesses, non pas de l'algorithme, mais de son implémentation logicielle et/ou matérielle.

Performance & Sécurité

- AES-128 est presque aussi rapide que DES. Précisément, AES-128 est 2.7 fois plus rapide que 3DES, lui-même 3 fois plus lent que DES.
- AES est donc **très rapide** et en outre **peu gourmand en mémoire**. Cela lui permet d'être efficace sur une grande variété de matériels.
- AES a été conçu pour résister aux attaques classiques comme la cryptanalyse *linéaire* ou *différentielle*.
- La meilleure **attaque**, due à des chercheurs de Microsoft en 2011, ne permet de gagner que 2 bits, soit 2^{126} opérations au lieu de 2^{128} pour une attaque par force brute. Cette attaque est donc **impraticable**.
- Attention : comme pour tout algorithme de chiffrement, AES n'est pas immunisé contre les attaques par canal auxiliaire (*side channel attack*). Ces attaques exploitent les failles et faiblesses, non pas de l'algorithme, mais de son implémentation logicielle et/ou matérielle.

AES-NI (New Instructions)

- En 2008, *Intel* a proposé une extension du jeu d'instructions de l'architecture *x86* afin d'accélérer les opérations de chiffrement et de déchiffrement d'AES.
- Outre *Intel* et *AMD*, cette extension est disponible sur d'autres architectures, comme par exemple *ARM* et *SPARC*.
- Le gain en performance est très significatif ($\times 5$, en terme de débit n'est pas rare).
- La liste des nouvelles instructions :

AESENC	Effectue un tour de chiffrement
AESENCLAST	Effectue le dernier tour de chiffrement
AESDEC	Effectue un tour de déchiffrement
AESDECLAST	Effectue le dernier tour de déchiffrement
AESKEYGENASSIST	Aide à la génération des sous-clés
AESIMC	Aide à l'opération $MixColumns^{-1}()$
PCLMULQDQ	Multiplication *Carry-Less*

AES-NI (New Instructions)

- En 2008, *Intel* a proposé une extension du jeu d'instructions de l'architecture *x86* afin d'accélérer les opérations de chiffrement et de déchiffrement d'AES.
- Outre *Intel* et *AMD*, cette extension est disponible sur d'autres architectures, comme par exemple *ARM* et *SPARC*.
- Le gain en performance est très significatif ($\times 5$, en terme de débit n'est pas rare).
- La liste des nouvelles instructions :

AESENC	Effectue un tour de chiffrement
AESENCLAST	Effectue le dernier tour de chiffrement
AESDEC	Effectue un tour de déchiffrement
AESDECLAST	Effectue le dernier tour de déchiffrement
AESKEYGENASSIST	Aide à la génération des sous-clés
AESIMC	Aide à l'opération $MixColumns^{-1}()$
PCLMULQDQ	Multiplication *Carry-Less*

AES-NI (New Instructions)

- En 2008, *Intel* a proposé une extension du jeu d'instructions de l'architecture *x86* afin d'accélérer les opérations de chiffrement et de déchiffrement d'AES.
- Outre *Intel* et *AMD*, cette extension est disponible sur d'autres architectures, comme par exemple *ARM* et *SPARC*.
- Le gain en performance est très significatif ($\times 5$, en terme de débit n'est pas rare).
- La liste des nouvelles instructions :

AESENC	Effectue un tour de chiffrement
AESENCLAST	Effectue le dernier tour de chiffrement
AESDEC	Effectue un tour de déchiffrement
AESDECLAST	Effectue le dernier tour de déchiffrement
AESKEYGENASSIST	Aide à la génération des sous-clés
AESIMC	Aide à l'opération $MixColumns^{-1}()$
PCLMULQDQ	Multiplication *Carry-Less*

AES-NI (New Instructions)

- En 2008, *Intel* a proposé une extension du jeu d'instructions de l'architecture *x86* afin d'accélérer les opérations de chiffrement et de déchiffrement d'AES.
- Outre *Intel* et *AMD*, cette extension est disponible sur d'autres architectures, comme par exemple *ARM* et *SPARC*.
- Le gain en performance est très significatif ($\times 5$, en terme de débit n'est pas rare).
- La liste des nouvelles instructions :

AESENC	Effectue un tour de chiffrement
AESENCLAST	Effectue le dernier tour de chiffrement
AESDEC	Effectue un tour de déchiffrement
AESDECLAST	Effectue le dernier tour de déchiffrement
AESKEYGENASSIST	Aide à la génération des sous-clés
AESIMC	Aide à l'opération $MixColumns^{-1}()$
PCLMULQDQ	Multiplication *Carry-Less*

AES-NI : Linux & OpenSSL

- Est-ce que AES-NI est supporté par mon système Linux ?

```
$ cpuid | grep -i aes | sort | uniq
AES instruction   = true
```

- Le driver AES-NI est-il chargé ?

```
$ sort -u /proc/crypto | grep module
module          : aesni_intel
module          : aes_x86_64
module          : blowfish_generic
.................................
```

- AES-NI est-il disponible sous OpenSSL ?

```
$ openssl engine
(rsax) RSAX engine support
(aesni) Intel AES-NI engine
(rdrand) Intel RDRAND engine
(dynamic) Dynamic engine loading support
.................................
```

AES-NI : Linux & OpenSSL

- Est-ce que AES-NI est supporté par mon système Linux ?

```
$ cpuid | grep -i aes | sort | uniq
AES instruction  = true
```

- Le driver AES-NI est-il chargé ?

```
$ sort -u /proc/crypto | grep module
module          : aesni_intel
module          : aes_x86_64
module          : blowfish_generic
.................................
```

- AES-NI est-il disponible sous OpenSSL ?

```
$ openssl engine
(rsax) RSAX engine support
(aesni) Intel AES-NI engine
(rdrand) Intel RDRAND engine
(dynamic) Dynamic engine loading support
.................................
```

AES-NI : Linux & OpenSSL

- Est-ce que AES-NI est supporté par mon système Linux ?

```
$ cpuid | grep -i aes | sort | uniq
AES instruction  = true
```

- Le driver AES-NI est-il chargé ?

```
$ sort -u /proc/crypto | grep module
module          : aesni_intel
module          : aes_x86_64
module          : blowfish_generic
.................................
```

- AES-NI est-il disponible sous OpenSSL ?

```
$ openssl engine
(rsax) RSAX engine support
(aesni) Intel AES-NI engine
(rdrand) Intel RDRAND engine
(dynamic) Dynamic engine loading support
.................................
```

AES-NI : gcc

- Le type intrinsèque utilisé est un mot de 128 bits : __m128i :

```
#include <wmmintrin.h>
char msg_in[16] = "Un beau message";
__m128i value;
char msg_out[16];

value = _mm_load_si128((__m128i *) msg_in);
_mm_store_si128((__m128i *) msg_out, value);
```

- Le chiffrement AES-128 d'un bloc :

```
msg_in = _mm_xor_si128(msg_in, K0);
msg_in = _mm_aesenc_si128(msg_in, K1);
.....................................
msg_in = _mm_aesenc_si128(msg_in, K9);
msg_in = _mm_aesenclast_si128(msg_in, K10);
_mm_store_si128((__m128i *) msg_out, msg_in);
```

AES-NI : gcc

- Le type intrinsèque utilisé est un mot de 128 bits : __m128i :

```
#include <wmmintrin.h>
char msg_in[16] = "Un beau message";
__m128i value;
char msg_out[16];

value = _mm_load_si128((__m128i *) msg_in);
_mm_store_si128((__m128i *) msg_out, value);
```

- Le chiffrement AES-128 d'un bloc :

```
msg_in = _mm_xor_si128(msg_in, K0);
msg_in = _mm_aesenc_si128(msg_in, K1);
.....................................
msg_in = _mm_aesenc_si128(msg_in, K9);
msg_in = _mm_aesenclast_si128(msg_in, K10);
_mm_store_si128((__m128i *) msg_out, msg_in);
```

Présentation

- Jusqu'à présent, on ne s'est préoccupé que du chiffrement d'un bloc. Or un message à chiffrer comporte généralement plusieurs blocs.
- Un mode opératoire est un algorithme, reposant sur un algorithme de chiffrement par bloc (3DES, AES,...), permettant de chiffrer des **données de taille arbitraire**.
- L'idée naïve, consistant à chiffrer chaque bloc indépendamment (mode *Electronic Codebook* - ECB) n'offre pas, comme nous le verrons, suffisamment de garantie quant à la confidentialité.
- Des dizaines de modes ont été proposés depuis la fin des années 70. Pour n'en citer que quelques uns parmi eux : ECB, *Cipher Block Chaining* (CBC), *Cipher Feedback* (CFB), *Output Feedback* (OFB) et *Counter (CTR)*, ce dernier étant dû à Diffie-Hellman (1979).
- Pour en savoir plus, le document du NIST (*Recommendation for Block Cipher Modes of Operation*) : ▸ SP800-38A

Présentation

- Jusqu'à présent, on ne s'est préoccupé que du chiffrement d'un bloc. Or un message à chiffrer comporte généralement plusieurs blocs.
- Un mode opératoire est un algorithme, reposant sur un algorithme de chiffrement par bloc (3DES, AES,...), permettant de chiffrer des **données de taille arbitraire**.
- L'idée naïve, consistant à chiffrer chaque bloc indépendamment (mode *Electronic Codebook* - ECB) n'offre pas, comme nous le verrons, suffisamment de garantie quant à la confidentialité.
- Des dizaines de modes ont été proposés depuis la fin des années 70. Pour n'en citer que quelques uns parmi eux : ECB, *Cipher Block Chaining* (CBC), *Cipher Feedback* (CFB), *Output Feedback* (OFB) et *Counter (CTR)*, ce dernier étant dû à Diffie-Hellman (1979).
- Pour en savoir plus, le document du NIST (*Recommendation for Block Cipher Modes of Operation*) : ▸ SP800-38A

Présentation

- Jusqu'à présent, on ne s'est préoccupé que du chiffrement d'un bloc. Or un message à chiffrer comporte généralement plusieurs blocs.
- Un mode opératoire est un algorithme, reposant sur un algorithme de chiffrement par bloc (3DES, AES,...), permettant de chiffrer des **données de taille arbitraire**.
- L'idée naïve, consistant à chiffrer chaque bloc indépendamment (mode *Electronic Codebook* - ECB) n'offre pas, comme nous le verrons, suffisamment de garantie quant à la confidentialité.
- Des dizaines de modes ont été proposés depuis la fin des années 70. Pour n'en citer que quelques uns parmi eux : ECB, *Cipher Block Chaining* (CBC), *Cipher Feedback* (CFB), *Output Feedback* (OFB) et *Counter (CTR)*, ce dernier étant dû à Diffie-Hellman (1979).
- Pour en savoir plus, le document du NIST (*Recommendation for Block Cipher Modes of Operation*) : ▸ SP800-38A

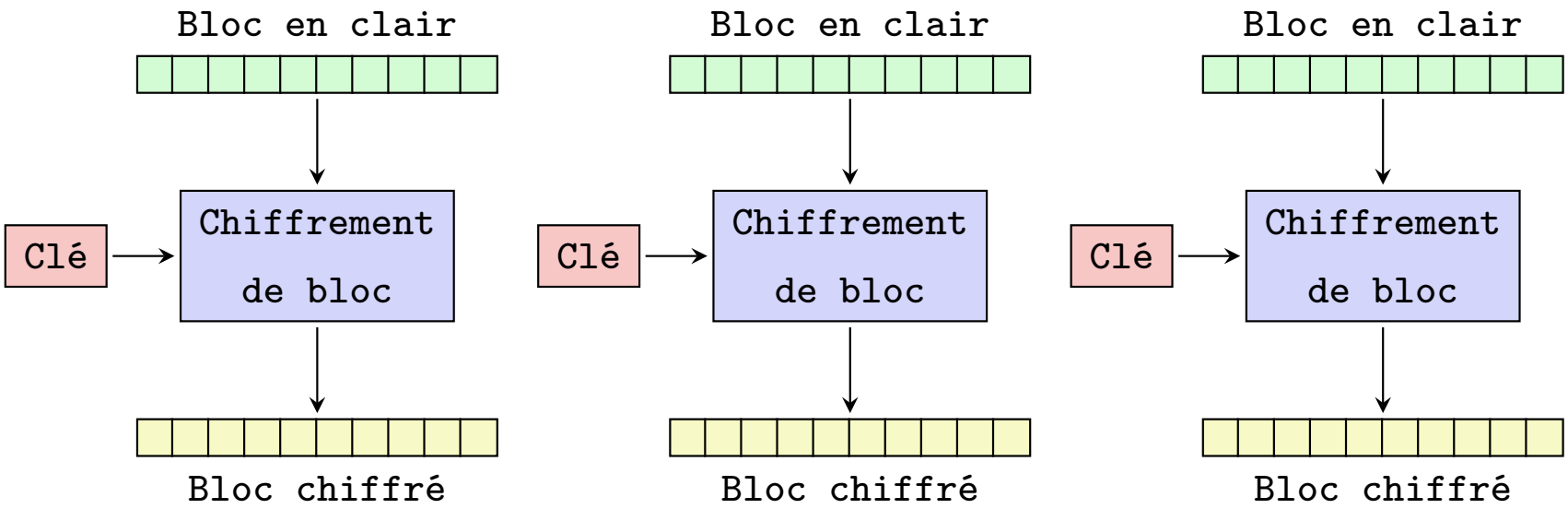
Présentation

- Jusqu'à présent, on ne s'est préoccupé que du chiffrement d'un bloc. Or un message à chiffrer comporte généralement plusieurs blocs.
- Un mode opératoire est un algorithme, reposant sur un algorithme de chiffrement par bloc (3DES, AES,...), permettant de chiffrer des **données de taille arbitraire**.
- L'idée naïve, consistant à chiffrer chaque bloc indépendamment (mode *Electronic Codebook* - ECB) n'offre pas, comme nous le verrons, suffisamment de garantie quant à la confidentialité.
- Des dizaines de modes ont été proposés depuis la fin des années 70. Pour n'en citer que quelques uns parmi eux : ECB, *Cipher Block Chaining* (CBC), *Cipher Feedback* (CFB), *Output Feedback* (OFB) et *Counter (CTR)*, ce dernier étant dû à Diffie-Hellman (1979).
- Pour en savoir plus, le document du NIST (*Recommendation for Block Cipher Modes of Operation*) : ▸ SP800-38A

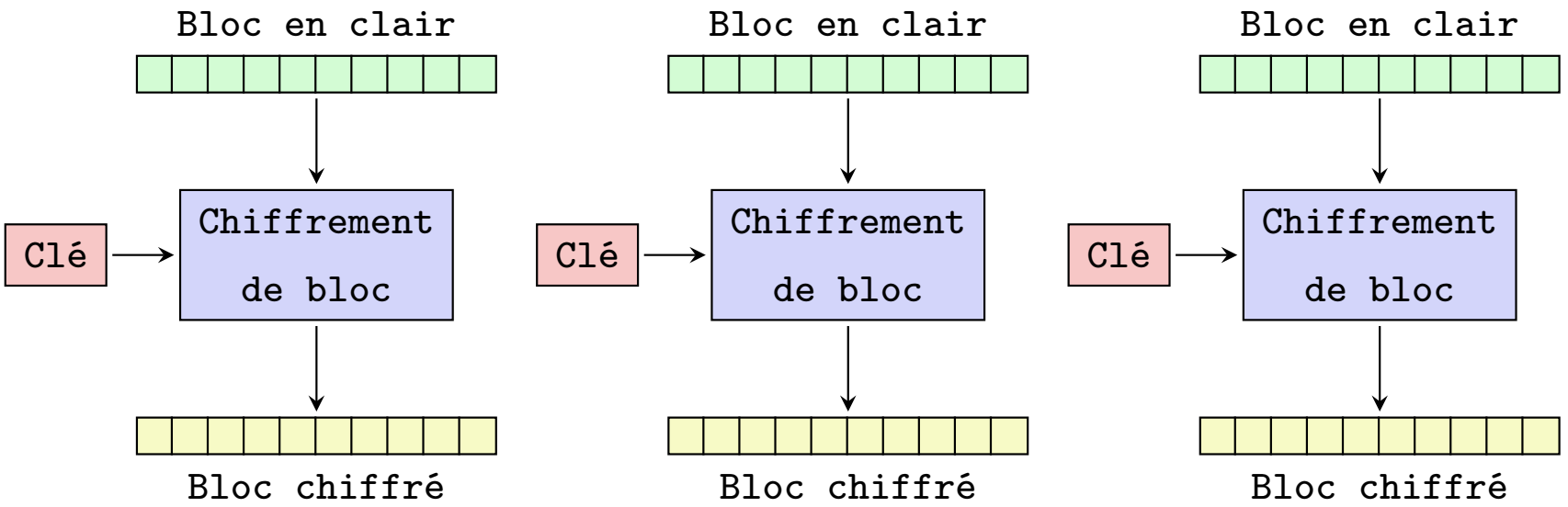
Présentation

- Jusqu'à présent, on ne s'est préoccupé que du chiffrement d'un bloc. Or un message à chiffrer comporte généralement plusieurs blocs.
- Un mode opératoire est un algorithme, reposant sur un algorithme de chiffrement par bloc (3DES, AES,...), permettant de chiffrer des **données de taille arbitraire**.
- L'idée naïve, consistant à chiffrer chaque bloc indépendamment (mode *Electronic Codebook* - ECB) n'offre pas, comme nous le verrons, suffisamment de garantie quant à la confidentialité.
- Des dizaines de modes ont été proposés depuis la fin des années 70. Pour n'en citer que quelques uns parmi eux : ECB, *Cipher Block Chaining* (CBC), *Cipher Feedback* (CFB), *Output Feedback* (OFB) et *Counter (CTR)*, ce dernier étant dû à Diffie-Hellman (1979).
- Pour en savoir plus, le document du NIST (*Recommendation for Block Cipher Modes of Operation*) : ▸ SP800-38A

Mode *Electronic Codebook* (ECB) - Chiffrement

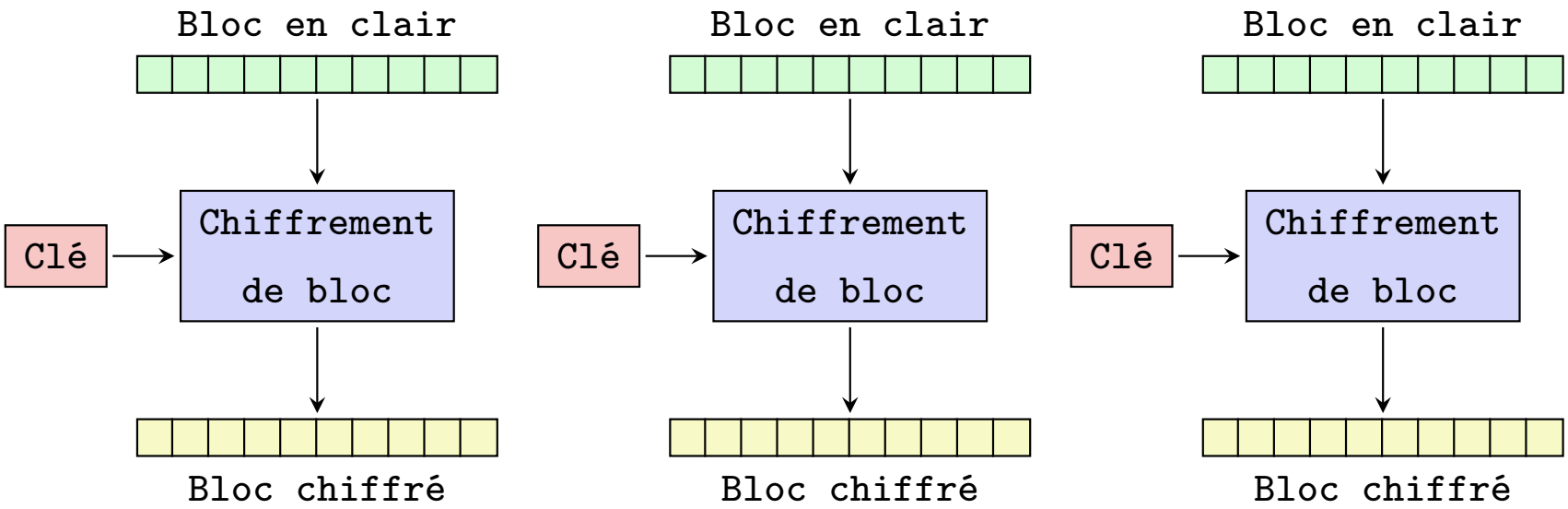


Mode *Electronic Codebook* (ECB) - Chiffrement



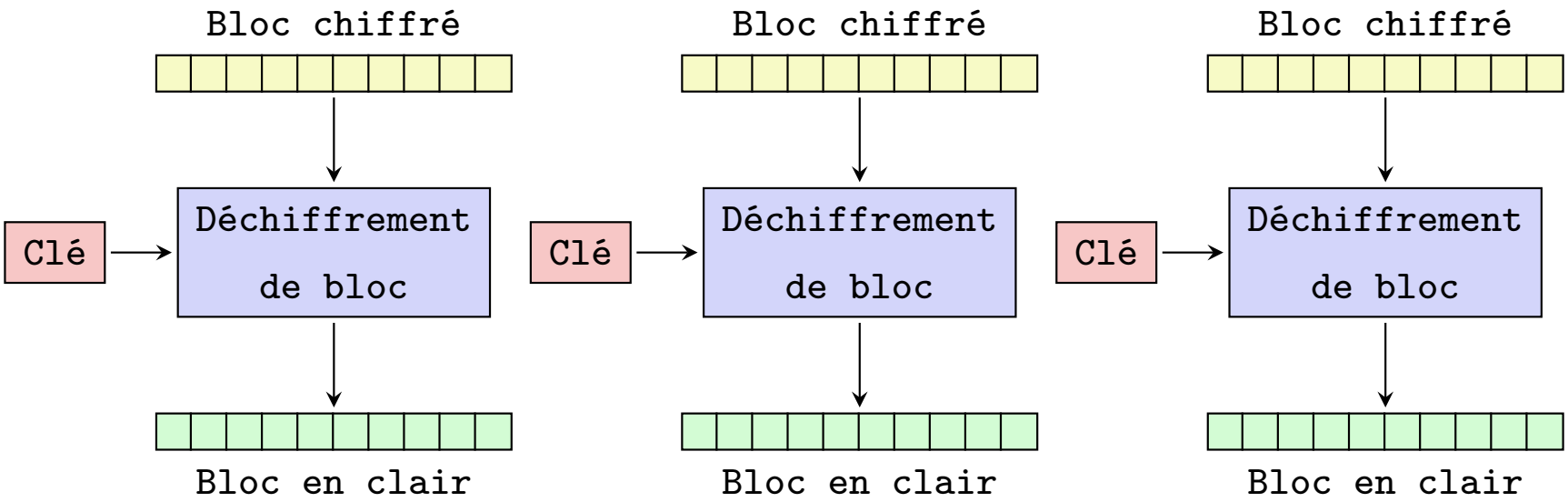
- Chaque bloc est (dé)chiffré **indépendamment** des autres blocs.
  Avantage : le (dé)chiffrement est parallélisable.

Mode *Electronic Codebook* (ECB) - Chiffrement

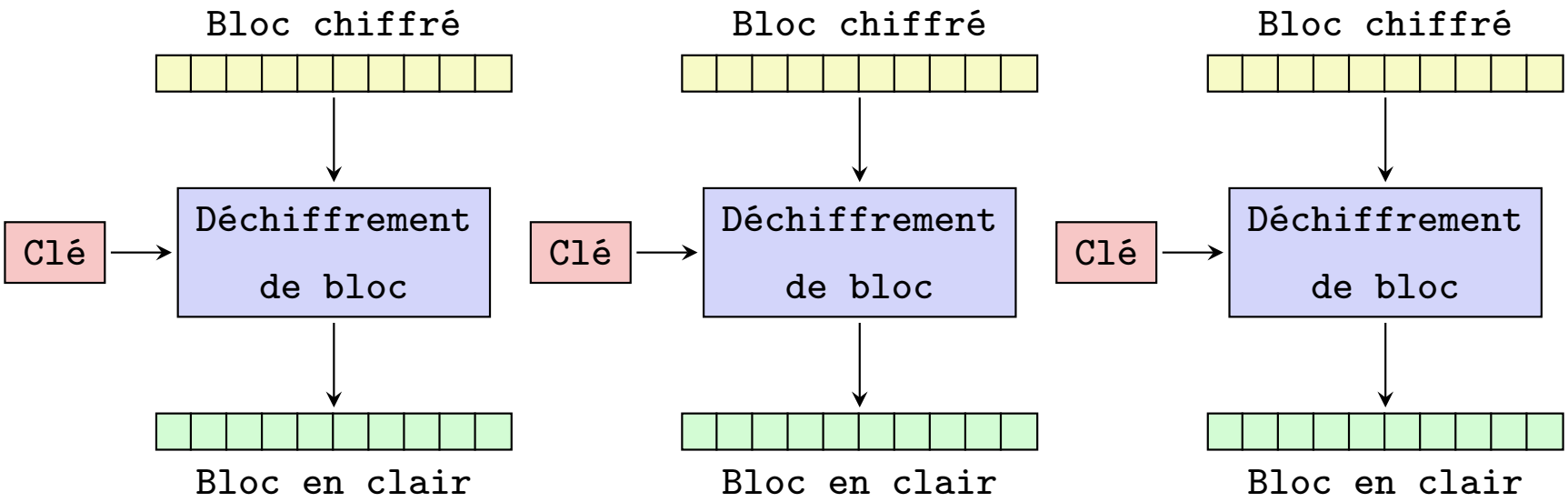


- Chaque bloc est (dé)chiffré **indépendamment** des autres blocs. Avantage : le (dé)chiffrement est parallélisable.
- Inconvénient majeur : deux blocs en clair identiques seront **chiffrés de manière identique**. Fortement déconseillé pour les applications cryptographiques.

Mode *Electronic Codebook* (ECB) - Déchiffrement

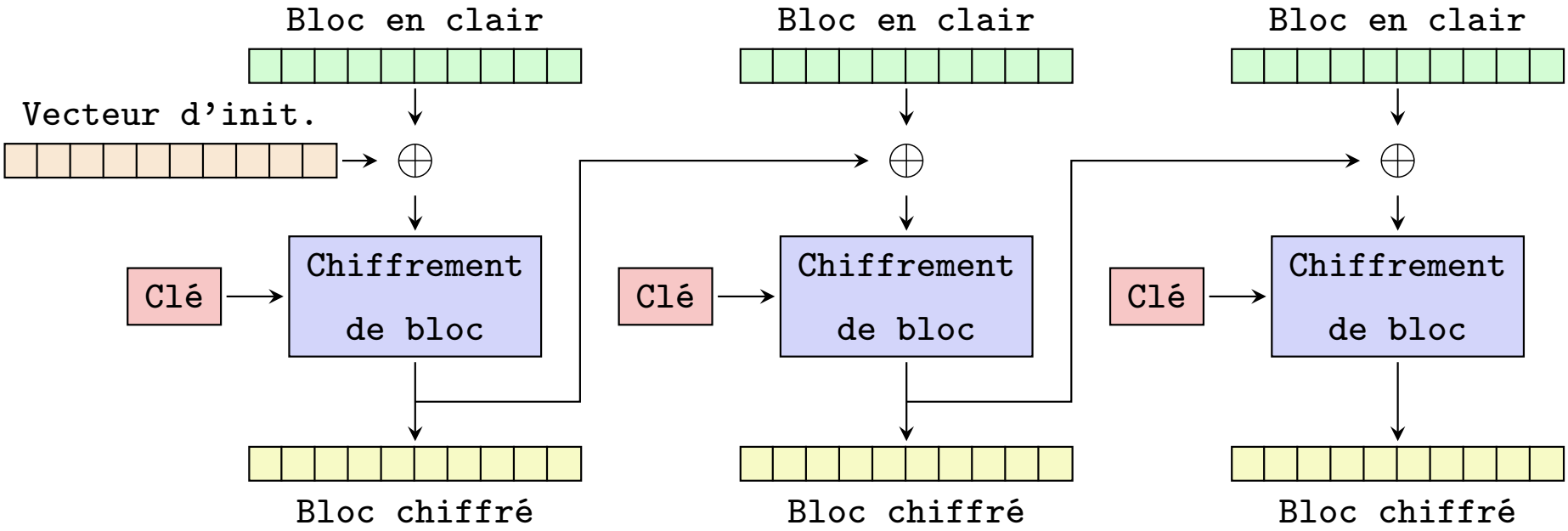


Mode *Electronic Codebook* (ECB) - Déchiffrement

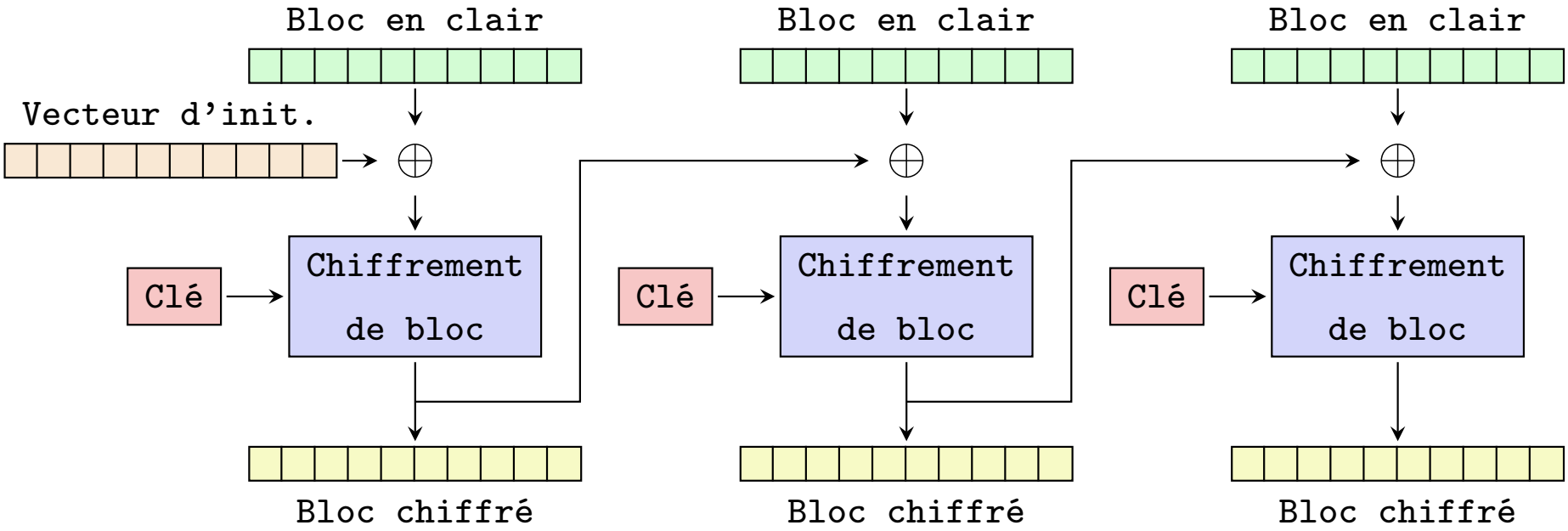


- Le déchiffrement, en ce qui concerne le mode opératoire, est identique au chiffrement.

Mode *Cipher Block Chaining* (CBC) - Chiffrement



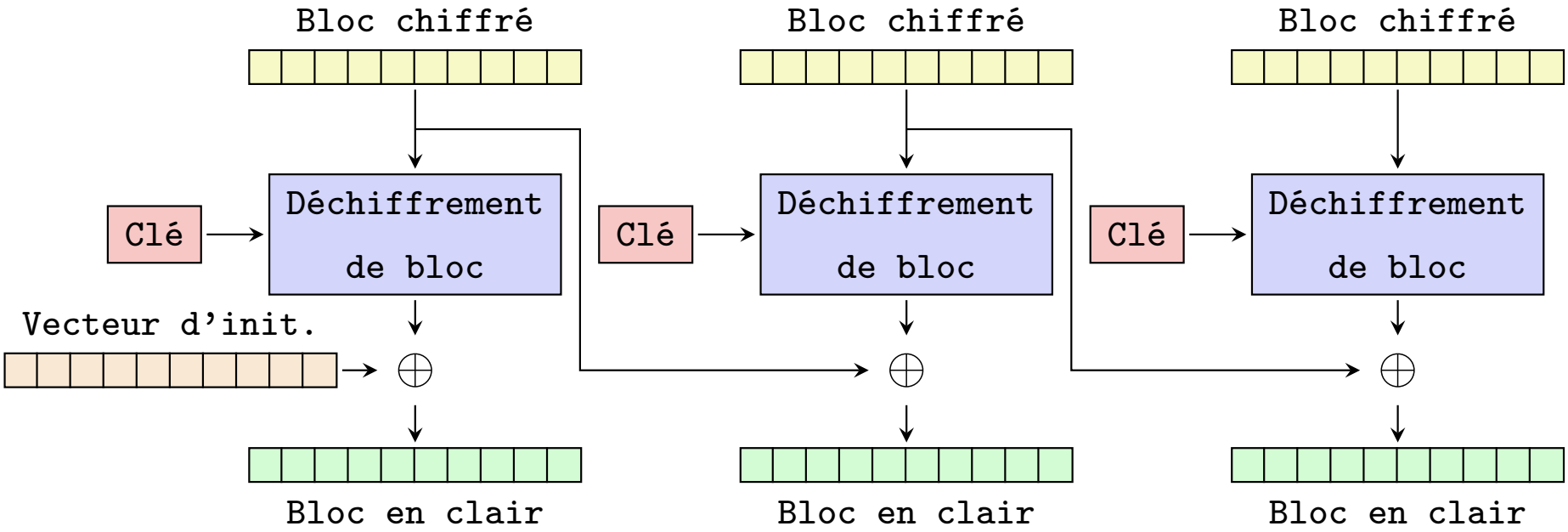
Mode *Cipher Block Chaining* (CBC) - Chiffrement



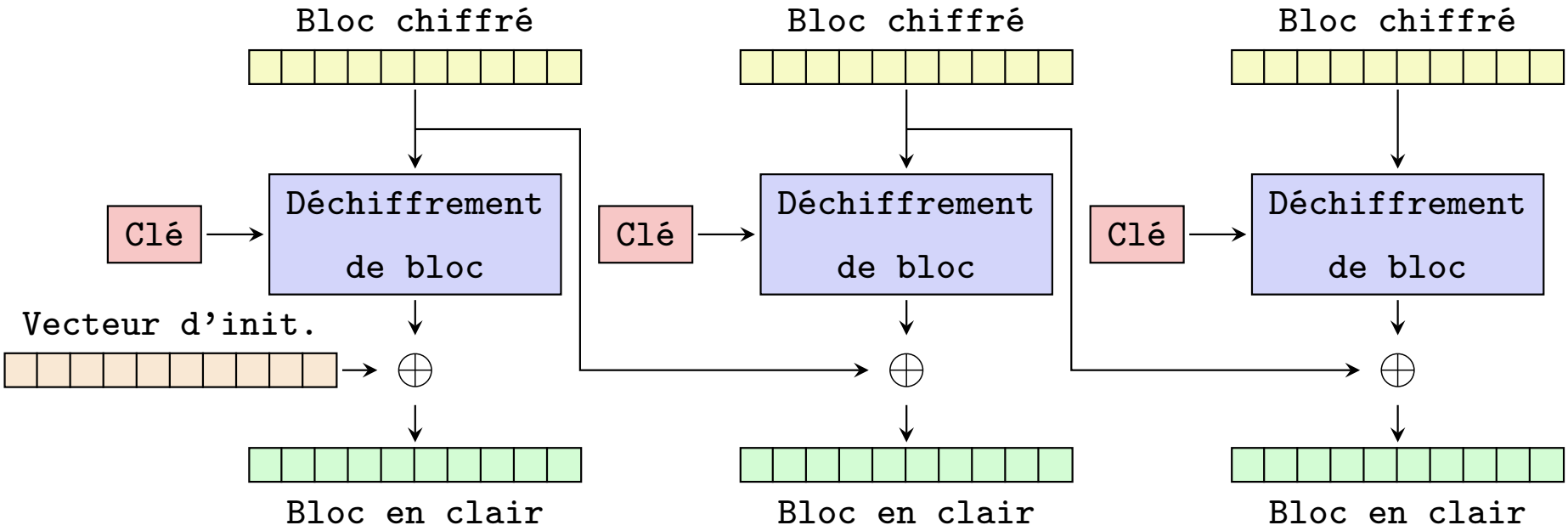
- Si $(B_i)_{i\geq 1}$, $(C_i)_{i\geq 1}$, $\mathcal{E}_K$ et IV désignent respectivement, la suite des blocs en clair, la suite des blocs chiffrés, la fonction de chiffrement et le vecteur d'initialisation, on a :

$$\begin{cases} C_0 = IV \\ C_i = \mathcal{E}_K(B_i) \oplus C_{i-1}, \text{ pour } i \geq 1. \end{cases}$$

Mode *Cipher Block Chaining* (CBC) - Déchiffrement



Mode *Cipher Block Chaining* (CBC) - Déchiffrement



- Si $(B_i)_{i\geq 1}$, $(C_i)_{i\geq 1}$, $\mathcal{E}_K^{-1}$ et IV désignent respectivement, la suite des blocs en clair, la suite des blocs chiffrés, la fonction de déchiffrement et le vecteur d'initialisation, on a :

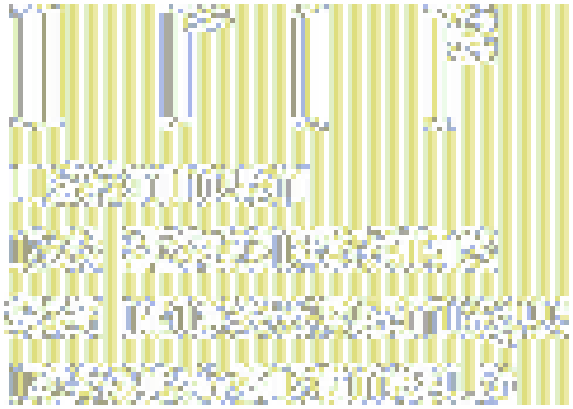
$$\begin{cases} C_0 = IV \\ B_i = \mathcal{E}_K^{-1}(C_i) \oplus C_{i-1}, \text{ pour } i \geq 1. \end{cases}$$

Différents chiffrements du logo de mon laboratoire

logo original

Différents chiffrements du logo de mon laboratoire

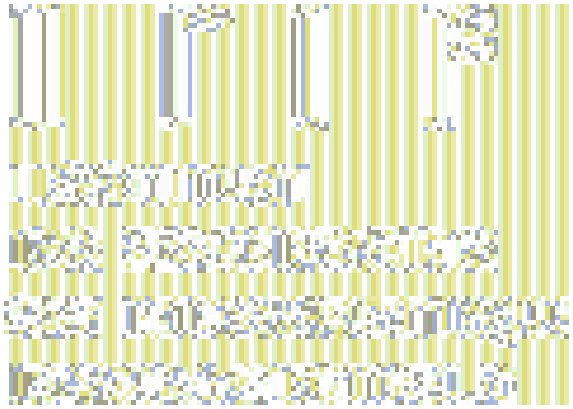
logo original



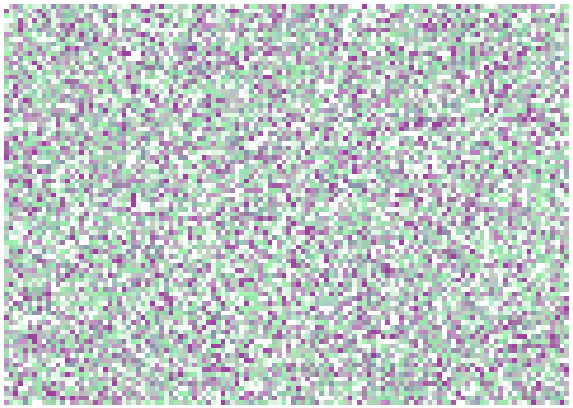
chiffré AES-128-ECB

Différents chiffrements du logo de mon laboratoire

logo original



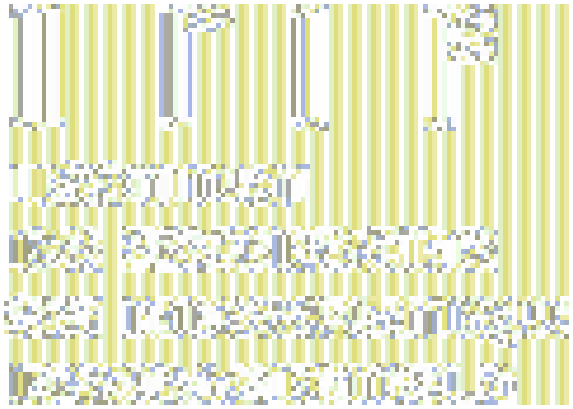
chiffré AES-128-ECB



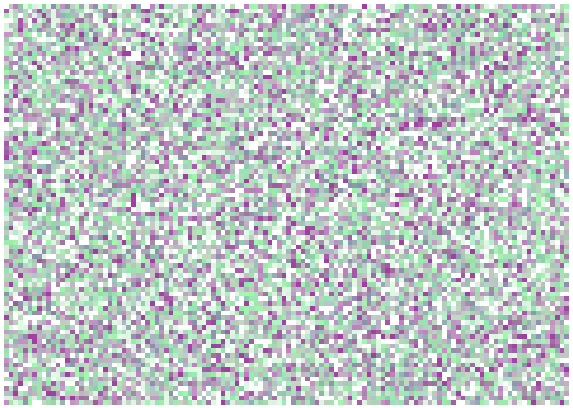
chiffré AES-128-CBC

Différents chiffrements du logo de mon laboratoire

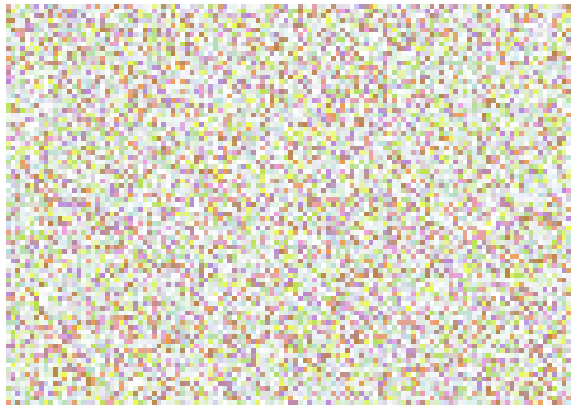
logo original



chiffré AES-128-ECB



chiffré AES-128-CBC



chiffré AES-128-CTR

Bibliothèque *OpenSSL* : `openssl enc`

Pour chiffrer le fichier `crypto.tex` avec AES-128 en mode CBC :

```
$ openssl enc -aes-128-cbc -in crypto.tex -out \
  crypto.tex-enc
enter aes-128-cbc encryption password: # MDP
Verifying - enter aes-128-cbc encryption password: # MDP
```

Pour déchiffrer le fichier `crypto.tex-enc` :

```
$ openssl enc -d -aes-128-cbc -in crypto.tex-enc -out \
  crypto.tex
enter aes-128-cbc decryption password: # MDP
```

Si on veut connaître le sel, la clé et l'IV utilisés :

```
$ openssl enc -p -aes-128-cbc -in crypto.tex -out \
  crypto.tex-enc
enter aes-128-cbc encryption password: # MDP
Verifying - enter aes-128-cbc encryption password: # MDP
salt=4F97FC74D6A7B435
key=B13AB0936E1E92F8F2EEFF1E2EA063AA
iv =8F19A3E5E7206B55A168BD7AEC63A33F
```

Bibliothèque *OpenSSL* : `openssl enc`

Pour chiffrer le fichier `crypto.tex` avec AES-128 en mode CBC :

```
$ openssl enc -aes-128-cbc -in crypto.tex -out \
  crypto.tex-enc
enter aes-128-cbc encryption password: # MDP
Verifying - enter aes-128-cbc encryption password: # MDP
```

Pour déchiffrer le fichier `crypto.tex-enc` :

```
$ openssl enc -d -aes-128-cbc -in crypto.tex-enc -out \
  crypto.tex
enter aes-128-cbc decryption password: # MDP
```

Si on veut connaître le sel, la clé et l'IV utilisés :

```
$ openssl enc -p -aes-128-cbc -in crypto.tex -out \
  crypto.tex-enc
enter aes-128-cbc encryption password: # MDP
Verifying - enter aes-128-cbc encryption password: # MDP
salt=4F97FC74D6A7B435
key=B13AB0936E1E92F8F2EEFF1E2EA063AA
iv =8F19A3E5E7206B55A168BD7AEC63A33F
```

Bibliothèque *OpenSSL* : openssl enc

Pour chiffrer le fichier crypto.tex avec AES-128 en mode CBC :

```
$ openssl enc -aes-128-cbc -in crypto.tex -out \
  crypto.tex-enc
enter aes-128-cbc encryption password: # MDP
Verifying - enter aes-128-cbc encryption password: # MDP
```

Pour déchiffrer le fichier crypto.tex-enc :

```
$ openssl enc -d -aes-128-cbc -in crypto.tex-enc -out \
  crypto.tex
enter aes-128-cbc decryption password: # MDP
```

Si on veut connaître le sel, la clé et l'IV utilisés :

```
$ openssl enc -p -aes-128-cbc -in crypto.tex -out \
  crypto.tex-enc
enter aes-128-cbc encryption password: # MDP
Verifying - enter aes-128-cbc encryption password: # MDP
salt=4F97FC74D6A7B435
key=B13AB0936E1E92F8F2EEFF1E2EA063AA
iv =8F19A3E5E7206B55A168BD7AEC63A33F
```

Le module *Python* : pycrypto

Un exemple de chiffrement AES sans *padding* (la clé, le vecteur d'initialisation et les données ont la bonne longueur : 16 octets) :

```
>>> from Crypto.Cipher import AES
>>> enc = AES.new("ma␣clé␣secrète", AES.MODE_CBC,
                  "vecteur␣d'init..")
>>> data = "Cours␣de␣crypto."
>>> ciphertxt = enc.encrypt(data) # chiffrement
>>> ciphertxt
b'\xa7\xa6]\x996\x07\xdf\xc0!\xde\x9e\x1c\xdcn\x19A'
>>> dec = AES.new("ma␣clé␣secrète", AES.MODE_CBC,
                  "vecteur␣d'init..")
>>> dec.decrypt(ciphertxt) # déchiffrement
b'Cours␣de␣crypto.'
```

- Pour fixer les idées, prenons le cas d'une urne électronique nécessitant la connaissance d'un secret s pour son ouverture à l'issue du scrutin. On voudrait que s soit **partagé** entre n scrutateurs $S_1, \ldots, S_n$ de façon à ce que k (ou plus) scrutateurs puissent ouvrir l'urne mais que cette ouverture soit impossible par $k-1$ (ou moins) scrutateurs.
- Un tel protocole cryptographique s'appelle un **schéma à seuil** et a été proposé par *Shamir* en 1979.
- On se donne un polynôme $P(X) = s + a_1 X + \cdots + a_{k-1} X^{k-1}$ à coefficients dans $\mathbb{F}_p$, $p > n$ et p premier. On communique alors à chaque scrutateur S_i le secret partiel $P(i) \in \mathbb{F}_p$.
- Si k (ou plus) scrutateurs mettent en commun les $P(i)$, ils peuvent reconstituer le polynôme P et donc, en particulier, le secret s.
- En revanche, à eux seuls, les scrutateurs $S_{i_1}, \ldots, S_{i_{k-1}}$, avec $1 \leq i_1 < \ldots < i_{k-1} \leq n$, ne pourront recouvrer le secret s. En effet, pour tout $a_0 \in \mathbb{F}_p$, il existe $Q \in \mathbb{F}_p[X]$ de degré $k-1$ tel que $Q(0) = a_0$ et $Q(i_j) = P(i_j)$ pour $j = 1, \ldots, k-1$.

- Pour fixer les idées, prenons le cas d'une urne électronique nécessitant la connaissance d'un secret s pour son ouverture à l'issue du scrutin. On voudrait que s soit **partagé** entre n scrutateurs $S_1, \ldots, S_n$ de façon à ce que k (ou plus) scrutateurs puissent ouvrir l'urne mais que cette ouverture soit impossible par $k-1$ (ou moins) scrutateurs.
- Un tel protocole cryptographique s'appelle un **schéma à seuil** et a été proposé par *Shamir* en 1979.
- On se donne un polynôme $P(X) = s + a_1 X + \cdots + a_{k-1} X^{k-1}$ à coefficients dans $\mathbb{F}_p$, $p > n$ et p premier. On communique alors à chaque scrutateur S_i le secret partiel $P(i) \in \mathbb{F}_p$.
- Si k (ou plus) scrutateurs mettent en commun les $P(i)$, ils peuvent reconstituer le polynôme P et donc, en particulier, le secret s.
- En revanche, à eux seuls, les scrutateurs $S_{i_1}, \ldots, S_{i_{k-1}}$, avec $1 \leq i_1 < \ldots < i_{k-1} \leq n$, ne pourront recouvrer le secret s. En effet, pour tout $a_0 \in \mathbb{F}_p$, il existe $Q \in \mathbb{F}_p[X]$ de degré $k-1$ tel que $Q(0) = a_0$ et $Q(i_j) = P(i_j)$ pour $j = 1, \ldots, k-1$.

- Pour fixer les idées, prenons le cas d'une urne électronique nécessitant la connaissance d'un secret s pour son ouverture à l'issue du scrutin. On voudrait que s soit **partagé** entre n scrutateurs $S_1, \ldots, S_n$ de façon à ce que k (ou plus) scrutateurs puissent ouvrir l'urne mais que cette ouverture soit impossible par $k-1$ (ou moins) scrutateurs.
- Un tel protocole cryptographique s'appelle un **schéma à seuil** et a été proposé par *Shamir* en 1979.
- On se donne un polynôme $P(X) = s + a_1 X + \cdots + a_{k-1} X^{k-1}$ à coefficients dans $\mathbb{F}_p$, $p > n$ et p premier. On communique alors à chaque scrutateur S_i le secret partiel $P(i) \in \mathbb{F}_p$.
- Si k (ou plus) scrutateurs mettent en commun les $P(i)$, ils peuvent reconstituer le polynôme P et donc, en particulier, le secret s.
- En revanche, à eux seuls, les scrutateurs $S_{i_1}, \ldots, S_{i_{k-1}}$, avec $1 \leq i_1 < \ldots < i_{k-1} \leq n$, ne pourront recouvrer le secret s. En effet, pour tout $a_0 \in \mathbb{F}_p$, il existe $Q \in \mathbb{F}_p[X]$ de degré $k-1$ tel que $Q(0) = a_0$ et $Q(i_j) = P(i_j)$ pour $j = 1, \ldots, k-1$.

- Pour fixer les idées, prenons le cas d'une urne électronique nécessitant la connaissance d'un secret s pour son ouverture à l'issue du scrutin. On voudrait que s soit **partagé** entre n scrutateurs $S_1, \ldots, S_n$ de façon à ce que k (ou plus) scrutateurs puissent ouvrir l'urne mais que cette ouverture soit impossible par $k-1$ (ou moins) scrutateurs.
- Un tel protocole cryptographique s'appelle un **schéma à seuil** et a été proposé par *Shamir* en 1979.
- On se donne un polynôme $P(X) = s + a_1 X + \cdots + a_{k-1} X^{k-1}$ à coefficients dans $\mathbb{F}_p$, $p > n$ et p premier. On communique alors à chaque scrutateur S_i le secret partiel $P(i) \in \mathbb{F}_p$.
- Si k (ou plus) scrutateurs mettent en commun les $P(i)$, ils peuvent reconstituer le polynôme P et donc, en particulier, le secret s.
- En revanche, à eux seuls, les scrutateurs $S_{i_1}, \ldots, S_{i_{k-1}}$, avec $1 \leq i_1 < \ldots < i_{k-1} \leq n$, ne pourront recouvrer le secret s. En effet, pour tout $a_0 \in \mathbb{F}_p$, il existe $Q \in \mathbb{F}_p[X]$ de degré $k-1$ tel que $Q(0) = a_0$ et $Q(i_j) = P(i_j)$ pour $j = 1, \ldots, k-1$.

- Pour fixer les idées, prenons le cas d'une urne électronique nécessitant la connaissance d'un secret s pour son ouverture à l'issue du scrutin. On voudrait que s soit **partagé** entre n scrutateurs $S_1, \ldots, S_n$ de façon à ce que k (ou plus) scrutateurs puissent ouvrir l'urne mais que cette ouverture soit impossible par $k-1$ (ou moins) scrutateurs.
- Un tel protocole cryptographique s'appelle un **schéma à seuil** et a été proposé par *Shamir* en 1979.
- On se donne un polynôme $P(X) = s + a_1 X + \cdots + a_{k-1} X^{k-1}$ à coefficients dans $\mathbb{F}_p$, $p > n$ et p premier. On communique alors à chaque scrutateur S_i le secret partiel $P(i) \in \mathbb{F}_p$.
- Si k (ou plus) scrutateurs mettent en commun les $P(i)$, ils peuvent reconstituer le polynôme P et donc, en particulier, le secret s.
- En revanche, à eux seuls, les scrutateurs $S_{i_1}, \ldots, S_{i_{k-1}}$, avec $1 \leq i_1 < \ldots < i_{k-1} \leq n$, ne pourront recouvrer le secret s. En effet, pour tout $a_0 \in \mathbb{F}_p$, il existe $Q \in \mathbb{F}_p[X]$ de degré $k-1$ tel que $Q(0) = a_0$ et $Q(i_j) = P(i_j)$ pour $j = 1, \ldots, k-1$.

Position du problème

- En cryptographie *asymétrique*, chaque acteur dispose d'un couple de clés : une clé **publique** et une clé **privée**.
- La clé privée, comme son nom l'indique, ne doit être à disposition que de son propriétaire (qui en a l'entière responsabilité). En revanche, le clé publique **doit être accessible** à tout un chacun.
- La diffusion de cette clé publique doit respecter plusieurs critères :
  - authentification : on doit être sûr que la clé est bien celle de la personne avec qui on va échanger des données confidentielles (risque de l'attaque *"man in the middle"*).
  - confiance : la personne en question doit être digne de confiance.
  - validité : une clé publique a une durée de vie en général **finie**. On doit donc pouvoir vérifier cette dernière.
- Un des moyens de résoudre ce problème : les *certificats* et *autorités de certification* gérés au sein d'une **infrastructure de gestion de clés** - IGC (*Public Key Infrastructure - PKI*).

Position du problème

- En cryptographie *asymétrique*, chaque acteur dispose d'un couple de clés : une clé **publique** et une clé **privée**.
- La clé privée, comme son nom l'indique, ne doit être à disposition que de son propriétaire (qui en a l'entière responsabilité). En revanche, le clé publique **doit être accessible** à tout un chacun.
- La diffusion de cette clé publique doit respecter plusieurs critères :
  - authentification : on doit être sûr que la clé est bien celle de la personne avec qui on va échanger des données confidentielles (risque de l'attaque *"man in the middle"*).
  - confiance : la personne en question doit être digne de confiance.
  - validité : une clé publique a une durée de vie en général **finie**. On doit donc pouvoir vérifier cette dernière.
- Un des moyens de résoudre ce problème : les *certificats* et *autorités de certification* gérés au sein d'une **infrastructure de gestion de clés** - IGC (*Public Key Infrastructure - PKI*).

Position du problème

- En cryptographie *asymétrique*, chaque acteur dispose d'un couple de clés : une clé **publique** et une clé **privée**.
- La clé privée, comme son nom l'indique, ne doit être à disposition que de son propriétaire (qui en a l'entière responsabilité). En revanche, le clé publique **doit être accessible** à tout un chacun.
- La diffusion de cette clé publique doit respecter plusieurs critères :
  - authentification : on doit être sûr que la clé est bien celle de la personne avec qui on va échanger des données confidentielles (risque de l'attaque *"man in the middle"*).
  - confiance : la personne en question doit être digne de confiance.
  - validité : une clé publique a une durée de vie en général **finie**. On doit donc pouvoir vérifier cette dernière.
- Un des moyens de résoudre ce problème : les *certificats* et *autorités de certification* gérés au sein d'une **infrastructure de gestion de clés** - IGC (*Public Key Infrastructure - PKI*).

Position du problème

- En cryptographie *asymétrique*, chaque acteur dispose d'un couple de clés : une clé **publique** et une clé **privée**.
- La clé privée, comme son nom l'indique, ne doit être à disposition que de son propriétaire (qui en a l'entière responsabilité). En revanche, le clé publique **doit être accessible** à tout un chacun.
- La diffusion de cette clé publique doit respecter plusieurs critères :
  - authentification : on doit être sûr que la clé est bien celle de la personne avec qui on va échanger des données confidentielles (risque de l'attaque *"man in the middle"*).
  - confiance : la personne en question doit être digne de confiance.
  - validité : une clé publique a une durée de vie en général **finie**. On doit donc pouvoir vérifier cette dernière.
- Un des moyens de résoudre ce problème : les *certificats* et *autorités de certification* gérés au sein d'une **infrastructure de gestion de clés** - IGC (*Public Key Infrastructure - PKI*).

Position du problème

- En cryptographie *asymétrique*, chaque acteur dispose d'un couple de clés : une clé **publique** et une clé **privée**.
- La clé privée, comme son nom l'indique, ne doit être à disposition que de son propriétaire (qui en a l'entière responsabilité). En revanche, le clé publique **doit être accessible** à tout un chacun.
- La diffusion de cette clé publique doit respecter plusieurs critères :
  - authentification : on doit être sûr que la clé est bien celle de la personne avec qui on va échanger des données confidentielles (risque de l'attaque *"man in the middle"*).
  - confiance : la personne en question doit être digne de confiance.
  - validité : une clé publique a une durée de vie en général **finie**. On doit donc pouvoir vérifier cette dernière.
- Un des moyens de résoudre ce problème : les *certificats* et *autorités de certification* gérés au sein d'une **infrastructure de gestion de clés** - IGC (*Public Key Infrastructure - PKI*).

Position du problème

- En cryptographie *asymétrique*, chaque acteur dispose d'un couple de clés : une clé **publique** et une clé **privée**.
- La clé privée, comme son nom l'indique, ne doit être à disposition que de son propriétaire (qui en a l'entière responsabilité). En revanche, le clé publique **doit être accessible** à tout un chacun.
- La diffusion de cette clé publique doit respecter plusieurs critères :
  - authentification : on doit être sûr que la clé est bien celle de la personne avec qui on va échanger des données confidentielles (risque de l'attaque *"man in the middle"*).
  - confiance : la personne en question doit être digne de confiance.
  - validité : une clé publique a une durée de vie en général **finie**. On doit donc pouvoir vérifier cette dernière.
- Un des moyens de résoudre ce problème : les *certificats* et *autorités de certification* gérés au sein d'une **infrastructure de gestion de clés** - IGC (*Public Key Infrastructure - PKI*).

Position du problème

- En cryptographie *asymétrique*, chaque acteur dispose d'un couple de clés : une clé **publique** et une clé **privée**.
- La clé privée, comme son nom l'indique, ne doit être à disposition que de son propriétaire (qui en a l'entière responsabilité). En revanche, le clé publique **doit être accessible** à tout un chacun.
- La diffusion de cette clé publique doit respecter plusieurs critères :
  - authentification : on doit être sûr que la clé est bien celle de la personne avec qui on va échanger des données confidentielles (risque de l'attaque *"man in the middle"*).
  - confiance : la personne en question doit être digne de confiance.
  - validité : une clé publique a une durée de vie en général **finie**. On doit donc pouvoir vérifier cette dernière.
- Un des moyens de résoudre ce problème : les *certificats* et *autorités de certification* gérés au sein d'une **infrastructure de gestion de clés** - IGC (*Public Key Infrastructure - PKI*).

Qu'est-ce qu'un certificat ?

- Formellement, c'est un document numérique contenant une **clé publique** ainsi que d'autres informations associées à cette clé.
- Ce document peut être authentifié (*signature numérique*) de différentes manières :
  - soit par une autorité de certification (AC). C'est la norme X.509 de l'ISO définie en 1988 dans le cadre du projet X.500. Le modèle sous-jacent est un *système hiérarchique* d'autorités de certification.
  - soit par quiconque appartenant à un réseau de confiance. C'est par exemple la *toile de confiance (web of trust)* d'OpenPGP.
- Dans le modèle X.509, une des limites est la confiance que l'on peut accorder à une AC. Cette confiance est essentielle car elle conditionne tout le reste. En effet si une AC est compromise, l'ensemble des clés qui en dépendent est compromis.
- C'est ce problème de confiance qui a amené *Paul Zimmermann*, auteur de PGP, à introduire, au début des années 90, le concept de toile de confiance.

Qu'est-ce qu'un certificat ?

- Formellement, c'est un document numérique contenant une **clé publique** ainsi que d'autres informations associées à cette clé.
- Ce document peut être authentifié (*signature numérique*) de différentes manières :
  - soit par une autorité de certification (AC). C'est la norme X.509 de l'ISO définie en 1988 dans le cadre du projet X.500. Le modèle sous-jacent est un *système hiérarchique* d'autorités de certification.
  - soit par quiconque appartenant à un réseau de confiance. C'est par exemple la *toile de confiance (web of trust)* d'OpenPGP.
- Dans le modèle X.509, une des limites est la confiance que l'on peut accorder à une AC. Cette confiance est essentielle car elle conditionne tout le reste. En effet si une AC est compromise, l'ensemble des clés qui en dépendent est compromis.
- C'est ce problème de confiance qui a amené *Paul Zimmermann*, auteur de PGP, à introduire, au début des années 90, le concept de toile de confiance.

Qu'est-ce qu'un certificat ?

- Formellement, c'est un document numérique contenant une **clé publique** ainsi que d'autres informations associées à cette clé.
- Ce document peut être authentifié (*signature numérique*) de différentes manières :
  - soit par une autorité de certification (AC). C'est la norme X.509 de l'ISO définie en 1988 dans le cadre du projet X.500. Le modèle sous-jacent est un *système hiérarchique* d'autorités de certification.
  - soit par quiconque appartenant à un réseau de confiance. C'est par exemple la *toile de confiance (web of trust)* d'OpenPGP.
- Dans le modèle X.509, une des limites est la confiance que l'on peut accorder à une AC. Cette confiance est essentielle car elle conditionne tout le reste. En effet si une AC est compromise, l'ensemble des clés qui en dépendent est compromis.
- C'est ce problème de confiance qui a amené *Paul Zimmermann*, auteur de PGP, à introduire, au début des années 90, le concept de toile de confiance.

Qu'est-ce qu'un certificat ?

- Formellement, c'est un document numérique contenant une **clé publique** ainsi que d'autres informations associées à cette clé.
- Ce document peut être authentifié (*signature numérique*) de différentes manières :
  - soit par une autorité de certification (AC). C'est la norme X.509 de l'ISO définie en 1988 dans le cadre du projet X.500. Le modèle sous-jacent est un *système hiérarchique* d'autorités de certification.
  - soit par quiconque appartenant à un réseau de confiance. C'est par exemple la *toile de confiance (web of trust)* d'OpenPGP.
- Dans le modèle X.509, une des limites est la confiance que l'on peut accorder à une AC. Cette confiance est essentielle car elle conditionne tout le reste. En effet si une AC est compromise, l'ensemble des clés qui en dépendent est compromis.
- C'est ce problème de confiance qui a amené *Paul Zimmermann*, auteur de PGP, à introduire, au début des années 90, le concept de toile de confiance.

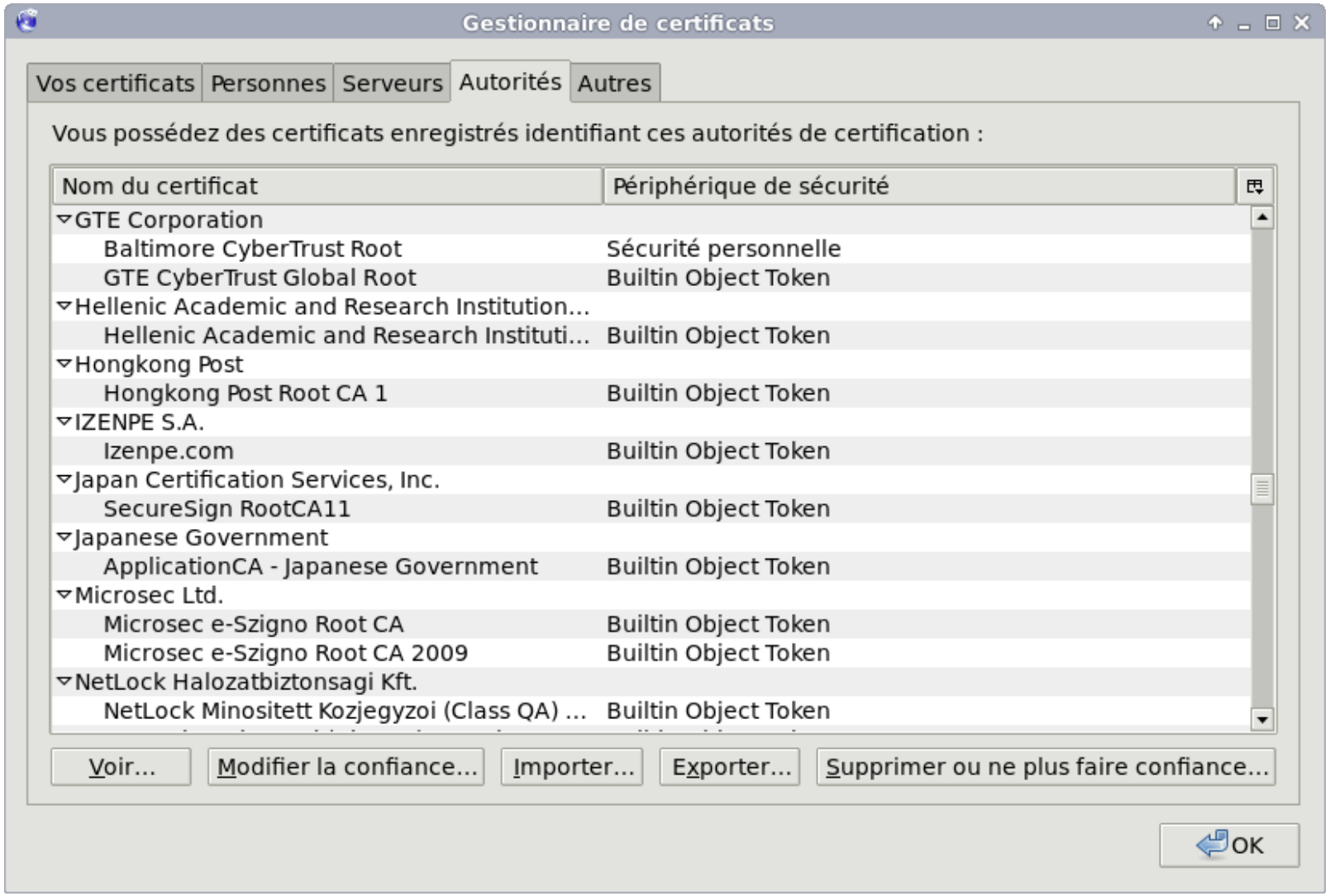
Qu'est-ce qu'un certificat ?

- Formellement, c'est un document numérique contenant une **clé publique** ainsi que d'autres informations associées à cette clé.
- Ce document peut être authentifié (*signature numérique*) de différentes manières :
  - soit par une autorité de certification (AC). C'est la norme X.509 de l'ISO définie en 1988 dans le cadre du projet X.500. Le modèle sous-jacent est un *système hiérarchique* d'autorités de certification.
  - soit par quiconque appartenant à un réseau de confiance. C'est par exemple la *toile de confiance (web of trust)* d'OpenPGP.
- Dans le modèle X.509, une des limites est la confiance que l'on peut accorder à une AC. Cette confiance est essentielle car elle conditionne tout le reste. En effet si une AC est compromise, l'ensemble des clés qui en dépendent est compromis.
- C'est ce problème de confiance qui a amené *Paul Zimmermann*, auteur de PGP, à introduire, au début des années 90, le concept de toile de confiance.

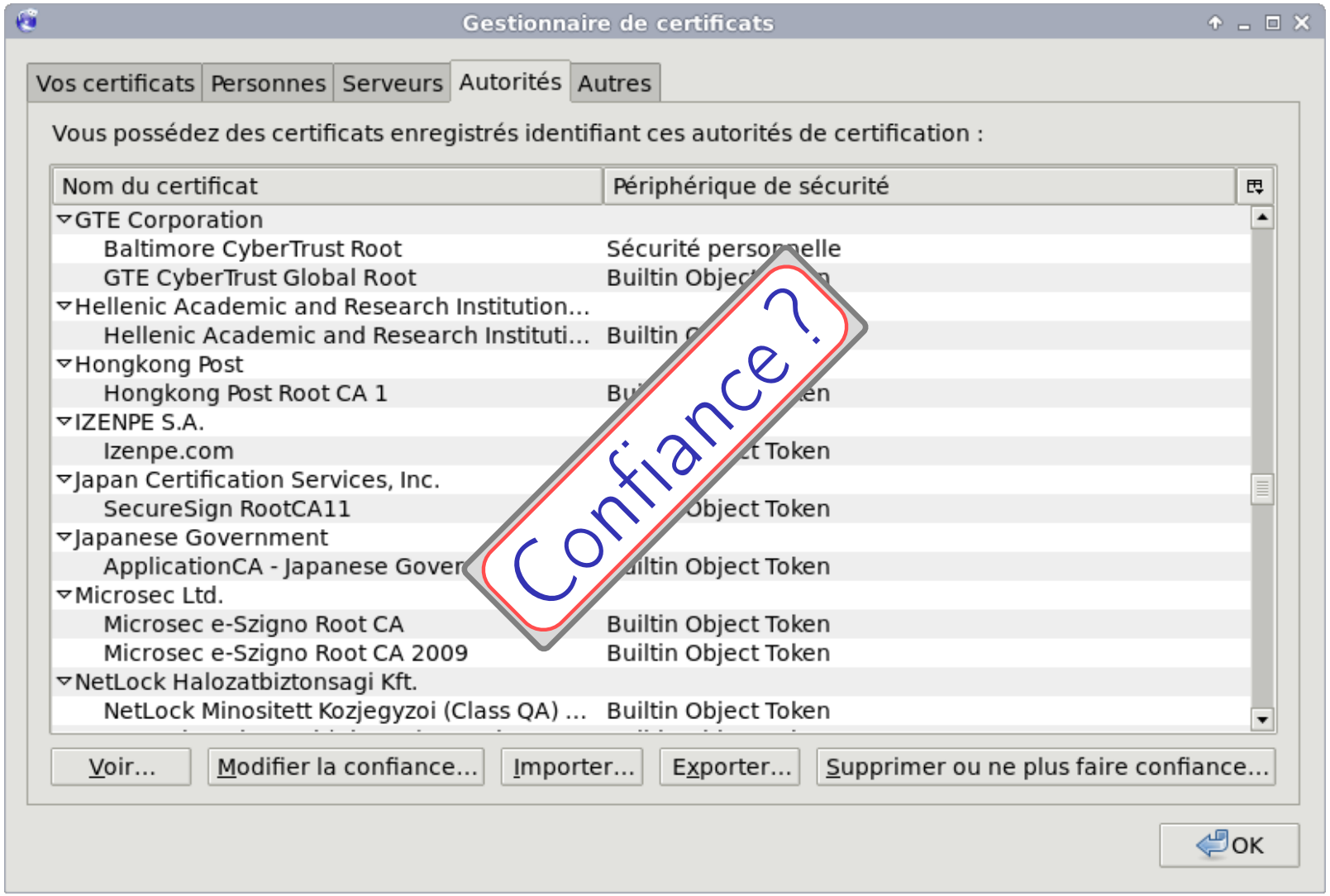
Qu'est-ce qu'un certificat ?

- Formellement, c'est un document numérique contenant une **clé publique** ainsi que d'autres informations associées à cette clé.
- Ce document peut être authentifié (*signature numérique*) de différentes manières :
  - soit par une autorité de certification (AC). C'est la norme X.509 de l'ISO définie en 1988 dans le cadre du projet X.500. Le modèle sous-jacent est un *système hiérarchique* d'autorités de certification.
  - soit par quiconque appartenant à un réseau de confiance. C'est par exemple la *toile de confiance (web of trust)* d'OpenPGP.
- Dans le modèle X.509, une des limites est la confiance que l'on peut accorder à une AC. Cette confiance est essentielle car elle conditionne tout le reste. En effet si une AC est compromise, l'ensemble des clés qui en dépendent est compromis.
- C'est ce problème de confiance qui a amené *Paul Zimmermann*, auteur de PGP, à introduire, au début des années 90, le concept de toile de confiance.

Quelques autorités de certification (navigateur *Firefox*)



Quelques autorités de certification (navigateur *Firefox*)



Certificat X.509

Données	`Version:`	Version du type de certificat X.509
	`Serial number:`	Numéro de série au sein de l'AC
	`Signature algorithm:`	Algorithme de signature utilisé
	`Issuer:`	Identité de l'AC (*Distinguished Name*) qui a émis ce certificat

	`Validity`
	` Not before: xx`
	` Not after : xx`
		Période de validité du certificat : dates de début et de fin

	`Subject:`	Identité du propriétaire (*Distinguished Name*) du certificat
	`Subject Public Key Info:` ` Public Key Algorithm:`	Informations sur la clé publique et les paramètres de celle-ci

	`X509v3 extensions:`
	` Extension name:`
	`  Extension value`
	` ...`
		Extensions optionnelles propres à la version 3

Signature	`Signature algorithm:`	Algorithme utilisé pour la signature
	`Signature`	Signature du certificat

Quelques extensions X.509v3

`X509v3 Basic Constraints:`	Indique s'il s'agit du certificat d'une AC ou non
`X509v3 Key Usage:`	Précise les fonctionnalités du certificat. Par exemple, il peut être utilisé pour signer d'autres certificats (*Certificate sign*)
`X509v3 subjectAltName:`	Autres noms du propriétaire du certificat. Ce sont des alias du champ *Subject*
`X509v3 issuerAltName:`	Autres noms de l'émetteur du certificat. Ce sont des alias du champ *Issuer*
`X509v3 CRL Distribution points:`	URI de la *Certificat Revocation List (CRL)* permettant de connaître le statut du certificat

Extensions X.509v3 : compléments

- Une extension peut être qualifiée de **critique** (*critical*).
- Une application doit **impérativement** traiter les extensions marquées comme *critique*.
- Une application, ayant à traiter une extension marquée comme *critique* qui lui est **inconnue**, doit ignorer le certificat contenant cette extension.
- Lorsque l'on utilise les extensions *Basic Constraints* et *Key Usage*, celles-ci doivent être marquées comme *critiques*.

Extensions X.509v3 : compléments

- Une extension peut être qualifiée de **critique** (*critical*).
- Une application doit **impérativement** traiter les extensions marquées comme *critique*.
- Une application, ayant à traiter une extension marquée comme *critique* qui lui est **inconnue**, doit ignorer le certificat contenant cette extension.
- Lorsque l'on utilise les extensions *Basic Constraints* et *Key Usage*, celles-ci doivent être marquées comme *critiques*.

Extensions X.509v3 : compléments

- Une extension peut être qualifiée de **critique** (*critical*).
- Une application doit **impérativement** traiter les extensions marquées comme *critique*.
- Une application, ayant à traiter une extension marquée comme *critique* qui lui est **inconnue**, doit ignorer le certificat contenant cette extension.
- Lorsque l'on utilise les extensions *Basic Constraints* et *Key Usage*, celles-ci doivent être marquées comme *critiques*.

Extensions X.509v3 : compléments

- Une extension peut être qualifiée de **critique** (*critical*).
- Une application doit **impérativement** traiter les extensions marquées comme *critique*.
- Une application, ayant à traiter une extension marquée comme *critique* qui lui est **inconnue**, doit ignorer le certificat contenant cette extension.
- Lorsque l'on utilise les extensions *Basic Constraints* et *Key Usage*, celles-ci doivent être marquées comme *critiques*.

Le certificat CNRS2-Plus émis par le CNRS :

```
$ openssl x509 -in CNRS2-Plus -text -noout
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number: 2 (0x2)
Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=FR, O=CNRS, CN=CNRS2
    Validity
        Not Before: Jan 21 09:03:14 2009 GMT
        Not After : Jan 20 09:03:14 2029 GMT
    Subject: C=FR, O=CNRS, CN=CNRS2-Plus
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:c8:41:5f:ba:8b:92:4d:42:4e:db:08:2b:0b:2e:
                38:6e:f0:18:63:7c:64:08:7d:8a:92:c4:86:03:7a:
                ..........................................
                27:6c:fc:2f:85:bf:9d:0d:2e:9e:4f:2c:0c:f3:39:
                03:6f
            Exponent: 65537 (0x10001) # nombre de Fermat F4
```

Certificat CNRS2-Plus (suite) :

```
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:TRUE
    X509v3 Subject Key Identifier:
        49:B4:FB:3C:59:9A:66:EB:3D:31:CA:29:6E:FB:81:...
    X509v3 Authority Key Identifier:
        keyid:50:97:B6:0D:F7:AC:33:17:AF:F1:1D:46:3C:...
        DirName:/C=FR/O=CNRS/CN=CNRS2
        serial:00

    X509v3 Key Usage: critical
        Certificate Sign, CRL Sign
    X509v3 CRL Distribution Points:

        Full Name:
          URI:http://crls.services.cnrs.fr/CNRS2/getder.crl

Signature Algorithm: sha1WithRSAEncryption
     3d:9b:06:d5:71:2b:82:6f:35:24:1b:b8:82:6e:95:ff:06:28:
     ad:2c:12:c2:aa:79:d1:e8:fb:02:d3:eb:f0:c4:f9:2c:8d:91:
     ..........................................
     5a:1b:28:36
```

Création d'un certificat auto-signé avec OpenSSL :

```
$ openssl genrsa -F4 -out ca.key 2048 # Création de la clé
$ openssl req -new -x509 -key ca.key -out ca.pem -days 365 \
    -set_serial 1 -subj /C=FR/CN=Cours-de-Cryto/O=IRIF/
```

Création d'un certificat auto-signé avec OpenSSL :

```
$ openssl genrsa -F4 -out ca.key 2048 # Création de la clé
$ openssl req -new -x509 -key ca.key -out ca.pem -days 365 \
    -set_serial 1 -subj /C=FR/CN=Cours-de-Cryto/O=IRIF/
```

- option `-new` : génère une nouvelle demande de certificat.

Création d'un certificat auto-signé avec OpenSSL :

```
$ openssl genrsa -F4 -out ca.key 2048 # Création de la clé
$ openssl req -new -x509 -key ca.key -out ca.pem -days 365 \
    -set_serial 1 -subj /C=FR/CN=Cours-de-Cryto/O=IRIF/
```

- option `-new` : génère une nouvelle demande de certificat.
- option `-x509` : génère un **certificat auto-signé** à la place d'une simple demande (ce qui est le cas lorsque l'option `-new` est seule).

Création d'un certificat auto-signé avec OpenSSL :

```
$ openssl genrsa -F4 -out ca.key 2048 # Création de la clé
$ openssl req -new -x509 -key ca.key -out ca.pem -days 365 \
    -set_serial 1 -subj /C=FR/CN=Cours-de-Cryto/O=IRIF/
```

- option `-new` : génère une nouvelle demande de certificat.
- option `-x509` : génère un **certificat auto-signé** à la place d'une simple demande (ce qui est le cas lorsque l'option `-new` est seule).
- option `-key` : le fichier contenant le couple (clé publique, clé privée). Ici c'est le fichier `ca.key` qui a été généré avec la commande `openssl genrsa`.

Création d'un certificat auto-signé avec OpenSSL :

```
$ openssl genrsa -F4 -out ca.key 2048 # Création de la clé
$ openssl req -new -x509 -key ca.key -out ca.pem -days 365 \
    -set_serial 1 -subj /C=FR/CN=Cours-de-Cryto/O=IRIF/
```

- option `-new` : génère une nouvelle demande de certificat.
- option `-x509` : génère un **certificat auto-signé** à la place d'une simple demande (ce qui est le cas lorsque l'option `-new` est seule).
- option `-key` : le fichier contenant le couple (clé publique, clé privée). Ici c'est le fichier `ca.key` qui a été généré avec la commande `openssl genrsa`.
- option `-out` : le nom du fichier de sortie qui contiendra le certificat auto-signé au format PEM (*Privacy-enhanced Electronic Mail* – RFCs 1421-1424).

Création d'un certificat auto-signé avec OpenSSL :

```
$ openssl genrsa -F4 -out ca.key 2048 # Création de la clé
$ openssl req -new -x509 -key ca.key -out ca.pem -days 365 \
    -set_serial 1 -subj /C=FR/CN=Cours-de-Cryto/O=IRIF/
```

- option `-new` : génère une nouvelle demande de certificat.
- option `-x509` : génère un **certificat auto-signé** à la place d'une simple demande (ce qui est le cas lorsque l'option `-new` est seule).
- option `-key` : le fichier contenant le couple (clé publique, clé privée). Ici c'est le fichier `ca.key` qui a été généré avec la commande `openssl genrsa`.
- option `-out` : le nom du fichier de sortie qui contiendra le certificat auto-signé au format PEM (*Privacy-enhanced Electronic Mail* – RFCs 1421-1424).
- option `-days` : durée de vie en jours du certificat (ici une année).

Création d'un certificat auto-signé avec OpenSSL :

```
$ openssl genrsa -F4 -out ca.key 2048 # Création de la clé
$ openssl req -new -x509 -key ca.key -out ca.pem -days 365 \
    -set_serial 1 -subj /C=FR/CN=Cours-de-Cryto/O=IRIF/
```

- option `-new` : génère une nouvelle demande de certificat.
- option `-x509` : génère un **certificat auto-signé** à la place d'une simple demande (ce qui est le cas lorsque l'option `-new` est seule).
- option `-key` : le fichier contenant le couple (clé publique, clé privée). Ici c'est le fichier `ca.key` qui a été généré avec la commande `openssl genrsa`.
- option `-out` : le nom du fichier de sortie qui contiendra le certificat auto-signé au format PEM (*Privacy-enhanced Electronic Mail* – RFCs 1421-1424).
- option `-days` : durée de vie en jours du certificat (ici une année).
- option `-set_serial` : numéro de série au sein de l'autorité de certification.

Création d'un certificat auto-signé avec OpenSSL :

```
$ openssl genrsa -F4 -out ca.key 2048 # Création de la clé
$ openssl req -new -x509 -key ca.key -out ca.pem -days 365 \
    -set_serial 1 -subj /C=FR/CN=Cours-de-Cryto/O=IRIF/
```

- option `-new` : génère une nouvelle demande de certificat.
- option `-x509` : génère un **certificat auto-signé** à la place d'une simple demande (ce qui est le cas lorsque l'option `-new` est seule).
- option `-key` : le fichier contenant le couple (clé publique, clé privée). Ici c'est le fichier `ca.key` qui a été généré avec la commande `openssl genrsa`.
- option `-out` : le nom du fichier de sortie qui contiendra le certificat auto-signé au format PEM (*Privacy-enhanced Electronic Mail* – RFCs 1421-1424).
- option `-days` : durée de vie en jours du certificat (ici une année).
- option `-set_serial` : numéro de série au sein de l'autorité de certification.
- option `-subj` : identité du propriétaire – `/C=ISO Country Code`, `/ST=State`, `/L=Location,city`, `/O=Organization`, `/OU=Organizational Unit` et `/CN=Common Name (FQDN pour un site WEB)`.

Création d'un certificat auto-signé (suite)

```
$ cat ca.pem
-----BEGIN CERTIFICATE-----
MIIDNTCCAh2gAwIBAgIBATANBgkqhkiG9w0BAQsFADA1MQswCQYDVQQGEwJGUjEX
MBUGA1UEAwwOQ291cnMtZGUtQ3J5dG8xDTALBgNVBAoMBElSSUYwHhcNMTYwOTA4
MDg0MTM3WhcNMTcwOTA4MDg0MTM3WjA1MQswCQYDVQQGEwJGUjEXMBUGA1UEAwwO
Q291cnMtZGUtQ3J5dG8xDTALBgNVBAoMBElSSUYwggEiMA0GCSqGSIb3DQEBAQUA
A4IBDwAwggEKAoIBAQDA/2UR9ceUIPU5KhMxk/s0TsUJZPC4QZfjafpm3M9KDJrL
VW+cGQYv9V07mrhxPXlmdYmAOhHutjI/QhdhpzCmVkhYpvjP6A6lyhCTHjCvbs7A
R8sq6Aj/dg2XuyzN4+aaWlZnDFsO2oyUuL9CGoknDhZmMjd5OwD1QjMUUmPJSdwN
20GLnOCWgYar28egsC82nFP9U2kEHZujPa87PjEmHd9Pc2bPdLVGBUWjbUs4h9i9
/8CeZZxjZ6SdH6bPcDnNKhg61Bo7DGe4O7rkwWF0ojLAlorhz+sBnQOmlB2eMWBE
gY9f4mxUrKTMp3wuLQzPBtt9H/f0wR2GTNYcEnkrAgMBAAGjUDBOMB0GA1UdDgQW
BBQ+WalqGPawObjPD5S/wjfxqYXvijAfBgNVHSMEGDAWgBQ+WalqGPawObjPD5S/
wjfxqYXvijAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQAfNjjSRMWN
iAYvgnAxAz6JZIS9tpthlgVbev1lIFaOHZHLIFr0D8TlgcHoWXiJsCy8MivXlTt7
Ye/Napp6x6X6LxMAorpKnQ1xWBwwuEB413iFzsU0v5qDLPYzB8uRE3d+T6aTx41n
hUG8gqDB+vNGab/O38USc6kMkOczbfUlrDmlDoj/DWHbrC68ozhDunQ5ElWk08fq
8dged4F3XiXFAKgcN8BWOi1CZKeUq8SAs1hIMvfROgRtENdW37wJ1E+SinlcjsTx
ezTTiDiZ+L8yNcCZZrkXJICiFWrldIBwk/gRg+jfqGt6TKX7RqqbbTaDsVmtQZJD
uQG39faWaknF
-----END CERTIFICATE-----
```

Création d'un certificat auto-signé (suite)

```
$ openssl x509 -in ca.pem -text -noout
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=FR, CN=Cours-de-Cryto, O=IRIF
    Validity
        Not Before: Sep  8 08:41:37 2016 GMT
        Not After : Sep  8 08:41:37 2017 GMT
    Subject: C=FR, CN=Cours-de-Cryto, O=IRIF
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:c0:ff:65:11:f5:c7:94:20:f5:39:2a:13:31:93:
                fb:34:4e:c5:09:64:f0:b8:41:97:e3:69:fa:66:dc:
                ..........................................
                0c:cf:06:db:7d:1f:f7:f4:c1:1d:86:4c:d6:1c:12:
                79:2b
            Exponent: 65537 (0x10001) # nombre de Fermat F4
```

Création d'un certificat auto-signé (suite et fin)

```
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                3E:59:A9:6A:18:F6:B0:39:B8:CF:0F:94:BF:C2:37:...
            X509v3 Authority Key Identifier:
                keyid:3E:59:A9:6A:18:F6:B0:39:B8:CF:0F:94:BF:...

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         1f:36:38:d2:44:c5:8d:88:06:2f:82:70:31:03:3e:89:64:84:
         bd:b6:9b:61:96:05:5b:7a:fd:65:20:56:8e:1d:91:cb:20:5a:
         ..................................................
         fb:46:aa:9b:6d:36:83:b1:59:ad:41:92:43:b9:01:b7:f5:f6:
         96:6a:49:c5
```

Création d'un certificat auto-signé (suite et fin)

```
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                3E:59:A9:6A:18:F6:B0:39:B8:CF:0F:94:BF:C2:37:...
            X509v3 Authority Key Identifier:
                keyid:3E:59:A9:6A:18:F6:B0:39:B8:CF:0F:94:BF:...

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         1f:36:38:d2:44:c5:8d:88:06:2f:82:70:31:03:3e:89:64:84:
         bd:b6:9b:61:96:05:5b:7a:fd:65:20:56:8e:1d:91:cb:20:5a:
         ..............................................
         fb:46:aa:9b:6d:36:83:b1:59:ad:41:92:43:b9:01:b7:f5:f6:
         96:6a:49:c5
```

Création de l'index permettant de rechercher, dans le répertoire, le certificat via le champ sujet :

```
$ ln -s ca.pem `openssl x509 -subject_hash -noout -in ca.pem`.0
$ ls -l
lrwxrwxrwx 1 ylg ylg 6 sept.  8 14:15 78c90f38.0 -> ca.pem
```

Création d'un certificat auto-signé (suite et fin)

```
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                3E:59:A9:6A:18:F6:B0:39:B8:CF:0F:94:BF:C2:37:...
            X509v3 Authority Key Identifier:
                keyid:3E:59:A9:6A:18:F6:B0:39:B8:CF:0F:94:BF:...

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         1f:36:38:d2:44:c5:8d:88:06:2f:82:70:31:03:3e:89:64:84:
         bd:b6:9b:61:96:05:5b:7a:fd:65:20:56:8e:1d:91:cb:20:5a:
         ..............................................
         fb:46:aa:9b:6d:36:83:b1:59:ad:41:92:43:b9:01:b7:f5:f6:
         96:6a:49:c5
```

Création de l'index permettant de rechercher, dans le répertoire, le certificat via le champ sujet :

```
$ ln -s ca.pem `openssl x509 -subject_hash -noout -in ca.pem`.0
$ ls -l
lrwxrwxrwx 1 ylg ylg 6 sept.  8 14:15 78c90f38.0 -> ca.pem
```

Vérification du certificat :

```
$ openssl verify -CApath . ca.pem
ca.pem: OK
```

Principes

- Pour être **valide**, un certificat doit être **signé par une AC**.
- Une AC possède son propre couple (clé privée, clé publique) associé à un certificat qui peut être, soit **auto-signé** (on a vu précédemment comment créer un tel certificat), soit **signé par une autre AC**.
- N'importe qui peut se déclarer AC, ce qui pose évidemment le problème de la **confiance** en une AC. Or cette confiance est cruciale car la securité de ce système hiérarchique repose, pour une grande part, sur celle-ci.
- La confiance accordée à une AC est **héritée** par toutes les ACs filles. Autrement dit, faire confiance à une AC *racine*, implique que l'on accorde la même confiance à toutes les ACs qui en dérivent.
- Deux ACs peuvent s'entendre afin de signer chacune le certificat de l'autre : c'est la notion de **confiance croisée**.
- L'AC est **garante** des informations contenues dans les certificats qu'elle délivre.

Principes

- Pour être **valide**, un certificat doit être **signé par une AC**.
- Une AC possède son propre couple (clé privée, clé publique) associé à un certificat qui peut être, soit **auto-signé** (on a vu précédemment comment créer un tel certificat), soit **signé par une autre AC**.
- N'importe qui peut se déclarer AC, ce qui pose évidemment le problème de la **confiance** en une AC. Or cette confiance est cruciale car la securité de ce système hiérarchique repose, pour une grande part, sur celle-ci.
- La confiance accordée à une AC est **héritée** par toutes les ACs filles. Autrement dit, faire confiance à une AC *racine*, implique que l'on accorde la même confiance à toutes les ACs qui en dérivent.
- Deux ACs peuvent s'entendre afin de signer chacune le certificat de l'autre : c'est la notion de **confiance croisée**.
- L'AC est **garante** des informations contenues dans les certificats qu'elle délivre.

Principes

- Pour être **valide**, un certificat doit être **signé par une AC**.
- Une AC possède son propre couple (clé privée, clé publique) associé à un certificat qui peut être, soit **auto-signé** (on a vu précédemment comment créer un tel certificat), soit **signé par une autre AC**.
- N'importe qui peut se déclarer AC, ce qui pose évidemment le problème de la **confiance** en une AC. Or cette confiance est cruciale car la securité de ce système hiérarchique repose, pour une grande part, sur celle-ci.
- La confiance accordée à une AC est **héritée** par toutes les ACs filles. Autrement dit, faire confiance à une AC *racine*, implique que l'on accorde la même confiance à toutes les ACs qui en dérivent.
- Deux ACs peuvent s'entendre afin de signer chacune le certificat de l'autre : c'est la notion de **confiance croisée**.
- L'AC est **garante** des informations contenues dans les certificats qu'elle délivre.

Principes

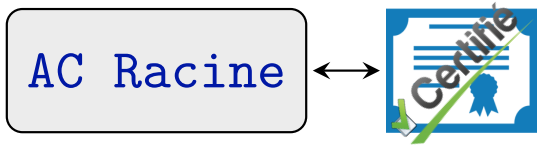
- Pour être **valide**, un certificat doit être **signé par une AC**.
- Une AC possède son propre couple (clé privée, clé publique) associé à un certificat qui peut être, soit **auto-signé** (on a vu précédemment comment créer un tel certificat), soit **signé par une autre AC**.
- N'importe qui peut se déclarer AC, ce qui pose évidemment le problème de la **confiance** en une AC. Or cette confiance est cruciale car la securité de ce système hiérarchique repose, pour une grande part, sur celle-ci.
- La confiance accordée à une AC est **héritée** par toutes les ACs filles. Autrement dit, faire confiance à une AC *racine*, implique que l'on accorde la même confiance à toutes les ACs qui en dérivent.
- Deux ACs peuvent s'entendre afin de signer chacune le certificat de l'autre : c'est la notion de **confiance croisée**.
- L'AC est **garante** des informations contenues dans les certificats qu'elle délivre.

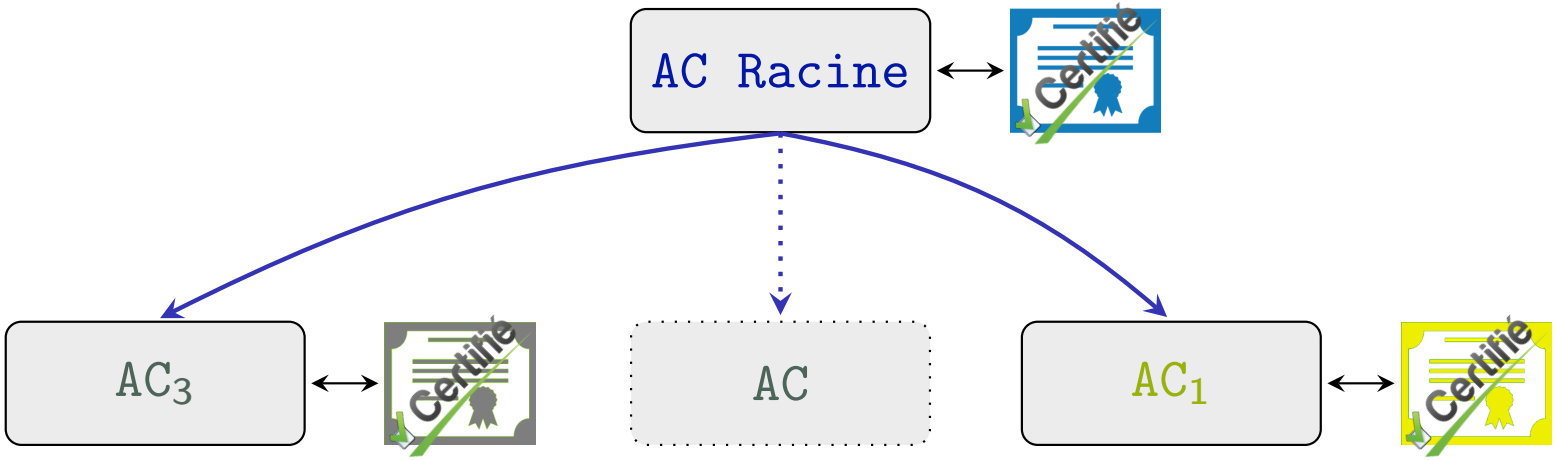
Principes

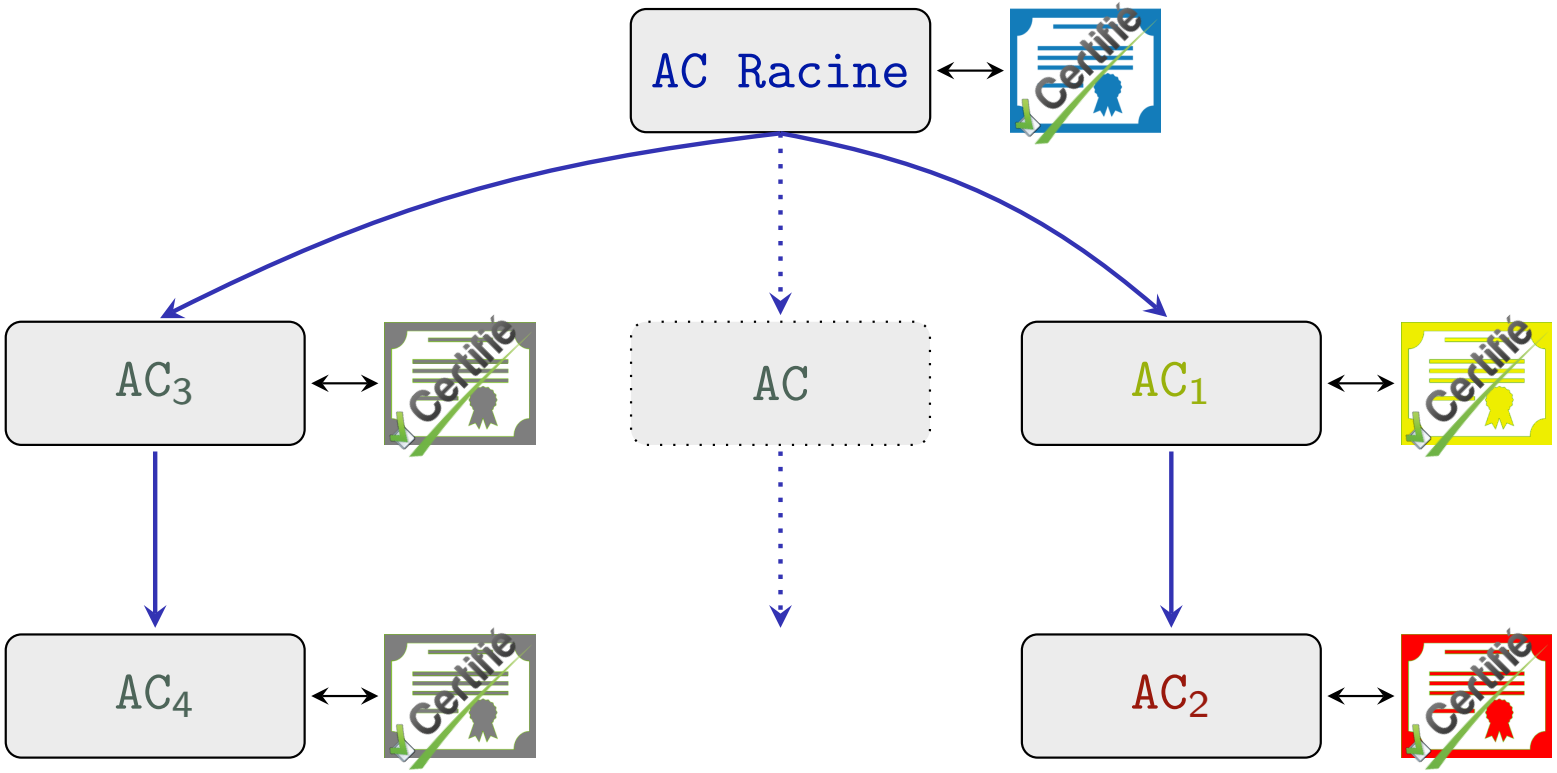
- Pour être **valide**, un certificat doit être **signé par une AC**.
- Une AC possède son propre couple (clé privée, clé publique) associé à un certificat qui peut être, soit **auto-signé** (on a vu précédemment comment créer un tel certificat), soit **signé par une autre AC**.
- N'importe qui peut se déclarer AC, ce qui pose évidemment le problème de la **confiance** en une AC. Or cette confiance est cruciale car la securité de ce système hiérarchique repose, pour une grande part, sur celle-ci.
- La confiance accordée à une AC est **héritée** par toutes les ACs filles. Autrement dit, faire confiance à une AC *racine*, implique que l'on accorde la même confiance à toutes les ACs qui en dérivent.
- Deux ACs peuvent s'entendre afin de signer chacune le certificat de l'autre : c'est la notion de **confiance croisée**.
- L'AC est **garante** des informations contenues dans les certificats qu'elle délivre.

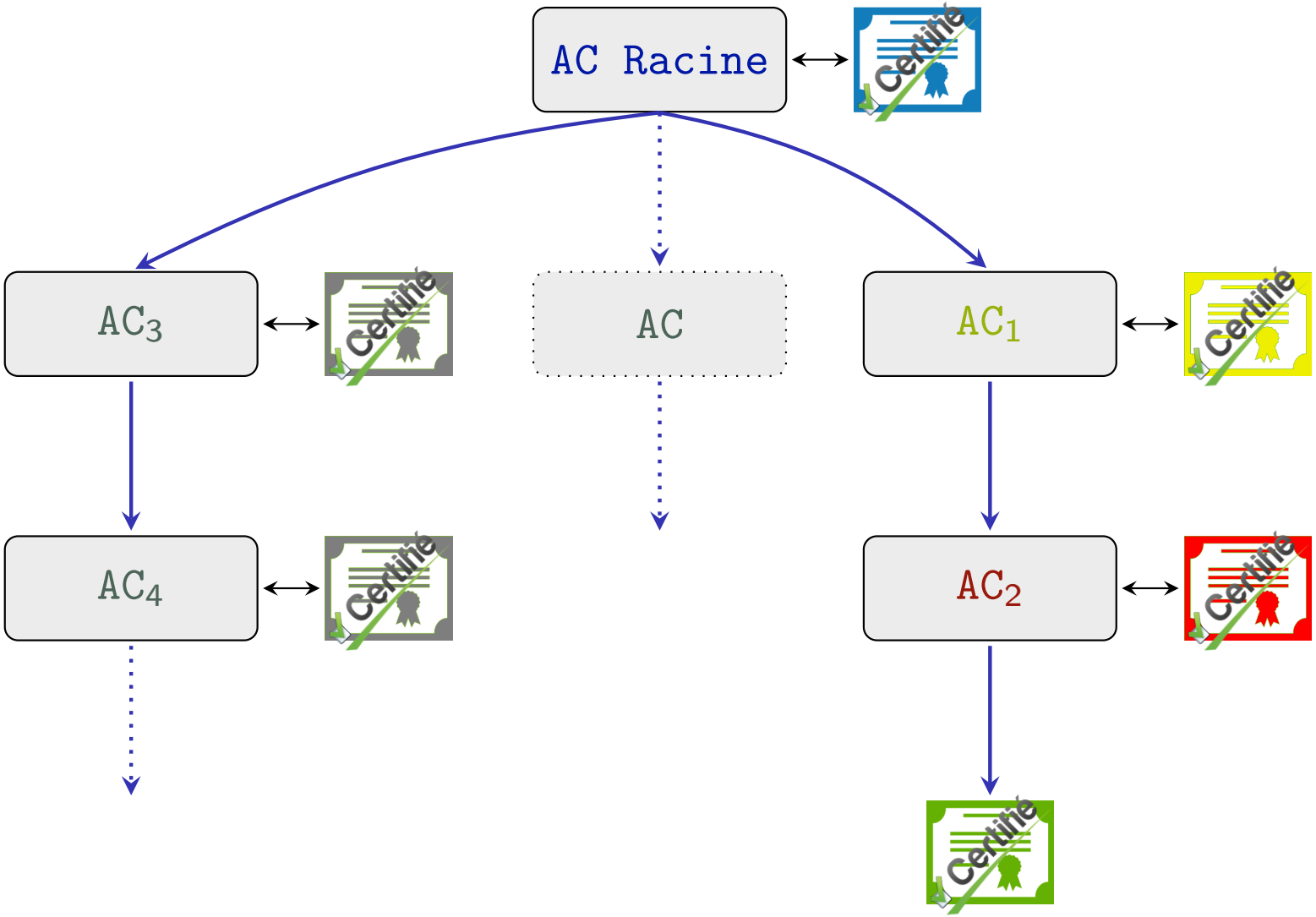
Principes

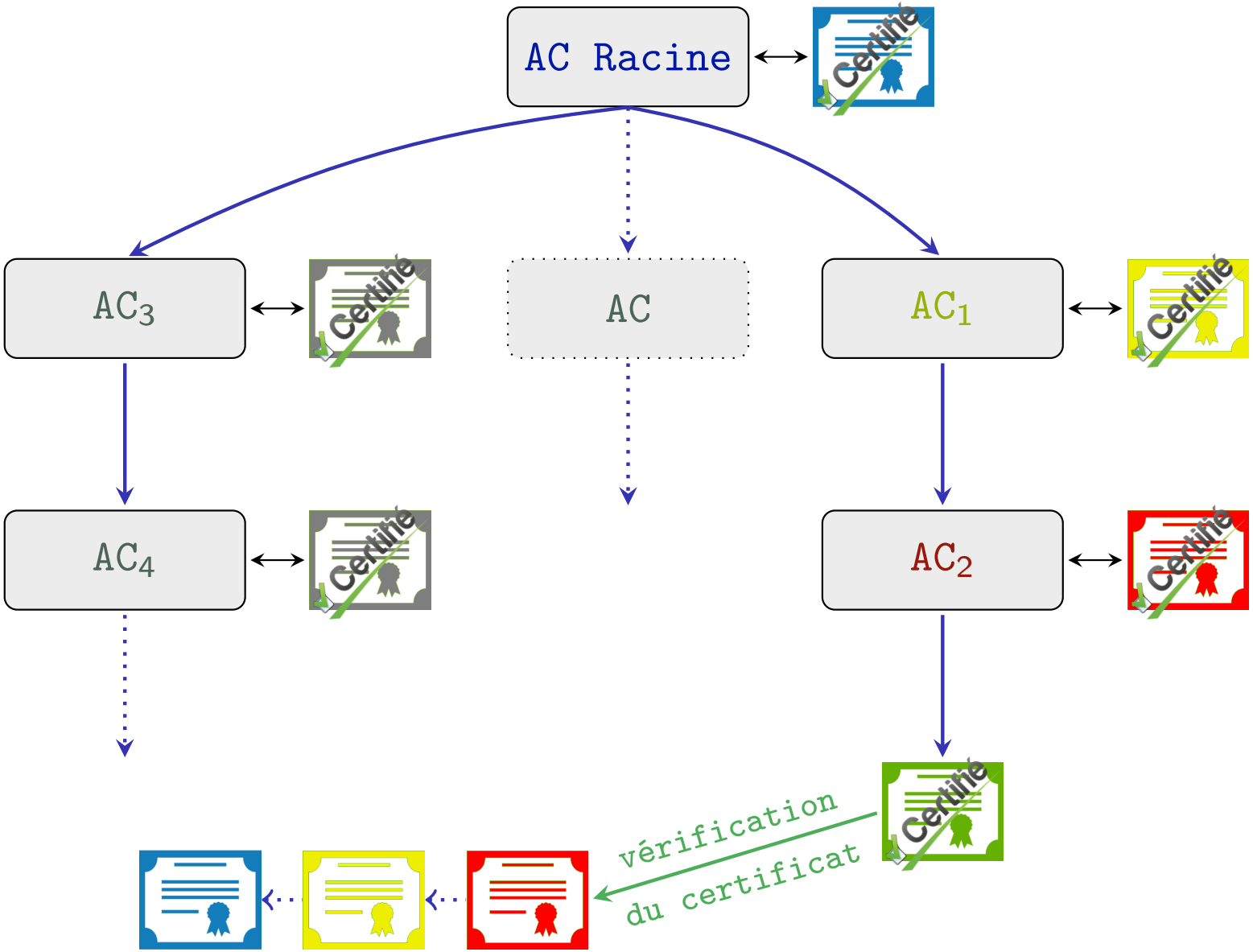
- Pour être **valide**, un certificat doit être **signé par une AC**.
- Une AC possède son propre couple (clé privée, clé publique) associé à un certificat qui peut être, soit **auto-signé** (on a vu précédemment comment créer un tel certificat), soit **signé par une autre AC**.
- N'importe qui peut se déclarer AC, ce qui pose évidemment le problème de la **confiance** en une AC. Or cette confiance est cruciale car la securité de ce système hiérarchique repose, pour une grande part, sur celle-ci.
- La confiance accordée à une AC est **héritée** par toutes les ACs filles. Autrement dit, faire confiance à une AC *racine*, implique que l'on accorde la même confiance à toutes les ACs qui en dérivent.
- Deux ACs peuvent s'entendre afin de signer chacune le certificat de l'autre : c'est la notion de **confiance croisée**.
- L'AC est **garante** des informations contenues dans les certificats qu'elle délivre.

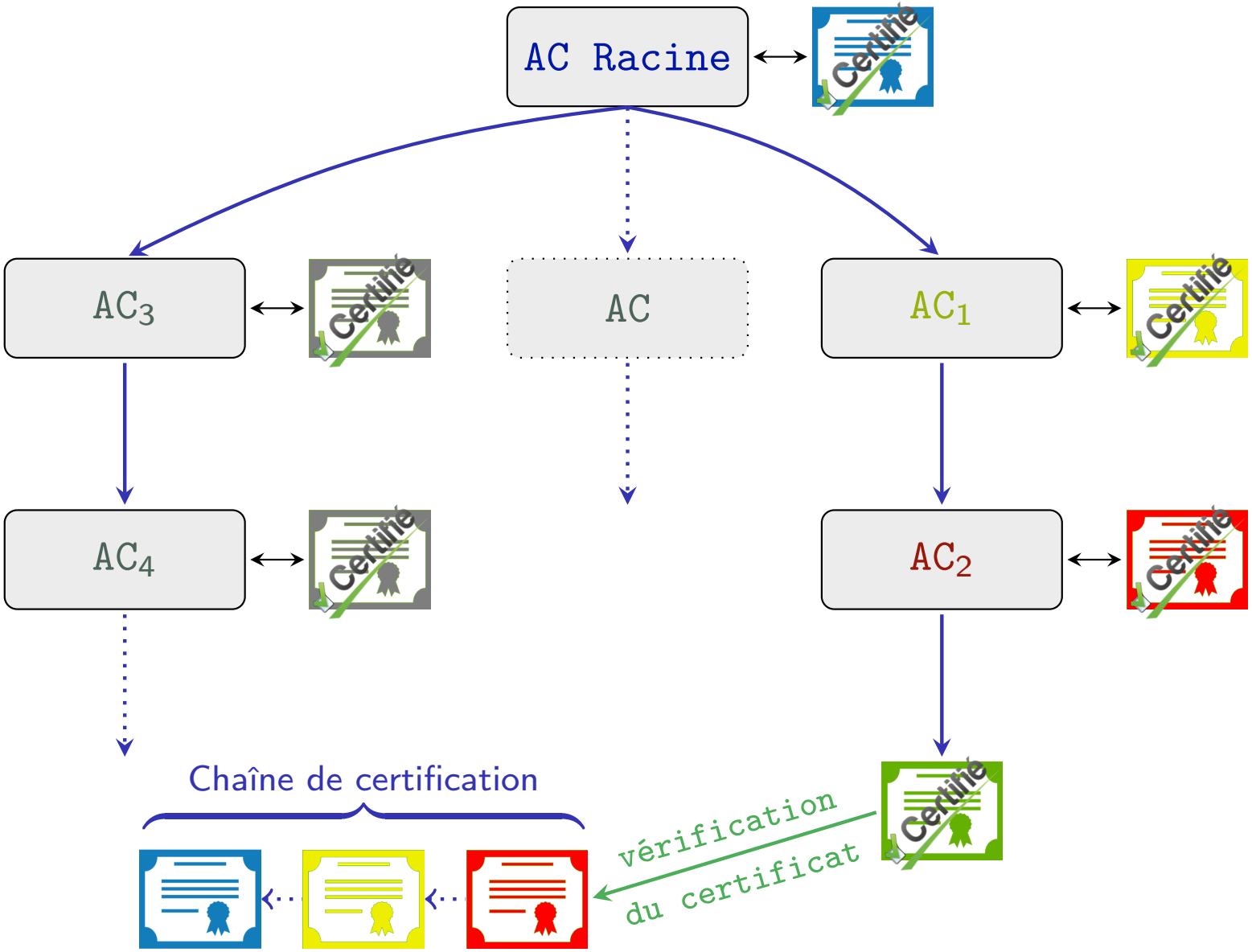

AC Racine
Certifie


AC Racine
Certifié
AC$_3$
Certifié
AC
AC$_1$
Certifié


AC Racine
Certifie
AC3
Certifie
AC
AC1
Certifie
AC4
Certifie
AC2
Certifie


AC Racine
Certifie
AC3
Certifie
AC
AC1
Certifie
AC4
Certifie
AC2
Certifie
Certifie


AC Racine
Certifie
AC3
AC
AC1
AC4
AC2
vérification
du certificat


AC Racine
Certifié
AC3
AC
AC1
AC4
AC2
Chaîne de certification
vérification
du certificat

Création d'un certificat utilisateur avec OpenSSL :

On crée tout d'abord une requête sur la machine de l'utilisateur :

```
$ openssl genrsa -F4 -out pt.key 2048 # Création de la clé
$ openssl req -new -key pt.key -out pt_req.pem -days 365 \
    -subj /C=FR/CN=Participant/O=IRIF/
```

Création d'un certificat utilisateur avec OpenSSL :

On crée tout d'abord une requête sur la machine de l'utilisateur :

```
$ openssl genrsa -F4 -out pt.key 2048 # Création de la clé
$ openssl req -new -key pt.key -out pt_req.pem -days 365 \
    -subj /C=FR/CN=Participant/O=IRIF/
```

On transmet ensuite cette requête (fichier `pt_req.pem`) à l'AC qui signe la requête comme suit :

```
$ openssl x509 -req -in pt_req.pem -CA ca.pem -CAkey ca.key \
    -CAcreateserial -out pt.pem
```

Création d'un certificat utilisateur avec OpenSSL :

On crée tout d'abord une requête sur la machine de l'utilisateur :

```
$ openssl genrsa -F4 -out pt.key 2048 # Création de la clé
$ openssl req -new -key pt.key -out pt_req.pem -days 365 \
    -subj /C=FR/CN=Participant/O=IRIF/
```

On transmet ensuite cette requête (fichier `pt_req.pem`) à l'AC qui signe la requête comme suit :

```
$ openssl x509 -req -in pt_req.pem -CA ca.pem -CAkey ca.key \
    -CAcreateserial -out pt.pem
```

L'AC renvoie au requérant le certificat signé (fichier `pt.pem`).

Création d'un certificat utilisateur avec OpenSSL :

On crée tout d'abord une requête sur la machine de l'utilisateur :

```
$ openssl genrsa -F4 -out pt.key 2048 # Création de la clé
$ openssl req -new -key pt.key -out pt_req.pem -days 365 \
    -subj /C=FR/CN=Participant/O=IRIF/
```

On transmet ensuite cette requête (fichier pt_req.pem) à l'AC qui signe la requête comme suit :

```
$ openssl x509 -req -in pt_req.pem -CA ca.pem -CAkey ca.key \
    -CAcreateserial -out pt.pem
```

L'AC renvoie au requérant le certificat signé (fichier pt.pem). Cette procédure est minimale et peu réaliste pour une AC quelque peu conséquente. Il faut plutôt procéder ainsi :

```
$ openssl ca -in pt_req.pem -out pt.pem -config ca.conf
Using configuration from ca.conf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'FR'
commonName            :ASN.1 12:'Participant'
organizationName      :ASN.1 12:'IRIF'
Certificate is to be certified until Sep  8 16:09:56 2017 GMT
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit? [y/n]:y
Write out database with 1 new entries
Data Base Updated
```

Création d'un certificat utilisateur avec OpenSSL (suite) :

Dans la commande `openssl ca`, toute la complexité réside dans le fichier de configuration `ca.conf`. Un exemple minimaliste :

```
[ CA_default ]
dir           =/etc/certs/CA  # where everything is kept
certs         =$dir/certs     # where the issued certs are kept
crl_dir       =$dir/crl       # where the issued crl are kept
new_certs=$dir/newcerts       # default place for new certs
database      =$dir/index.txt # database index file.
```

Création d'un certificat utilisateur avec OpenSSL (suite) :

Dans la commande `openssl ca`, toute la complexité réside dans le fichier de configuration `ca.conf`. Un exemple minimaliste :

```
[ CA_default ]
dir           =/etc/certs/CA  # where everything is kept
certs         =$dir/certs     # where the issued certs are kept
crl_dir       =$dir/crl       # where the issued crl are kept
new_certs=$dir/newcerts       # default place for new certs
database      =$dir/index.txt # database index file.
```

Il faut aussi créer les fichiers et catalogues nécessaires (liste non exhaustive) :

```
$ mkdir -p /etc/certs/CA
$ cd /etc/certs/CA
$ mkdir certs crl newcerts private
$ echo "01" > serial
$ touch index.txt
$ ...
```

Création d'un certificat utilisateur avec OpenSSL (suite) :

Dans la commande `openssl ca`, toute la complexité réside dans le fichier de configuration `ca.conf`. Un exemple minimaliste :

```
[ CA_default ]
dir           =/etc/certs/CA  # where everything is kept
certs         =$dir/certs     # where the issued certs are kept
crl_dir       =$dir/crl       # where the issued crl are kept
new_certs=$dir/newcerts       # default place for new certs
database      =$dir/index.txt # database index file.
```

Il faut aussi créer les fichiers et catalogues nécessaires (liste non exhaustive) :

```
$ mkdir -p /etc/certs/CA
$ cd /etc/certs/CA
$ mkdir certs crl newcerts private
$ echo "01" > serial
$ touch index.txt
$ ...
```

En pratique, on adapte le fichier fourni avec la distribution OpenSSL : `openssl.cnf`. C'est un fichier de taille conséquente (environ 350 lignes, commentaires compris). Fort heureusement, l'adaptation à sa propre AC ne nécessite en général que relativement peu de modifications.

- Une définition formelle : une IGC (*Public Key Infrastructure - PKI*) est un ensemble de personnes, matériels et logiciels, régis par des règles et des procédures et permettant de **créer**, **gérer** et **distribuer** des certificats X.509.
- C'est donc à la fois une entité **administrative** et une entité **technique** qui aura en charge l'ensemble des procédures concernant les certificats dont elle a la responsabilité.
- Plus précisément, une IGC assure les missions suivantes :
  - création et révocation des certificats X.509,
  - diffusion et publication des certificats X.509 (via un annuaire LDAP par exemple),
  - plus rarement, un service de séquestre et de recouvrement des clés privées. Ce service est bien sûr très utile en cas de perte de la clé privée de votre certificat. Le problème est que votre clé privée perd toute **légitimité** car vous la partagez avec un tiers (la confieriez-vous au service de sequestre de la NSA par exemple ?).

- Une définition formelle : une IGC (*Public Key Infrastructure - PKI*) est un ensemble de personnes, matériels et logiciels, régis par des règles et des procédures et permettant de **créer**, **gérer** et **distribuer** des certificats X.509.
- C'est donc à la fois une entité **administrative** et une entité **technique** qui aura en charge l'ensemble des procédures concernant les certificats dont elle a la responsabilité.
- Plus précisément, une IGC assure les missions suivantes :
  - création et révocation des certificats X.509,
  - diffusion et publication des certificats X.509 (via un annuaire LDAP par exemple),
  - plus rarement, un service de séquestre et de recouvrement des clés privées. Ce service est bien sûr très utile en cas de perte de la clé privée de votre certificat. Le problème est que votre clé privée perd toute **légitimité** car vous la partagez avec un tiers (la confieriez-vous au service de sequestre de la NSA par exemple ?).

- Une définition formelle : une IGC (*Public Key Infrastructure - PKI*) est un ensemble de personnes, matériels et logiciels, régis par des règles et des procédures et permettant de **créer**, **gérer** et **distribuer** des certificats X.509.
- C'est donc à la fois une entité **administrative** et une entité **technique** qui aura en charge l'ensemble des procédures concernant les certificats dont elle a la responsabilité.
- Plus précisément, une IGC assure les missions suivantes :
  - création et révocation des certificats X.509,
  - diffusion et publication des certificats X.509 (via un annuaire LDAP par exemple),
  - plus rarement, un service de séquestre et de recouvrement des clés privées. Ce service est bien sûr très utile en cas de perte de la clé privée de votre certificat. Le problème est que votre clé privée perd toute **légitimité** car vous la partagez avec un tiers (la confieriez-vous au service de sequestre de la NSA par exemple ?).

- Une définition formelle : une IGC (*Public Key Infrastructure - PKI*) est un ensemble de personnes, matériels et logiciels, régis par des règles et des procédures et permettant de **créer**, **gérer** et **distribuer** des certificats X.509.
- C'est donc à la fois une entité **administrative** et une entité **technique** qui aura en charge l'ensemble des procédures concernant les certificats dont elle a la responsabilité.
- Plus précisément, une IGC assure les missions suivantes :
  - création et révocation des certificats X.509,
  - diffusion et publication des certificats X.509 (via un annuaire LDAP par exemple),
  - plus rarement, un service de séquestre et de recouvrement des clés privées. Ce service est bien sûr très utile en cas de perte de la clé privée de votre certificat. Le problème est que votre clé privée perd toute **légitimité** car vous la partagez avec un tiers (la confieriez-vous au service de sequestre de la NSA par exemple ?).

- Une définition formelle : une IGC (*Public Key Infrastructure - PKI*) est un ensemble de personnes, matériels et logiciels, régis par des règles et des procédures et permettant de **créer**, **gérer** et **distribuer** des certificats X.509.
- C'est donc à la fois une entité **administrative** et une entité **technique** qui aura en charge l'ensemble des procédures concernant les certificats dont elle a la responsabilité.
- Plus précisément, une IGC assure les missions suivantes :
  - création et révocation des certificats X.509,
  - diffusion et publication des certificats X.509 (via un annuaire LDAP par exemple),
  - plus rarement, un service de séquestre et de recouvrement des clés privées. Ce service est bien sûr très utile en cas de perte de la clé privée de votre certificat. Le problème est que votre clé privée perd toute **légitimité** car vous la partagez avec un tiers (la confieriez-vous au service de sequestre de la NSA par exemple ?).

- Une définition formelle : une IGC (*Public Key Infrastructure - PKI*) est un ensemble de personnes, matériels et logiciels, régis par des règles et des procédures et permettant de **créer**, **gérer** et **distribuer** des certificats X.509.
- C'est donc à la fois une entité **administrative** et une entité **technique** qui aura en charge l'ensemble des procédures concernant les certificats dont elle a la responsabilité.
- Plus précisément, une IGC assure les missions suivantes :
  - création et révocation des certificats X.509,
  - diffusion et publication des certificats X.509 (via un annuaire LDAP par exemple),
  - plus rarement, un service de séquestre et de recouvrement des clés privées. Ce service est bien sûr très utile en cas de perte de la clé privée de votre certificat. Le problème est que votre clé privée perd toute **légitimité** car vous la partagez avec un tiers (la confieriez-vous au service de sequestre de la NSA par exemple ?).

Les éléments constitutifs d'une IGC

- Autorité d'Enregistrement (AE) : elle reçoit et traite les demandes de création, renouvellement et révocation de certificats. Elle doit notamment **s'assurer de l'identité** des demandeurs.
- Opérateur de Certification (OC) : il effectue toutes les opérations demandées par l'AC nécessitant la *clé privée* de celle-ci. Il n'est en principe **pas connecté** au réseau.
- Service de Publication (SP) : il met à disposition de tous, via un annuaire (le plus souvent LDAP), les certificats issus de l'IGC, le certificat de l'AC et éventuellement les listes de révocation (CRL).
- Service de Validation (SV) : il permet à tout utilisateur de vérifier la validité d'un certificat (expiration, vol/perte de la clé privée associée,...).
- Service de Séquestre (SS) : il stocke au sein de l'IGC les couples (clé privée, clé publique) des certificats produits. Dangereux !

Les éléments constitutifs d'une IGC

- Autorité d'Enregistrement (AE) : elle reçoit et traite les demandes de création, renouvellement et révocation de certificats. Elle doit notamment **s'assurer de l'identité** des demandeurs.
- Opérateur de Certification (OC) : il effectue toutes les opérations demandées par l'AC nécessitant la *clé privée* de celle-ci. Il n'est en principe **pas connecté** au réseau.
- Service de Publication (SP) : il met à disposition de tous, via un annuaire (le plus souvent LDAP), les certificats issus de l'IGC, le certificat de l'AC et éventuellement les listes de révocation (CRL).
- Service de Validation (SV) : il permet à tout utilisateur de vérifier la validité d'un certificat (expiration, vol/perte de la clé privée associée,...).
- Service de Séquestre (SS) : il stocke au sein de l'IGC les couples (clé privée, clé publique) des certificats produits. Dangereux !

Les éléments constitutifs d'une IGC

- Autorité d'Enregistrement (AE) : elle reçoit et traite les demandes de création, renouvellement et révocation de certificats. Elle doit notamment **s'assurer de l'identité** des demandeurs.
- Opérateur de Certification (OC) : il effectue toutes les opérations demandées par l'AC nécessitant la *clé privée* de celle-ci. Il n'est en principe **pas connecté** au réseau.
- Service de Publication (SP) : il met à disposition de tous, via un annuaire (le plus souvent LDAP), les certificats issus de l'IGC, le certificat de l'AC et éventuellement les listes de révocation (CRL).
- Service de Validation (SV) : il permet à tout utilisateur de vérifier la validité d'un certificat (expiration, vol/perte de la clé privée associée,...).
- Service de Séquestre (SS) : il stocke au sein de l'IGC les couples (clé privée, clé publique) des certificats produits. Dangereux !

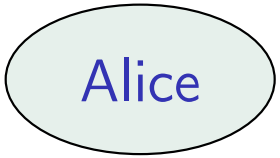
Les éléments constitutifs d'une IGC

- Autorité d'Enregistrement (AE) : elle reçoit et traite les demandes de création, renouvellement et révocation de certificats. Elle doit notamment **s'assurer de l'identité** des demandeurs.
- Opérateur de Certification (OC) : il effectue toutes les opérations demandées par l'AC nécessitant la *clé privée* de celle-ci. Il n'est en principe **pas connecté** au réseau.
- Service de Publication (SP) : il met à disposition de tous, via un annuaire (le plus souvent LDAP), les certificats issus de l'IGC, le certificat de l'AC et éventuellement les listes de révocation (CRL).
- Service de Validation (SV) : il permet à tout utilisateur de vérifier la validité d'un certificat (expiration, vol/perte de la clé privée associée,...).
- Service de Séquestre (SS) : il stocke au sein de l'IGC les couples (clé privée, clé publique) des certificats produits. Dangereux !

Les éléments constitutifs d'une IGC

- Autorité d'Enregistrement (AE) : elle reçoit et traite les demandes de création, renouvellement et révocation de certificats. Elle doit notamment **s'assurer de l'identité** des demandeurs.
- Opérateur de Certification (OC) : il effectue toutes les opérations demandées par l'AC nécessitant la *clé privée* de celle-ci. Il n'est en principe **pas connecté** au réseau.
- Service de Publication (SP) : il met à disposition de tous, via un annuaire (le plus souvent LDAP), les certificats issus de l'IGC, le certificat de l'AC et éventuellement les listes de révocation (CRL).
- Service de Validation (SV) : il permet à tout utilisateur de vérifier la validité d'un certificat (expiration, vol/perte de la clé privée associée,...).
- Service de Séquestre (SS) : il stocke au sein de l'IGC les couples (clé privée, clé publique) des certificats produits. Dangereux !

Demande de certificat

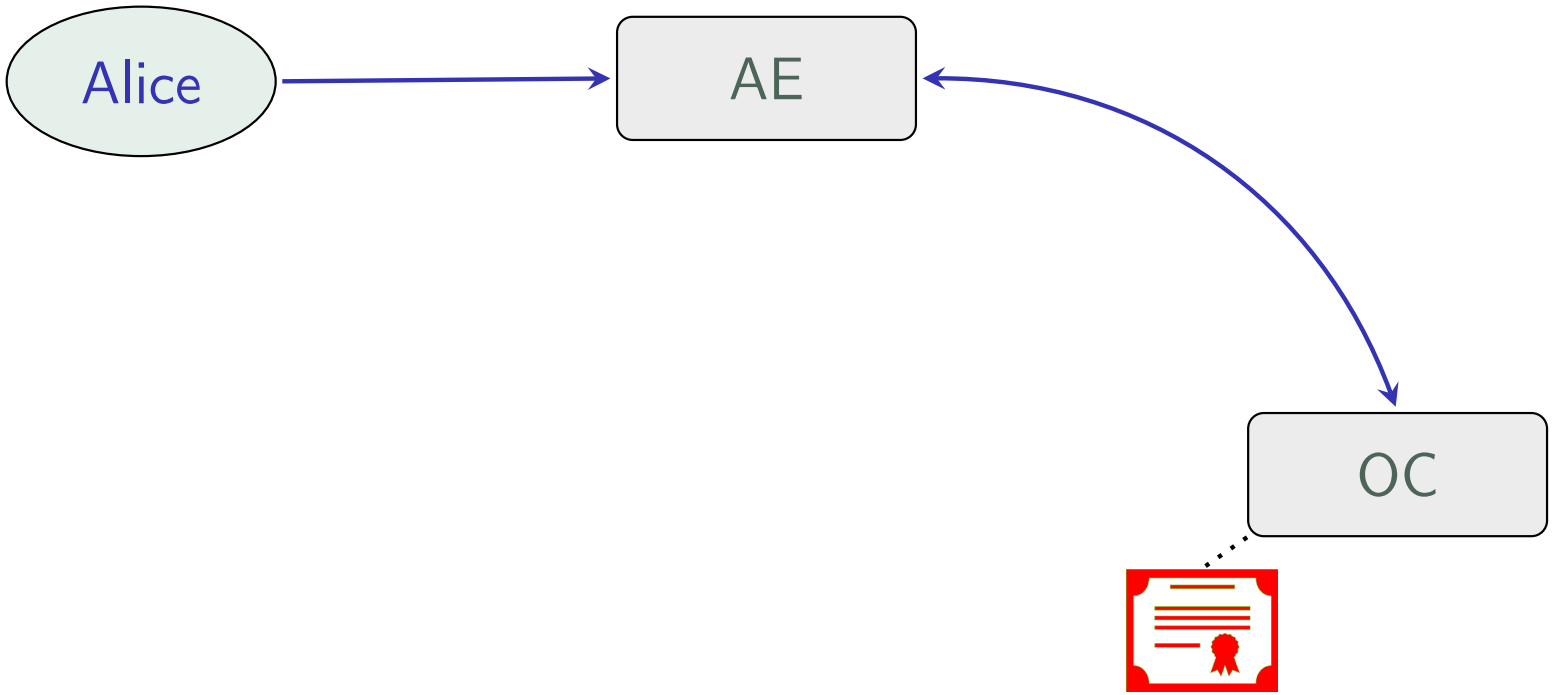


Alice souhaite obtenir un certificat

Demande de certificat

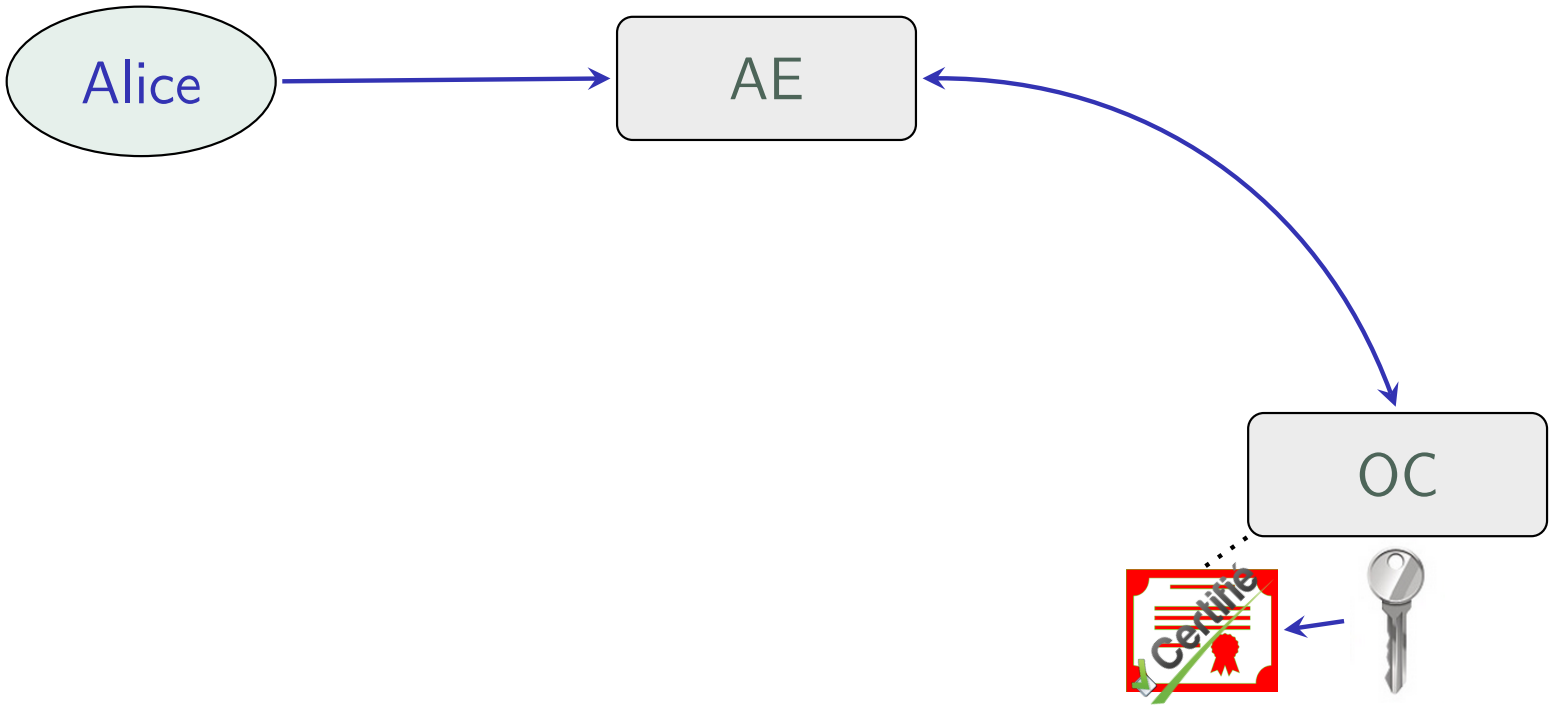
Alice envoie sa requête à l'AE

Demande de certificat



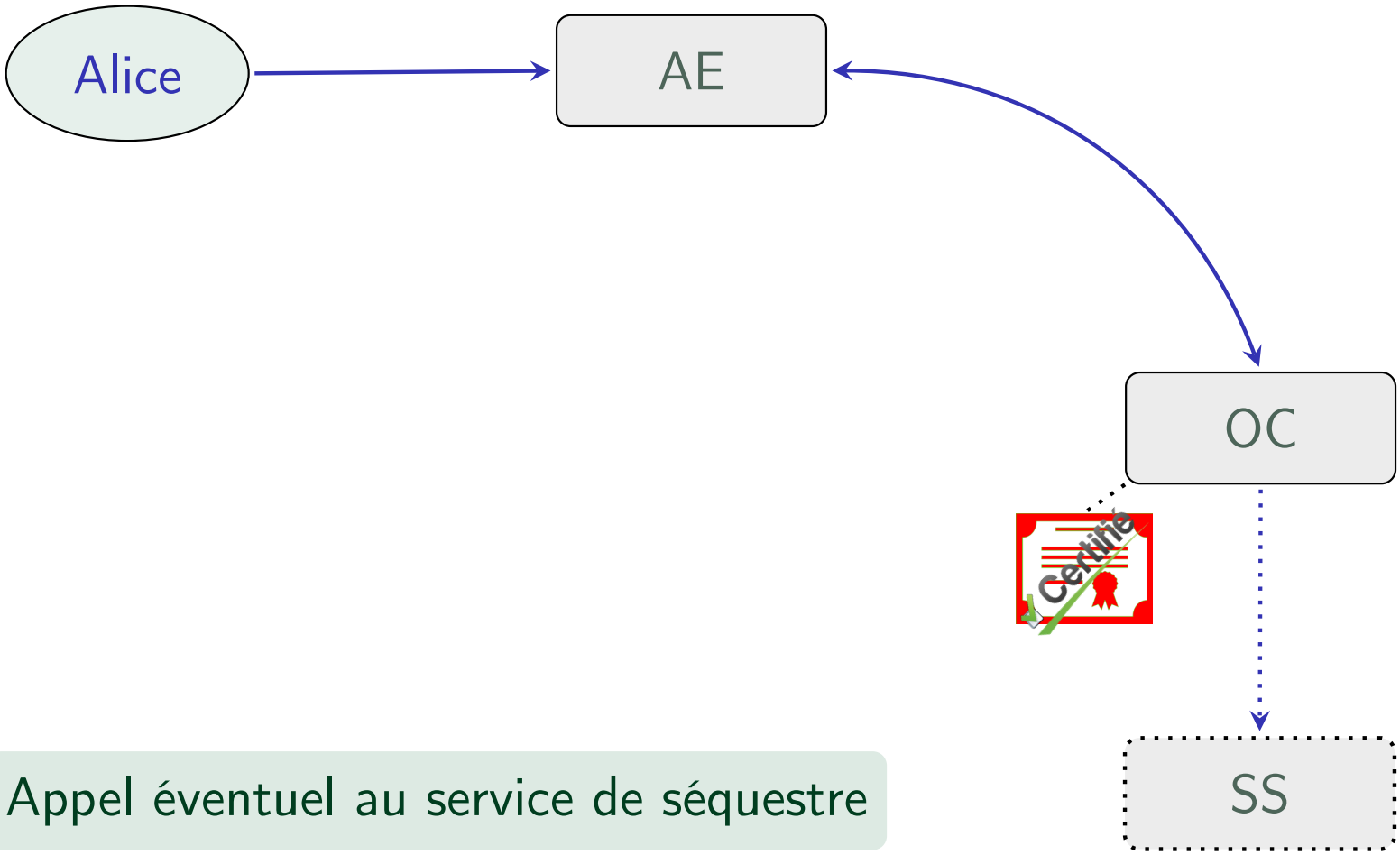
L'AE vérifie l'identité d'Alice et transmet à l'OC

Demande de certificat



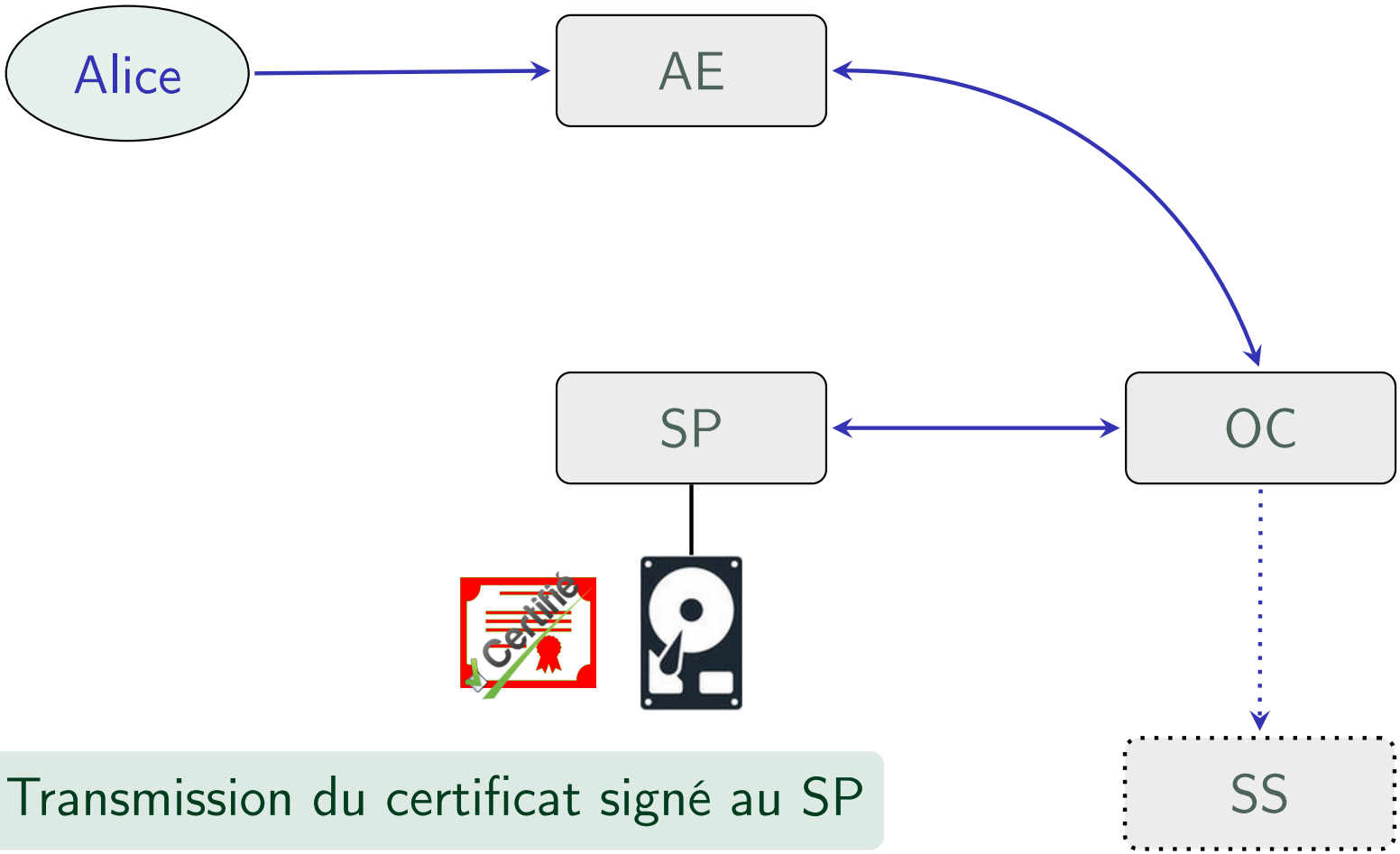
L'OC signe le certificat avec la clé privée de l'AC

Demande de certificat



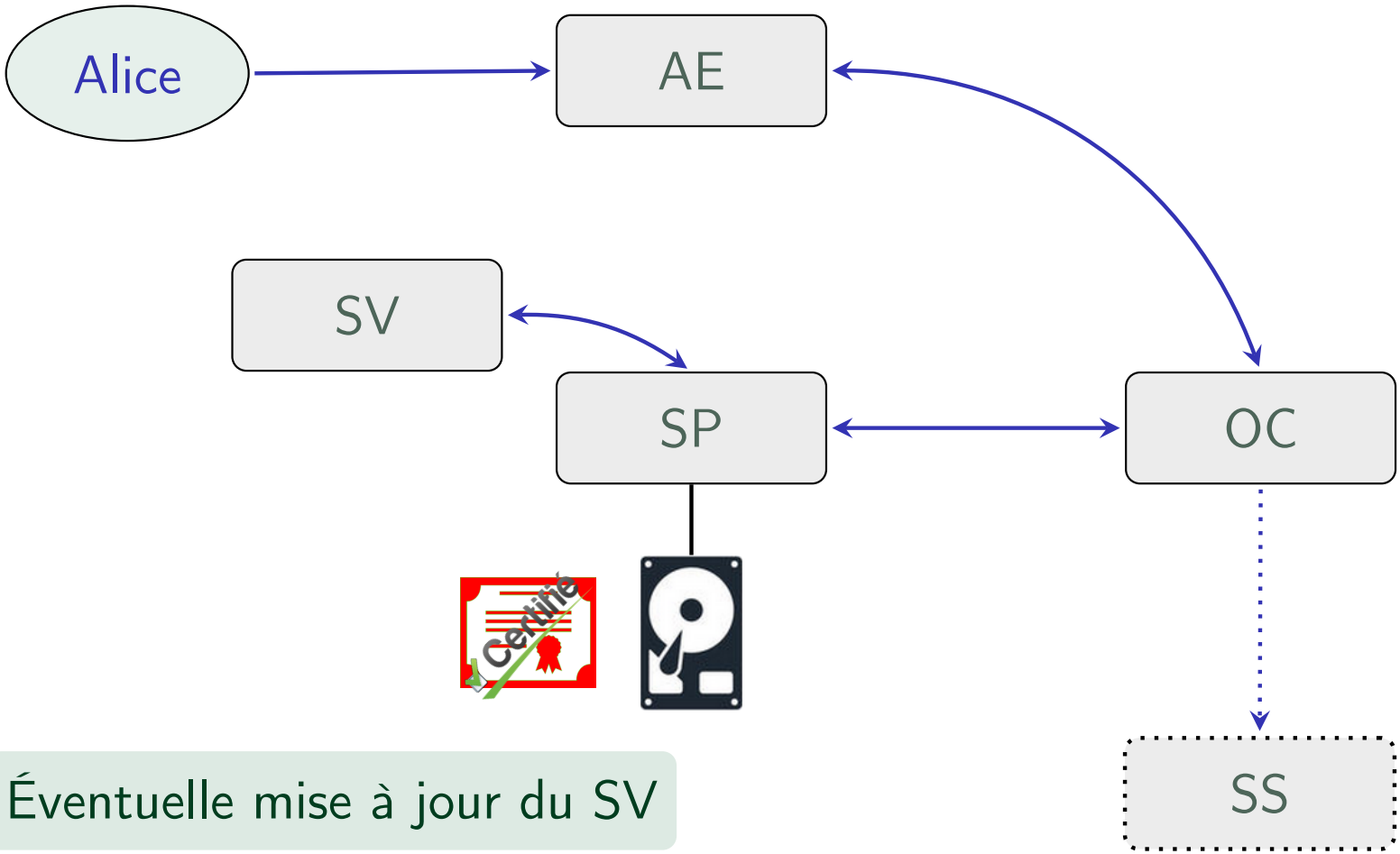
Appel éventuel au service de séquestre

Demande de certificat



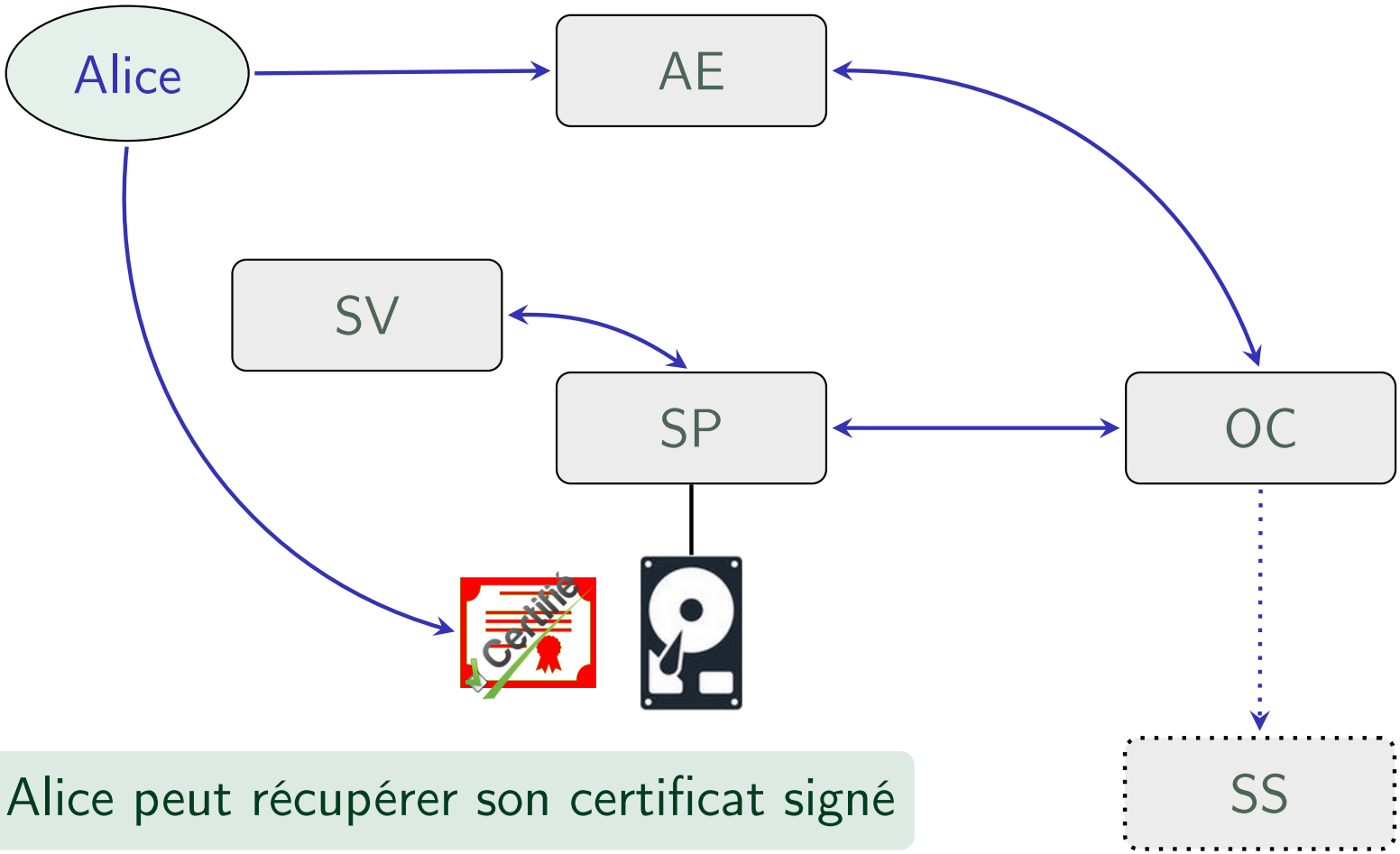
Transmission du certificat signé au SP

Demande de certificat



Éventuelle mise à jour du SV

Demande de certificat



Alice peut récupérer son certificat signé

- Le fichier spécial `/dev/random` est un générateur **pseudo-aléatoire** qui prend sa source d'aléa dans l'**environnement du système** (comme par exemple les périphériques). Il est de ce fait **bloquant** lorsqu'il n'y a pas assez d'activité sur le système.

  C'est le générateur recommandé pour la création de clés à usage cryptographique.

- Le fichier spécial `/dev/random` est un générateur **pseudo-aléatoire** qui prend sa source d'aléa dans l'**environnement du système** (comme par exemple les périphériques). Il est de ce fait **bloquant** lorsqu'il n'y a pas assez d'activité sur le système.

  C'est le générateur recommandé pour la création de clés à usage cryptographique.
- Le fichier spécial `/dev/urandom` est identique au précédent à cela près qu'il n'est **pas bloquant** (d'où son nom : *u* pour *unlimited*). Par conséquent, il est de **moins bonne qualité** puisqu'il n'attend pas qu'il y ait suffisamment d'entropie (d'activité).

  Néanmoins bien adapté pour la création d'un *nonce* (number only once).

- Le fichier spécial `/dev/random` est un générateur **pseudo-aléatoire** qui prend sa source d'aléa dans l'**environnement du système** (comme par exemple les périphériques). Il est de ce fait **bloquant** lorsqu'il n'y a pas assez d'activité sur le système.

  C'est le générateur recommandé pour la création de clés à usage cryptographique.
- Le fichier spécial `/dev/urandom` est identique au précédent à cela près qu'il n'est **pas bloquant** (d'où son nom : *u* pour *unlimited*). Par conséquent, il est de **moins bonne qualité** puisqu'il n'attend pas qu'il y ait suffisamment d'entropie (d'activité).

  Néanmoins bien adapté pour la création d'un *nonce* (number only once).
- Les implémentations **varient fortement** d'un système UNIX à l'autre. Par exemple sous FreeBSD, `/dev/random` n'est pas bloquant et `/dev/urandom` n'est qu'un lien sur ce dernier.

Pour générer un entier pseudo-aléatoire de 128 bits,

Pour générer un entier pseudo-aléatoire de 128 bits,

sur un système Unix :

```
$ dd if=/dev/random bs=1 count=16 2>/dev/null|xxd -ps
05c92eb83ff81581b938016ffe7f434b
```

Pour générer un entier pseudo-aléatoire de 128 bits,

sur un système Unix :

```
$ dd if=/dev/random bs=1 count=16 2>/dev/null|xxd -ps
05c92eb83ff81581b938016ffe7f434b
```

avec la commande openssl :

```
$ openssl rand -hex 16
3ab16bbcaf9b8ed839a9d4ba3e3412c1
```

Pour générer un entier pseudo-aléatoire de 128 bits,

sur un système Unix :

```
$ dd if=/dev/random bs=1 count=16 2>/dev/null|xxd -ps
05c92eb83ff81581b938016ffe7f434b
```

avec la commande openssl :

```
$ openssl rand -hex 16
3ab16bbcaf9b8ed839a9d4ba3e3412c1
```

avec la commande openssl (sur un système unix) :

```
$ openssl rand -hex -rand /dev/urandom 16
2048 semi-random bytes loaded
2bd4e1335df83568e9863c1cc0963b57
```

avec le langage Python :

```
>>> import random
>>> hex(random.randrange(2**128))
'0x39d4537b9561f4ffa4fa6b7638304508'
```

avec le langage Python :

```
>>> import random
>>> hex(random.randrange(2**128))
'0x39d4537b9561f4ffa4fa6b7638304508'
```

avec le langage C :

```
int main(void)
{
  int fd;
  unsigned char buf[16], *c;

  fd = open("/dev/random", O_RDONLY);
  (void) read(fd, (void *) buf, sizeof(buf));
  for (c = buf; c - buf < sizeof(buf); c++)
      fprintf(stdout, "%02x", *c);
  fprintf(stdout, "\n");
}
```

- Simon Singh, *L'Histoire des codes secrets : De l'Égypte des pharaons à l'ordinateur quantique*, 1999.
- Bruce Schneier, *Applied Cryptography, Second Edition : Protocols, Algorithms, and Source Code in C*, 1996.
- Alfred Menezes, Paul J. van Oorschot et Scott A. Vanstone, *Handbook of applied cryptography*, 1997. Une version électronique en ligne : `http://cacr.uwaterloo.ca/hac/`.
- Gilles Zémor, *Cours de cryptographie*, 2000.
- Joseph H. Silverman, *The Arithmetic of Elliptic Curves*, 1986.
- Neal Koblitz, *A Course in Number Theory and Cryptography*, 1994 (2ème édition).
- Toutes les RFCs, `https://www.rfc-editor.org/rfc-index.html`.
- Les normes publiées par le *National Institute of Standards and Technology (NIST)*, `https://www.nist.org`.