Introduction à la sécurité
Cours 3 – Cryptologie asymétrique

Julien Lavauzelle

Université Paris 8

Licence 3 Informatique et Vidéoludisme

18/09/2024

Ces slides sont en partie inspirées de celles de Pablo Rauzy, concepteur du cours.

Voir sa page web : `pablo.rauzy.name/teaching/is`

Pour aller plus loin dans ce domaine, j'enseigne un cours de M1 sur ce domaine.

Voir le cours de l'année dernière : `lvzl.fr/teaching/2023-24/cp.html`

Plan

Entiers modulaires

Entiers modulaires. La **division euclidienne** de l'entier naturel a par l'entier naturel $b \neq 0$ donner deux entiers, le **quotient** q et le **reste** r, qui satisfont les équations suivantes :

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

La valeur du reste r est aussi appelée « a **modulo** b », et est notée $a \mod b$.

Entiers modulaires

Entiers modulaires. La **division euclidienne** de l'entier naturel a par l'entier naturel $b \neq 0$ donner deux entiers, le **quotient** q et le **reste** r, qui satisfont les équations suivantes :

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

La valeur du reste r est aussi appelée « a **modulo** b », et est notée $a \mod b$.

Structure algébrique. Si l'on fixe un entier N, alors on peut ajouter, soustraire et multiplier tous les restes modulo N : cela se passe comme si on faisait d'abord les calculs sur les entiers, puis on en prenait les restes. On parle de l'**anneau quotient** des restes modulo N, noté $\mathbb{Z}/N\mathbb{Z}$:

- **anneau** : structure algébrique qui permet la somme, l'opposé, la multiplication ;
- **quotient** : issu d'une relation, la réduction modulaire.

Tables de calcul

Exemples :

1. les entiers modulo 24 représentent les heures de la journée ;
2. les bits $\{0, 1\}$ peuvent être représentés par $\mathbb{Z}/2\mathbb{Z}$: la somme correspond au XOR, et la multiplication au AND.

Tables de calcul

Exemples :

1. les entiers modulo 24 représentent les heures de la journée ;
2. les bits $\{0, 1\}$ peuvent être représentés par $\mathbb{Z}/2\mathbb{Z}$: la somme correspond au XOR, et la multiplication au AND.

Pour $N = 10$, voici les tables d'addition et de multiplication :

$+$	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

$\times$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

Et l'inversion et la division ?

Question 1. Peut-on prendre l'opposé/soustraire des entiers modulaires ?

Et l'inversion et la division ?

Question 1. Peut-on prendre l'opposé/soustraire des entiers modulaires ?
$\rightarrow$ **oui !** $\rightarrow$ les trouver à l'aide la table

Et l'inversion et la division ?

Question 1. Peut-on prendre l'opposé/soustraire des entiers modulaires ?
$\rightarrow$ **oui !** $\rightarrow$ les trouver à l'aide la table

Question 2. Peut-on inverser/diviser des entiers modulaires ?

Et l'inversion et la division ?

Question 1. Peut-on prendre l'opposé/soustraire des entiers modulaires ?
→ **oui !** → les trouver à l'aide la table

Question 2. Peut-on inverser/diviser des entiers modulaires ?
→ **pas toujours...**

Et l'inversion et la division ?

Question 1. Peut-on prendre l'opposé/soustraire des entiers modulaires ?
$\rightarrow$ **oui !** $\rightarrow$ les trouver à l'aide la table

Question 2. Peut-on inverser/diviser des entiers modulaires ?
$\rightarrow$ **pas toujours...**

Trouvez tous les entiers modulo 10 qu'on ne peut pas inverser (c'est-à-dire, les entiers x tels que pour n'importe quel y, on n'a jamais $x \times y = 1$ modulo 10).

Et l'inversion et la division ?

Question 1. Peut-on prendre l'opposé/soustraire des entiers modulaires ?
$\rightarrow$ **oui !** $\rightarrow$ les trouver à l'aide la table

Question 2. Peut-on inverser/diviser des entiers modulaires ?
$\rightarrow$ **pas toujours...**

Trouvez tous les entiers modulo 10 qu'on ne peut pas inverser (c'est-à-dire, les entiers x tels que pour n'importe quel y, on n'a jamais $x \times y = 1$ modulo 10).

En général, seuls les entiers **premiers avec** $N = 10$ peuvent être inversés.

Calcul effectif de l'inverse modulaire

On appelle **groupe des inversibles**, noté $(\mathbb{Z}/N\mathbb{Z})^\times$ l'ensemble des inversibles modulo N compris entre 0 et $N-1$. C'est un **groupe**, dans le sens où on peut multiplier et inverser (mais pas additionner) sans sortir de l'ensemble.

Calcul effectif de l'inverse modulaire

On appelle **groupe des inversibles**, noté $(\mathbb{Z}/N\mathbb{Z})^\times$ l'ensemble des inversibles modulo N compris entre 0 et $N-1$. C'est un **groupe**, dans le sens où on peut multiplier et inverser (mais pas additionner) sans sortir de l'ensemble.

Exemple : pour $N = 9$, $a = 2$ et $b = 7$ sont inversibles, et $ab = 14 \equiv 5 \mod 9$ l'est aussi. En revanche, $a + b = 9 \equiv 0$ n'est pas inversible.

Calcul effectif de l'inverse modulaire

On appelle **groupe des inversibles**, noté $(\mathbb{Z}/N\mathbb{Z})^\times$ l'ensemble des inversibles modulo N compris entre 0 et $N-1$. C'est un **groupe**, dans le sens où on peut multiplier et inverser (mais pas additionner) sans sortir de l'ensemble.

Exemple : pour $N = 9$, $a = 2$ et $b = 7$ sont inversibles, et $ab = 14 \equiv 5 \mod 9$ l'est aussi. En revanche, $a + b = 9 \equiv 0$ n'est pas inversible.

Algorithme d'inversion modulaire. Pour inverser x modulo N, on effectue l'**algorithme d'Euclide étendu** (celui pour trouver le PGCD) en cherchant les coefficients de Bezout a et b correspondant à x et N. On obtient :

$$a \cdot x + b \cdot N = 1$$

Si on réduit cette équation modulo N, on a alors

$$a \cdot x \equiv 1 \mod N$$

donc a est l'inverse de x modulo N.

Indicatrice d'Euler

Le nombre d'entiers compris entre 0 et $N-1$, qui sont inversibles modulo N, est noté $\phi(N)$. C'est **l'indicatrice d'Euler** :

$$\phi(N) := \#\{i \in [0, N-1], \mathrm{pgcd}(i, N) = 1\}\,.$$

Indicatrice d'Euler

Le nombre d'entiers compris entre 0 et $N-1$, qui sont inversibles modulo N, est noté $\phi(N)$. C'est **l'indicatrice d'Euler** :

$$\phi(N) := \#\{i \in [0, N-1], \operatorname{pgcd}(i, N) = 1\}\,.$$

Remarque. Il existe une formule pour calculer $\phi(N)$ **si on connaît la factorisation** de N en produit de nombres premiers.

$$N = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k} \quad \Longrightarrow \quad \phi(N) = (p_1 - 1)p_1^{e_1 - 1}(p_2 - 1)p_2^{e_2 - 1} \cdots (p_k - 1)p_k^{e_k - 1}$$

Indicatrice d'Euler

Le nombre d'entiers compris entre 0 et $N-1$, qui sont inversibles modulo N, est noté $\phi(N)$. C'est **l'indicatrice d'Euler** :

$$\phi(N) := \#\{i \in [0, N-1], \operatorname{pgcd}(i, N) = 1\}\,.$$

Remarque. Il existe une formule pour calculer $\phi(N)$ **si on connaît la factorisation** de N en produit de nombres premiers.

$$N = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k} \quad \Longrightarrow \quad \phi(N) = (p_1 - 1)p_1^{e_1 - 1}(p_2 - 1)p_2^{e_2 - 1} \cdots (p_k - 1)p_k^{e_k - 1}$$

Important. Dans le contexte qui va suivre, on aura $N = pq$, et donc $\phi(N) = (p-1)(q-1)$.

Indicatrice d'Euler

Le nombre d'entiers compris entre 0 et $N-1$, qui sont inversibles modulo N, est noté $\phi(N)$. C'est **l'indicatrice d'Euler** :

$$\phi(N) := \#\{i \in [0, N-1], \operatorname{pgcd}(i, N) = 1\}\,.$$

Remarque. Il existe une formule pour calculer $\phi(N)$ **si on connaît la factorisation** de N en produit de nombres premiers.

$$N = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k} \quad \Longrightarrow \quad \phi(N) = (p_1 - 1)p_1^{e_1 - 1}(p_2 - 1)p_2^{e_2 - 1} \cdots (p_k - 1)p_k^{e_k - 1}$$

Important. Dans le contexte qui va suivre, on aura $N = pq$, et donc $\phi(N) = (p-1)(q-1)$.

Théorème d'Euler. Pour tout entier a premier avec N, on a

$$a^{\phi(N)} \equiv 1 \mod N\,.$$

Théorème des restes chinois

Théorème des restes chinois (reformulé simplement). Si $N = pq$ avec p et q premiers entre eux, alors il existe une **bijection**, que l'on sait calculer **efficacement**, entre les restes modulo N et ceux modulo p et q.

Théorème des restes chinois

Théorème des restes chinois (reformulé simplement). Si $N = pq$ avec p et q premiers entre eux, alors il existe une **bijection**, que l'on sait calculer **efficacement**, entre les restes modulo N et ceux modulo p et q.

Autrement dit, pour $x \in \mathbb{Z}$ un entier quelconque (inconnu) :

1. **(sens facile)** Si on connaît $x_N := x \mod N$, alors on peut calculer efficacement $x_p := x \mod p$ et $x_q := x \mod q$. Pour cela, on calcule respectivement $x_N \mod p$ et $x_N \mod q$.
2. **(sens moins facile)** Si on connaît x_p et x_q, on peut aussi retouver x_N ! L'opération est plus complexe. D'abord on cherche les coefficients de Bezout de p et q :

$$a \cdot p + b \cdot q = 1\,.$$

Puis, on calcule :

$$x = (a \cdot p \cdot x_q) + (b \cdot q \cdot x_p) \mod N$$

Théorème des restes chinois

Théorème des restes chinois (reformulé simplement). Si $N = pq$ avec p et q premiers entre eux, alors il existe une **bijection**, que l'on sait calculer **efficacement**, entre les restes modulo N et ceux modulo p et q.

Autrement dit, pour $x \in \mathbb{Z}$ un entier quelconque (inconnu) :

1. **(sens facile)** Si on connaît $x_N := x \mod N$, alors on peut calculer efficacement $x_p := x \mod p$ et $x_q := x \mod q$. Pour cela, on calcule respectivement $x_N \mod p$ et $x_N \mod q$.
2. **(sens moins facile)** Si on connaît x_p et x_q, on peut aussi retouver x_N ! L'opération est plus complexe. D'abord on cherche les coefficients de Bezout de p et q :

$$a \cdot p + b \cdot q = 1 .$$

Puis, on calcule :

$$x = (a \cdot p \cdot x_q) + (b \cdot q \cdot x_p) \mod N$$

Par ailleurs, toutes les transformations ci-dessus passent à la somme et au produit. On parle **d'isomorphisme d'anneaux**.

Plan

Plan

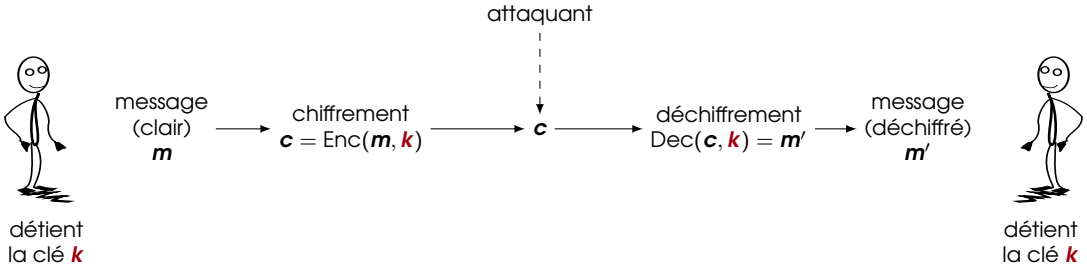
Rappel : le chiffrement symétrique

Rappel : le chiffrement **symétrique**, ou **à clé secrète**.

Les protagonistes (Alice et Bob) possèdent un **secret commun** : la clé secrète k.

De manière **symétrique**, ils utilisent cette clé, à la fois pour masquer l'information **et** pour la rendre de nouveau intelligible.

→ *Exemple pour le chiffrement :*



Inconvénients de la cryptographie symétrique

Quelques **inconvénients** liés à la cryptographie symétrique.

Inconvénients de la cryptographie symétrique

Quelques **inconvénients** liés à la cryptographie symétrique.

1. Si Alice et Bob ne se sont jamais rencontrés, comment peuvent-ils **mettre en place un secret commun** ?

Inconvénients de la cryptographie symétrique

Quelques **inconvénients** liés à la cryptographie symétrique.

1. Si Alice et Bob ne se sont jamais rencontrés, comment peuvent-ils **mettre en place un secret commun** ?
2. Comment **engager** une conversation avec une entité inconnue ?
   $\rightarrow$ exemple : pour le paiement en ligne, vous envoyez vos données à une nouvelle autorité bancaire

Inconvénients de la cryptographie symétrique

Quelques **inconvénients** liés à la cryptographie symétrique.

1. Si Alice et Bob ne se sont jamais rencontrés, comment peuvent-ils **mettre en place un secret commun** ?
2. Comment **engager** une conversation avec une entité inconnue ?
   → exemple : pour le paiement en ligne, vous envoyez vos données à une nouvelle autorité bancaire
3. Dans un réseau à n participants, il y a **une clé** à stocker **par paire** de participants, c'est-à-dire

$$\frac{n(n-1)}{2} \text{ clés}$$

   → pour 1000 participants, $\simeq$ 500 000 clés à stocker...
   → pour une université à 30 000 étudiants et personnels : $\simeq$ 450 000 000 clés

Inconvénients de la cryptographie symétrique

Quelques **inconvénients** liés à la cryptographie symétrique.

1. Si Alice et Bob ne se sont jamais rencontrés, comment peuvent-ils **mettre en place un secret commun** ?
2. Comment **engager** une conversation avec une entité inconnue ?
   → exemple : pour le paiement en ligne, vous envoyez vos données à une nouvelle autorité bancaire
3. Dans un réseau à n participants, il y a **une clé** à stocker **par paire** de participants, c'est-à-dire

$$\frac{n(n-1)}{2} \text{ clés}$$

   → pour 1000 participants, $\simeq$ 500 000 clés à stocker...
   → pour une université à 30 000 étudiants et personnels : $\simeq$ 450 000 000 clés
4. **Gestion de clés difficile :** ajout de nouveaux participants, perte de clés
   → on a besoin de communiquer avec **tous** les autres participants
   → coût linéaire en n pour chacune de ces opérations

Besoin de cryptographie asymétrique

Solution : la **cryptographie à clé publique**, ou **cryptographie asymétrique**.

Besoin de cryptographie asymétrique

Solution : la **cryptographie à clé publique**, ou **cryptographie asymétrique**.

- chaque protagoniste P_i engendre un couple de clés $(\mathtt{pk}_i, \mathtt{sk}_i)$
  - la **clé publique** $\mathtt{pk}_i$ est distribuée publiquement (donc, elle est aussi connue des adversaires)
  - la **clé privée** $\mathtt{sk}_i$ est gardée secrètement par i

Besoin de cryptographie asymétrique

Solution : la **cryptographie à clé publique**, ou **cryptographie asymétrique**.

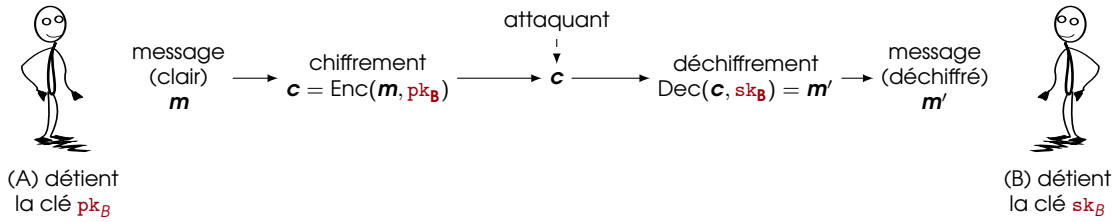
- chaque protagoniste P_i engendre un couple de clés $(\mathtt{pk}_i, \mathtt{sk}_i)$
  - la **clé publique** $\mathtt{pk}_i$ est distribuée publiquement (donc, elle est aussi connue des adversaires)
  - la **clé privée** $\mathtt{sk}_i$ est gardée secrètement par i
- dans le réseau : n clés publiques, n clés privées (au lieu de $\simeq n^2/2$ clés secrètes)

Besoin de cryptographie asymétrique

Solution : la **cryptographie à clé publique**, ou **cryptographie asymétrique**.

- chaque protagoniste P_i engendre un couple de clés $(\mathtt{pk}_i, \mathtt{sk}_i)$
  - la **clé publique** $\mathtt{pk}_i$ est distribuée publiquement (donc, elle est aussi connue des adversaires)
  - la **clé privée** $\mathtt{sk}_i$ est gardée secrètement par i
- dans le réseau : n clés publiques, n clés privées (au lieu de $\simeq n^2/2$ clés secrètes)

Exemple pour le chiffrement, avec Alice (A) et Bob (B) :



La clé $\mathtt{pk_B}$ est publique (tout le monde chiffrer un message pour Bob).
La clé $\mathtt{sk_B}$ est privée (seul Bob peut déchiffrer ce qu'on lui envoie).

Un peu d'histoire

La cryptographie à clé publique est historiquement liée à **Walter Diffie et Martin Hellman** (prix Turing 2015). Article fondateur :

New directions in cryptography. W. Diffie, M. Hellman. IEEE Trans. Inf. Theory. **1976**.

Un peu d'histoire

La cryptographie à clé publique est historiquement liée à **Walter Diffie et Martin Hellman** (prix Turing 2015). Article fondateur :

New directions in cryptography. W. Diffie, M. Hellman. IEEE Trans. Inf. Theory. **1976**.

Auparavant, quelques mentions :

- Ellis en 1970, dans un article intitulé : *The possibility of non-secret encryption*, initialement gardé secret.
- Cocks en 1973, dans une note intitulée : *A note on non-secret encryption*.

Un peu d'histoire

La cryptographie à clé publique est historiquement liée à **Walter Diffie et Martin Hellman** (prix Turing 2015). Article fondateur :

New directions in cryptography. W. Diffie, M. Hellman. IEEE Trans. Inf. Theory. **1976**.

Auparavant, quelques mentions :

- Ellis en 1970, dans un article intitulé : *The possibility of non-secret encryption*, initialement gardé secret.
- Cocks en 1973, dans une note intitulée : *A note on non-secret encryption*.

Peu après, premières **mises en pratique** de l'idée. Parmi les plus célèbres :

- le système de **chiffrement RSA** (par Rivest, Shamir et Adleman) en 1977,
- le système de **chiffrement ElGamal** en 1984,
- le schéma de signature de **Schnorr** en 1989.

Un peu d'histoire

La cryptographie à clé publique est historiquement liée à **Walter Diffie et Martin Hellman** (prix Turing 2015). Article fondateur :

New directions in cryptography. W. Diffie, M. Hellman. IEEE Trans. Inf. Theory. **1976**.

Auparavant, quelques mentions :

- Ellis en 1970, dans un article intitulé : *The possibility of non-secret encryption*, initialement gardé secret.
- Cocks en 1973, dans une note intitulée : *A note on non-secret encryption*.

Peu après, premières **mises en pratique** de l'idée. Parmi les plus célèbres :

- le système de **chiffrement RSA** (par Rivest, Shamir et Adleman) en 1977,
- le système de **chiffrement ElGamal** en 1984,
- le schéma de signature de **Schnorr** en 1989.

Domaine encore **en évolution** (recherche très active) :

- progrès sur les algorithmes d'analyse ?
- résistance à l'ordinateur quantique ?

Fonctions à sens unique

La cryptographie asymétrique repose sur les concepts de **fonctions à sens unique** (*one-way function*) et de **trappe** (*trapdoor*).

Fonctions à sens unique

La cryptographie asymétrique repose sur les concepts de **fonctions à sens unique** (*one-way function*) et de **trappe** (*trapdoor*).

Définition. Une **fonction à sens unique** est une fonction $f : \mathcal{X} \to \mathcal{Y}$ telle que :

Fonctions à sens unique

La cryptographie asymétrique repose sur les concepts de **fonctions à sens unique** (*one-way function*) et de **trappe** (*trapdoor*).

Définition. Une **fonction à sens unique** est une fonction $f : \mathcal{X} \to \mathcal{Y}$ telle que :

1. il est « facile » de calculer $f(x)$ pour tout $x \in \mathcal{X}$,

Fonctions à sens unique

La cryptographie asymétrique repose sur les concepts de **fonctions à sens unique** (*one-way function*) et de **trappe** (*trapdoor*).

Définition. Une **fonction à sens unique** est une fonction $f : \mathcal{X} \to \mathcal{Y}$ telle que :

1. il est « facile » de calculer $f(x)$ pour tout $x \in \mathcal{X}$,
2. pour presque tout $y \in \mathcal{Y}$ de la forme $y = f(x)$, il est « difficile » de calculer x à partir de y seulement.

Fonctions à sens unique

La cryptographie asymétrique repose sur les concepts de **fonctions à sens unique** (*one-way function*) et de **trappe** (*trapdoor*).

Définition. Une **fonction à sens unique** est une fonction $f : \mathcal{X} \to \mathcal{Y}$ telle que :

1. il est « facile » de calculer $f(x)$ pour tout $x \in \mathcal{X}$,
2. pour presque tout $y \in \mathcal{Y}$ de la forme $y = f(x)$, il est « difficile » de calculer x à partir de y seulement.

La notion de « facilité » se réfère à une **capacité de calcul**.

Fonctions à sens unique

La cryptographie asymétrique repose sur les concepts de **fonctions à sens unique** (*one-way function*) et de **trappe** (*trapdoor*).

Définition. Une **fonction à sens unique** est une fonction $f : \mathcal{X} \to \mathcal{Y}$ telle que :

1. il est « facile » de calculer $f(x)$ pour tout $x \in \mathcal{X}$,
2. pour presque tout $y \in \mathcal{Y}$ de la forme $y = f(x)$, il est « difficile » de calculer x à partir de y seulement.

La notion de « facilité » se réfère à une **capacité de calcul**.

– On suppose actuellement que des calculs nécessitant plus de 2^{80} opérations binaires sont très difficiles, et ceux plus de 2^{128} opérations binaires sont infaisables.

Penser : 2^{30} op. = 1 seconde sur un processeur.

Fonctions à sens unique

La cryptographie asymétrique repose sur les concepts de **fonctions à sens unique** (*one-way function*) et de **trappe** (*trapdoor*).

Définition. Une **fonction à sens unique** est une fonction $f : \mathcal{X} \to \mathcal{Y}$ telle que :

1. il est « facile » de calculer $f(x)$ pour tout $x \in \mathcal{X}$,
2. pour presque tout $y \in \mathcal{Y}$ de la forme $y = f(x)$, il est « difficile » de calculer x à partir de y seulement.

La notion de « facilité » se réfère à une **capacité de calcul**.

- On suppose actuellement que des calculs nécessitant plus de 2^{80} opérations binaires sont très difficiles, et ceux plus de 2^{128} opérations binaires sont infaisables.

  Penser : 2^{30} op. = 1 seconde sur un processeur.

- On parlera souvent de complexité **polynomiale** ou **exponentielle** par rapport à un ou plusieurs paramètres du système.

Exemples de fonctions à sens unique

Exemple 1. La fonction d'**exponentiation modulo** N, pour certaines valeurs de N. Pour certaines valeurs de $g \in \mathbb{Z}/N\mathbb{Z}$ (appelées générateurs), la fonction

$$\begin{array}{rccc} f: & \mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & g^x \mod N \end{array}$$

est à sens unique.

1. L'évaluation de f est **rapide** : calculer $f(x)$ demande $\simeq \log(N)$ opérations dans le groupe, grâce à l'**algorithme d'exponentiation binaire** (voir slide suivante).

Exemples de fonctions à sens unique

Exemple 1. La fonction d'**exponentiation modulo** N, pour certaines valeurs de N. Pour certaines valeurs de $g \in \mathbb{Z}/N\mathbb{Z}$ (appelées générateurs), la fonction

$$\begin{array}{rccc} f: & \mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & g^x \mod N \end{array}$$

est à sens unique.

1. L'évaluation de f est **rapide** : calculer $f(x)$ demande $\simeq \log(N)$ opérations dans le groupe, grâce à l'**algorithme d'exponentiation binaire** (voir slide suivante).
2. Si N est bien choisi, inverser f est **coûteux**. C'est le problème du **logarithme discret**.

Exemples de fonctions à sens unique

Exemple 1. La fonction d'**exponentiation modulo** N, pour certaines valeurs de N. Pour certaines valeurs de $g \in \mathbb{Z}/N\mathbb{Z}$ (appelées générateurs), la fonction

$$\begin{array}{rccc} f: & \mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & g^x \mod N \end{array}$$

est à sens unique.

1. L'évaluation de f est **rapide** : calculer $f(x)$ demande $\simeq \log(N)$ opérations dans le groupe, grâce à l'**algorithme d'exponentiation binaire** (voir slide suivante).
2. Si N est bien choisi, inverser f est **coûteux**. C'est le problème du **logarithme discret**.

Exemple 2. La fonction de **multiplication de deux grands nombres premiers**. On note $\mathcal{P}_M = \{(p, q) \mid p, q \text{ nombres premiers distincts} \leq M\}$.

Exemples de fonctions à sens unique

Exemple 1. La fonction d'**exponentiation modulo** N, pour certaines valeurs de N. Pour certaines valeurs de $g \in \mathbb{Z}/N\mathbb{Z}$ (appelées générateurs), la fonction

$$\begin{array}{rccc} f: & \mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & g^x \mod N \end{array}$$

est à sens unique.

1. L'évaluation de f est **rapide** : calculer $f(x)$ demande $\simeq \log(N)$ opérations dans le groupe, grâce à l'**algorithme d'exponentiation binaire** (voir slide suivante).
2. Si N est bien choisi, inverser f est **coûteux**. C'est le problème du **logarithme discret**.

Exemple 2. La fonction de **multiplication de deux grands nombres premiers**. On note $\mathcal{P}_M = \{(p, q) \mid p, q \text{ nombres premiers distincts} \leq M\}$. On définit

$$\begin{array}{rccc} f: & \mathcal{P}_M & \to & \mathbb{Z} \\ & (p, q) & \mapsto & N = pq \end{array}$$

Exemples de fonctions à sens unique

Exemple 1. La fonction d'**exponentiation modulo** N, pour certaines valeurs de N. Pour certaines valeurs de $g \in \mathbb{Z}/N\mathbb{Z}$ (appelées générateurs), la fonction

$$\begin{array}{rccc} f: & \mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & g^x \mod N \end{array}$$

est à sens unique.

1. L'évaluation de f est **rapide** : calculer $f(x)$ demande $\simeq \log(N)$ opérations dans le groupe, grâce à l'**algorithme d'exponentiation binaire** (voir slide suivante).
2. Si N est bien choisi, inverser f est **coûteux**. C'est le problème du **logarithme discret**.

Exemple 2. La fonction de **multiplication de deux grands nombres premiers**. On note $\mathcal{P}_M = \{(p, q) \mid p, q \text{ nombres premiers distincts} \leq M\}$. On définit

$$\begin{array}{rccc} f: & \mathcal{P}_M & \to & \mathbb{Z} \\ & (p, q) & \mapsto & N = pq \end{array}$$

1. Évaluation **rapide** : complexité polynomiale en $\log(M)$ (méthode naïve : $\log^2(M)$).

Exemples de fonctions à sens unique

Exemple 1. La fonction d'**exponentiation modulo** N, pour certaines valeurs de N. Pour certaines valeurs de $g \in \mathbb{Z}/N\mathbb{Z}$ (appelées générateurs), la fonction

$$\begin{array}{rccc} f: & \mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & g^x \mod N \end{array}$$

est à sens unique.

1. L'évaluation de f est **rapide** : calculer $f(x)$ demande $\simeq \log(N)$ opérations dans le groupe, grâce à l'**algorithme d'exponentiation binaire** (voir slide suivante).
2. Si N est bien choisi, inverser f est **coûteux**. C'est le problème du **logarithme discret**.

Exemple 2. La fonction de **multiplication de deux grands nombres premiers**. On note $\mathcal{P}_M = \{(p, q) \mid p, q \text{ nombres premiers distincts} \leq M\}$. On définit

$$\begin{array}{rccc} f: & \mathcal{P}_M & \to & \mathbb{Z} \\ & (p, q) & \mapsto & N = pq \end{array}$$

1. Évaluation **rapide** : complexité polynomiale en $\log(M)$ (méthode naïve : $\log^2(M)$).
2. Pour M assez grand, inverser f est **coûteux** en pratique. C'est le problème de la **factorisation d'entiers**.

Exponentiation rapide en 1 slide

Supposons que l'on sache évaluer rapidement :

- la multiplication de deux éléments $(g, h) \mapsto gh$
- le carré d'un élément $g \mapsto g^2$

Alors, pour évaluer l'opération $g \mapsto g^m$, il suffit de $\lceil \log(m) \rceil$ multiplications et $\lceil \log(m) \rceil$ élévations au carré.

Preuve : pour cela, on décompose m en binaire : $m = \sum_{i=0}^{d} m_i 2^i$, et on va calculer de manière itérative :

$$g^{m_0} \cdot (g^2)^{m_1} \cdot (g^{2^2})^{m_2} \cdots (g^{2^d})^{m_d} = g^m$$

Algorithme :

- `pow` $\leftarrow g$
- `res` $\leftarrow 1$
- **Pour** $i = 0, \ldots, d$, **faire :**
  - **Si** $m_i = 1$, **alors** `res` $\leftarrow$ `pow` $\times$ `res` *// ici on multiplie par* $g^{m_i 2^i}$
  - `pow` $\leftarrow$ `pow` $**$ `2`
- **Retourner** `res`.

Remarque : le calcul des m_i (c'est-à-dire, la décomposition en binaire) peut se faire en cours de boucle.

Fonctions à trappe

Définition. Une fonction à sens unique $f : \mathcal{X} \to \mathcal{Y}$ admet une **trappe** T si la connaissance de T permet de calculer facilement n'importe quel x à partir de son image $y = f(x) \in \mathcal{Y}$.

Fonctions à trappe

Définition. Une fonction à sens unique $f : \mathcal{X} \to \mathcal{Y}$ admet une **trappe** T si la connaissance de T permet de calculer facilement n'importe quel x à partir de son image $y = f(x) \in \mathcal{Y}$.

Exemple 3. Soit $N = pq$ un entier difficile à factoriser. Alors, la fonction de « carré modulaire » est à sens unique :

$$\begin{array}{rccc} f : & \mathbb{Z}/N\mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & x^2 \mod N \end{array}$$

Ici, une **trappe** consiste en la connaissance d'un exposant d tel que, pour tout x on a $x = x^{2d} \mod N$.

Fonctions à trappe

Définition. Une fonction à sens unique $f : \mathcal{X} \to \mathcal{Y}$ admet une **trappe** T si la connaissance de T permet de calculer facilement n'importe quel x à partir de son image $y = f(x) \in \mathcal{Y}$.

Exemple 3. Soit $N = pq$ un entier difficile à factoriser. Alors, la fonction de « carré modulaire » est à sens unique :

$$\begin{array}{rccc} f : & \mathbb{Z}/N\mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & x^2 \mod N \end{array}$$

Ici, une **trappe** consiste en la connaissance d'un exposant d tel que, pour tout x on a $x = x^{2d} \mod N$.

- On peut alors calculer $f(x)^d \mod N = (x^2 \mod N)^d \mod N = x \mod N$.

Définition. Une fonction à sens unique $f : \mathcal{X} \to \mathcal{Y}$ admet une **trappe** T si la connaissance de T permet de calculer facilement n'importe quel x à partir de son image $y = f(x) \in \mathcal{Y}$.

Exemple 3. Soit $N = pq$ un entier difficile à factoriser. Alors, la fonction de « carré modulaire » est à sens unique :

$$\begin{array}{rccc} f : & \mathbb{Z}/N\mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & x^2 \mod N \end{array}$$

Ici, une **trappe** consiste en la connaissance d'un exposant d tel que, pour tout x on a $x = x^{2d} \mod N$.

- On peut alors calculer $f(x)^d \mod N = (x^2 \mod N)^d \mod N = x \mod N$.
- **Cet exposant existe !** On a exactement :

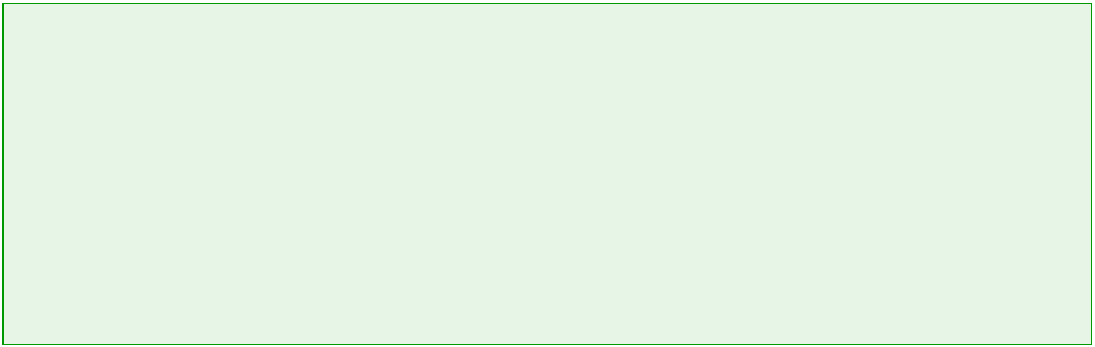
$$d = 2^{-1} \mod \phi(N)$$

où $\phi(N)$ est la fonction indicatrice d'Euler. Ainsi, $\phi(N)$ peut également être vue comme une trappe.

Plan

Définitions

Un schéma de **chiffrement asymétrique** (ou **à clé publique**) engage deux entités (Alice et Bob) et se décompose en trois étapes.



Définitions

Un schéma de **chiffrement asymétrique** (ou **à clé publique**) engage deux entités (Alice et Bob) et se décompose en trois étapes.

1. **Génération de clés.** Alice engendre une paire de clés $(\mathtt{pk}_A, \mathtt{sk}_A)$ où :
   - $\mathtt{pk}_A$ est la **clé publique** d'Alice (distribuée à tout le monde, dont Bob)
   - $\mathtt{sk}_A$ est la **clé privée** d'Alice (gardée secrètement)

   L'algorithme de génération de clés, KeyGen, prend en entrée un **paramètre de sécurité** désirée, ainsi que les paramètres du système.

Définitions

Un schéma de **chiffrement asymétrique** (ou **à clé publique**) engage deux entités (Alice et Bob) et se décompose en trois étapes.

1. **Génération de clés.** Alice engendre une paire de clés $(\mathtt{pk}_A, \mathtt{sk}_A)$ où :
   - $\mathtt{pk}_A$ est la **clé publique** d'Alice (distribuée à tout le monde, dont Bob)
   - $\mathtt{sk}_A$ est la **clé privée** d'Alice (gardée secrètement)

   L'algorithme de génération de clés, KeyGen, prend en entrée un **paramètre de sécurité** désirée, ainsi que les paramètres du système.
2. **Chiffrement.** Supposons que **Bob** souhaite envoyer un message $\boldsymbol{m}$ à Alice. Pour cela, Bob utilise la clé publique $\mathtt{pk}_A$ d'Alice :

$$\boldsymbol{c} = \mathrm{Enc}(\boldsymbol{m}, \mathtt{pk}_A)$$

Définitions

Un schéma de **chiffrement asymétrique** (ou **à clé publique**) engage deux entités (Alice et Bob) et se décompose en trois étapes.

1. **Génération de clés.** Alice engendre une paire de clés $(\mathtt{pk}_A, \mathtt{sk}_A)$ où :
   - $\mathtt{pk}_A$ est la **clé publique** d'Alice (distribuée à tout le monde, dont Bob)
   - $\mathtt{sk}_A$ est la **clé privée** d'Alice (gardée secrètement)

   L'algorithme de génération de clés, KeyGen, prend en entrée un **paramètre de sécurité** désirée, ainsi que les paramètres du système.
2. **Chiffrement.** Supposons que **Bob** souhaite envoyer un message $\boldsymbol{m}$ à Alice. Pour cela, Bob utilise la clé publique $\mathtt{pk}_A$ d'Alice :
   $$\boldsymbol{c} = \mathrm{Enc}(\boldsymbol{m}, \mathtt{pk}_A)$$
3. **Déchiffrement.** Alice souhaite déchiffrer un chiffré $\boldsymbol{c}$ qui lui est adressé. Pour cela, Alice utilise sa clé privée $\mathtt{sk}_A$:
   $$\boldsymbol{m}' = \mathrm{Dec}(\boldsymbol{c}, \mathtt{sk}_A)$$

Définitions

Un schéma de **chiffrement asymétrique** (ou **à clé publique**) engage deux entités (Alice et Bob) et se décompose en trois étapes.

1. **Génération de clés.** Alice engendre une paire de clés $(\mathtt{pk}_A, \mathtt{sk}_A)$ où :
   - $\mathtt{pk}_A$ est la **clé publique** d'Alice (distribuée à tout le monde, dont Bob)
   - $\mathtt{sk}_A$ est la **clé privée** d'Alice (gardée secrètement)

   L'algorithme de génération de clés, KeyGen, prend en entrée un **paramètre de sécurité** désirée, ainsi que les paramètres du système.
2. **Chiffrement.** Supposons que **Bob** souhaite envoyer un message $\boldsymbol{m}$ à Alice. Pour cela, Bob utilise la clé publique $\mathtt{pk}_A$ d'Alice :
   $$\boldsymbol{c} = \mathrm{Enc}(\boldsymbol{m}, \mathtt{pk}_A)$$
3. **Déchiffrement.** Alice souhaite déchiffrer un chiffré $\boldsymbol{c}$ qui lui est adressé. Pour cela, Alice utilise sa clé privée $\mathtt{sk}_A$:
   $$\boldsymbol{m}' = \mathrm{Dec}(\boldsymbol{c}, \mathtt{sk}_A)$$

Le schéma de chiffrement est dit **valide** si pour tout message $\boldsymbol{m}$, on a

$$\mathrm{Dec}(\mathrm{Enc}(\boldsymbol{m}, \mathtt{pk}_A), \mathtt{sk}_A) = \boldsymbol{m}.$$

Idée informelle

Génération de clefs : création d'une fonction f à sens unique et à trappe T

- la **clé publique** est la fonction f ;
- la **clé privée** est la trappe T qui permet d'inverser f.

Idée informelle

Génération de clefs : création d'une fonction f à sens unique et à trappe T

- la **clé publique** est la fonction f ;
- la **clé privée** est la trappe T qui permet d'inverser f.

Chiffrement : application de la fonction à sens unique f sur le message $\boldsymbol{m}$

$$\mathrm{Enc}(\boldsymbol{m}, \mathtt{pk}) = f(\boldsymbol{m}) =: \boldsymbol{c}$$

$\rightarrow$ tout le monde peut chiffrer car f est la clé publique

Idée informelle

Génération de clefs : création d'une fonction f à sens unique et à trappe T
- la **clé publique** est la fonction f ;
- la **clé privée** est la trappe T qui permet d'inverser f.

Chiffrement : application de la fonction à sens unique f sur le message $\boldsymbol{m}$

$$\mathrm{Enc}(\boldsymbol{m}, \mathtt{pk}) = f(\boldsymbol{m}) =: \boldsymbol{c}$$

$\rightarrow$ tout le monde peut chiffrer car f est la clé publique

Déchiffrement : application de l'**inverse de la fonction** grâce à la trappe

$$\mathrm{Dec}(\boldsymbol{c}, \mathtt{sk}) = f_{(T)}^{-1}(\boldsymbol{c}) \qquad (= \boldsymbol{m})$$

$\rightarrow$ seul le destinataire « officiel » peut déchiffrer car la clé privée est la trappe

Sécurité d'un chiffrement asymétrique : types d'attaques

On distingue **différents types d'attaques** (= résultat de l'attaque) sur un chiffrement asymétrique.

Sécurité d'un chiffrement asymétrique : types d'attaques

On distingue **différents types d'attaques** (= résultat de l'attaque) sur un chiffrement asymétrique.
Par ordre d'importance décroissant :

Sécurité d'un chiffrement asymétrique : types d'attaques

On distingue **différents types d'attaques** (= résultat de l'attaque) sur un chiffrement asymétrique.
Par ordre d'importance décroissant :

Sécurité d'un chiffrement asymétrique : types d'attaques

On distingue **différents types d'attaques** (= résultat de l'attaque) sur un chiffrement asymétrique.

Par ordre d'importance décroissant :

1. Le **cassage total**. Dans cette attaque, l'adversaire récupère un moyen (algorithme efficace, données) de déchiffrer **tous** les messages chiffrés à destination d'Alice.
   $\rightarrow$ une manière d'opérer un cassage total est de retrouver la clé privée d'Alice.

Sécurité d'un chiffrement asymétrique : types d'attaques

On distingue **différents types d'attaques** (= résultat de l'attaque) sur un chiffrement asymétrique.

Par ordre d'importance décroissant :

1. Le **cassage total**. Dans cette attaque, l'adversaire récupère un moyen (algorithme efficace, données) de déchiffrer **tous** les messages chiffrés à destination d'Alice.
   $\rightarrow$ une manière d'opérer un cassage total est de retrouver la clé privée d'Alice.
2. Le **cassage partiel**. Dans ce type d'attaque, l'adversaire retrouve
   - une information partielle sur la clé, ou les messages de Bob
   - ou bien, le message de Bob avec bonne probabilité

Sécurité d'un chiffrement asymétrique : types d'attaques

On distingue **différents types d'attaques** (= résultat de l'attaque) sur un chiffrement asymétrique.

Par ordre d'importance décroissant :

1. Le **cassage total**. Dans cette attaque, l'adversaire récupère un moyen (algorithme efficace, données) de déchiffrer **tous** les messages chiffrés à destination d'Alice.
   $\rightarrow$ une manière d'opérer un cassage total est de retrouver la clé privée d'Alice.
2. Le **cassage partiel**. Dans ce type d'attaque, l'adversaire retrouve
   - une information partielle sur la clé, ou les messages de Bob
   - ou bien, le message de Bob avec bonne probabilité
3. La **distinction de l'aléa**. Ce type d'attaque correspond à la capacité à distinguer un texte aléatoire (dans l'espace des chiffrés) d'un chiffré particulier.

Sécurité d'un chiffrement asymétrique : modes d'attaques

On distingue **différents modes d'attaques** (= moyens de l'attaquant) sur un chiffrement asymétrique, par ordre de pouvoir de l'attaquant

Sécurité d'un chiffrement asymétrique : modes d'attaques

On distingue **différents modes d'attaques** (= moyens de l'attaquant) sur un chiffrement asymétrique, par ordre de pouvoir de l'attaquant

- **Attaque à chiffré seul** (COA, *ciphertext only attack*) : l'attaquant détient des chiffrés précédemment calculés.
  → Autrement dit, il observe ce qui passe sur le réseau.
- **Attaque à clair connu** (KPA, *known plaintext attack*) : l'attaquant a accès des couples de clairs/chiffrés précédemment calculés.
  → Autrement dit, il a vu des messages être chiffrés/déchiffrés.
- **Attaque à clair choisi** (CPA, *chosen plaintext attack*) : l'attaquant a accès à des couples de clairs/chiffrés dont il a pu choisir les clairs.
  → Autrement dit, il a eu accès à la fonction de chiffrement pendant un certain temps.
- **Attaques à chiffré choisi** (CCA, *chosen ciphertext attack* : l'attaquant a accès à des couples de clairs/chiffrés dont il a pu choisir les clairs ou les chiffrés.
  → Autrement dit, il a eu accès à la fonction de déchiffrement pendant un certain temps.

Sécurité d'un chiffrement asymétrique : modes d'attaques

On distingue **différents modes d'attaques** (= moyens de l'attaquant) sur un chiffrement asymétrique, par ordre de pouvoir de l'attaquant

- **Attaque à chiffré seul** (COA, *ciphertext only attack*) : l'attaquant détient des chiffrés précédemment calculés.
  → Autrement dit, il observe ce qui passe sur le réseau.
- **Attaque à clair connu** (KPA, *known plaintext attack*) : l'attaquant a accès des couples de clairs/chiffrés précédemment calculés.
  → Autrement dit, il a vu des messages être chiffrés/déchiffrés.
- **Attaque à clair choisi** (CPA, *chosen plaintext attack*) : l'attaquant a accès à des couples de clairs/chiffrés dont il a pu choisir les clairs.
  → Autrement dit, il a eu accès à la fonction de chiffrement pendant un certain temps.
- **Attaques à chiffré choisi** (CCA, *chosen ciphertext attack* : l'attaquant a accès à des couples de clairs/chiffrés dont il a pu choisir les clairs ou les chiffrés.
  → Autrement dit, il a eu accès à la fonction de déchiffrement pendant un certain temps.

Remarque. Lorsque le type d'attaque est la distinction, on parle de sécurité IND-CPA pour l'attaque à clair choisi, et IND-CCA1/IND-CCA2 pour l'attaque à chiffré choisi.

Définir une notion de sécurité

Comment estimer la **sécurité** d'un système cryptographique ?

Définir une notion de sécurité

Comment estimer la **sécurité** d'un système cryptographique ?

Deux approches :

1. sécurité **inconditionnelle** : peu importe la capacité de calcul de l'attaquant, il ne peut pas obtenir d'information sur le secret,
2. sécurité **calculatoire** : attaquer le système nécessite au moins une certaine capacité de calcul.

Définir une notion de sécurité

Comment estimer la **sécurité** d'un système cryptographique ?

Deux approches :

1. sécurité **inconditionnelle** : peu importe la capacité de calcul de l'attaquant, il ne peut pas obtenir d'information sur le secret,
2. sécurité **calculatoire** : attaquer le système nécessite au moins une certaine capacité de calcul.

En pratique, une bonne sécurité calculatoire est suffisante.

Définir une notion de sécurité

Comment estimer la **sécurité** d'un système cryptographique ?

Deux approches :

1. sécurité **inconditionnelle** : peu importe la capacité de calcul de l'attaquant, il ne peut pas obtenir d'information sur le secret,
2. sécurité **calculatoire** : attaquer le système nécessite au moins une certaine capacité de calcul.

En pratique, une bonne sécurité calculatoire est suffisante. Actuellement,

- « très bonne sécurité » : $> 2^{128}$ opérations,
- « mauvaise sécurité » : $\leq 2^{80}$ opérations.

Définir une notion de sécurité

Comment estimer la **sécurité** d'un système cryptographique ?

Deux approches :

1. sécurité **inconditionnelle** : peu importe la capacité de calcul de l'attaquant, il ne peut pas obtenir d'information sur le secret,
2. sécurité **calculatoire** : attaquer le système nécessite au moins une certaine capacité de calcul.

En pratique, une bonne sécurité calculatoire est suffisante. Actuellement,

- « très bonne sécurité » : $> 2^{128}$ opérations,
- « mauvaise sécurité » : $\leq 2^{80}$ opérations.

Pour la sécurité calculatoire, on essaie de **réduire** la résolution d'un problème que l'on suppose difficile, à la capacité à attaquer le système.

Définir une notion de sécurité

Comment estimer la **sécurité** d'un système cryptographique ?

Deux approches :

1. sécurité **inconditionnelle** : peu importe la capacité de calcul de l'attaquant, il ne peut pas obtenir d'information sur le secret,
2. sécurité **calculatoire** : attaquer le système nécessite au moins une certaine capacité de calcul.

En pratique, une bonne sécurité calculatoire est suffisante. Actuellement,

- « très bonne sécurité » : $> 2^{128}$ opérations,
- « mauvaise sécurité » : $\leq 2^{80}$ opérations.

Pour la sécurité calculatoire, on essaie de **réduire** la résolution d'un problème que l'on suppose difficile, à la capacité à attaquer le système.

Informellement, cela signifie :

« *Si un attaquant sait attaquer le système cryptographique A, alors il peut résoudre le problème mathématique B* »

Remarques :

- pour démontrer qu'un système **n'est pas sûr**, il suffit donc de démontrer que l'on peut réduire l'attaque du système à la résolution d'un problème facile

Réduction de sécurité

Remarques :

- pour démontrer qu'un système **n'est pas sûr**, il suffit donc de démontrer que l'on peut réduire l'attaque du système à la résolution d'un problème facile
- donc, si on démontre :

  « *Si un attaquant sait résoudre le problème B,*
  alors il peut attaquer le système A »,

  alors :
  - il est **nécessaire** que le problème B soit difficile pour que le système A puisse éventuellement être sûr,

Réduction de sécurité

Remarques :

- pour démontrer qu'un système **n'est pas sûr**, il suffit donc de démontrer que l'on peut réduire l'attaque du système à la résolution d'un problème facile
- donc, si on démontre :

  « *Si un attaquant sait résoudre le problème B,*
  alors il peut attaquer le système A »,

  alors :
  - il est **nécessaire** que le problème B soit difficile pour que le système A puisse éventuellement être sûr,
  - même si le problème B est difficile, cela ne **prouve** pas la sécurité du système A.

Plan

Contexte et motivations

On souhaite imiter (voire améliorer) certaines propriétés des signatures manuscrites.

Exemples de ce qu'on souhaite signer **numériquement** :

- des emails
- du code (mise à jour de logiciels)
- des transactions bancaires (ecommerce)
- de la communication publique (sites web)
- des clés de chiffrement, des certificats

Contexte et motivations

On souhaite imiter (voire améliorer) certaines propriétés des signatures manuscrites.

Exemples de ce qu'on souhaite signer **numériquement** :

- des emails
- du code (mise à jour de logiciels)
- des transactions bancaires (ecommerce)
- de la communication publique (sites web)
- des clés de chiffrement, des certificats

Objectifs :

- **Intégrité** : on peut vérifier si le message a été modifié ou non.
- **Authenticité** : on peut associer un message à un émetteur.
- **Non-répudiation** : on ne peut pas nier avoir émis une signature valide.
- **Infalsifiabilité** : une autre personne ne peut pas prétendre avoir émis la signature.
- **Non-réutilisation :** on ne peut pas utiliser une même signature sur deux messages différents.

Contexte et motivations

On souhaite imiter (voire améliorer) certaines propriétés des signatures manuscrites.

Exemples de ce qu'on souhaite signer **numériquement** :

- des emails
- du code (mise à jour de logiciels)
- des transactions bancaires (ecommerce)
- de la communication publique (sites web)
- des clés de chiffrement, des certificats

Objectifs :

- **Intégrité** : on peut vérifier si le message a été modifié ou non.
- **Authenticité** : on peut associer un message à un émetteur.
- **Non-répudiation** : on ne peut pas nier avoir émis une signature valide.
- **Infalsifiabilité** : une autre personne ne peut pas prétendre avoir émis la signature.
- **Non-réutilisation :** on ne peut pas utiliser une même signature sur deux messages différents.

Remarque. Ces propriétés ne sont pas toutes vérifiées par la signature « physique ».

Remarque. Une signature d'un message m n'est pas :

1. du chiffrement (on ne cache pas la valeur m),
2. « l'inverse du chiffrement asymétrique » (parfois ça y ressemble),
3. un MAC (*message authentication code*) : les signatures sont publiquement vérifiables.

Remarque. Une signature d'un message m n'est pas :

1. du chiffrement (on ne cache pas la valeur m),
2. « l'inverse du chiffrement asymétrique » (parfois ça y ressemble),
3. un MAC (*message authentication code*) : les signatures sont publiquement vérifiables.

La signature va s'**apposer** au message, et devra dépendre explicitement de lui.

Remarque. Une signature d'un message m n'est pas :

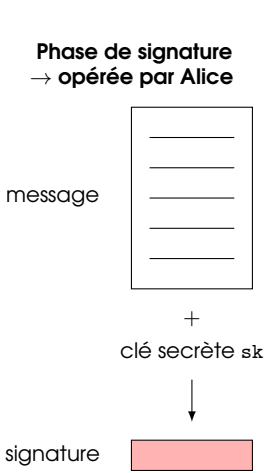
1. du chiffrement (on ne cache pas la valeur m),
2. « l'inverse du chiffrement asymétrique » (parfois ça y ressemble),
3. un MAC (*message authentication code*) : les signatures sont publiquement vérifiables.

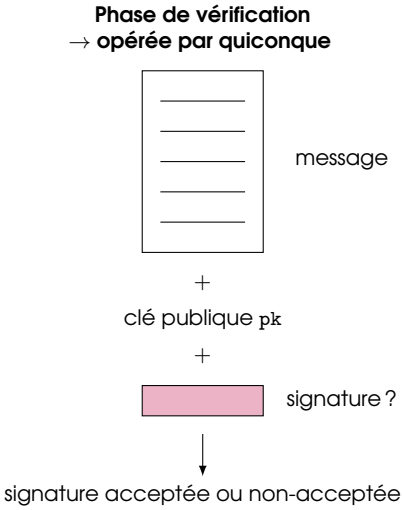
La signature va s'**apposer** au message, et devra dépendre explicitement de lui.

Par conséquent, les propriétés **additionnelles** désirables d'une signature sont :

- des signatures courtes,
- des algorithmes de signature et vérification rapides,
- des clés courtes.

Un dessin récapitulatif





Définition

Définition. Un schéma de **signature numérique** (à clé publique) est constitué de deux ensembles et trois algorithmes :

1. $\mathcal{M}$ un **ensemble de messages**.
2. $\mathcal{S}$ un **ensemble de signatures**.
3. **KeyGen** l'**algorithme de génération de clés**. Il renvoie un couple $(\texttt{pk}, \texttt{sk})$ de clé publique/clé privée.
4. **Sign** un **algorithme de signature**, qui prend en entrée un message $\boldsymbol{m} \in \mathcal{M}$, la clé privée $\texttt{sk}$, et retourne une signature $s = \textbf{Sign}(\boldsymbol{m}, \texttt{sk}) \in \mathcal{S}$.
5. **Verif** un **algorithme de vérification**, qui renvoie une valeur booléenne $\texttt{True}/\texttt{False}$. L'algorithme **Verif** prend en entrée la clé publique $\texttt{pk}$, le message $\boldsymbol{m}$ et la signature s.

Le schéma de signature est **valide** si

$$\forall (\boldsymbol{m}, s) \in \mathcal{M} \times \mathcal{S}, \quad \textbf{Verif}(\boldsymbol{m}, s, \texttt{pk}) = \texttt{True} \iff s = \textbf{Sign}(\boldsymbol{m}, \texttt{sk})$$

Moyens d'attaque sur une signature

Pour la **sécurité** du schéma, il faut définir un modèle d'attaquant (moyens) et un modèle d'attaque (objectifs).

Moyens d'attaque sur une signature

Pour la **sécurité** du schéma, il faut définir un modèle d'attaquant (moyens) et un modèle d'attaque (objectifs).

On distingue les moyens suivants :

Moyens d'attaque sur une signature

Pour la **sécurité** du schéma, il faut définir un modèle d'attaquant (moyens) et un modèle d'attaque (objectifs).

On distingue les moyens suivants :

1. **Attaque à clé seule** (*key-only attack*) : l'attaquant ne dispose que de la clé publique

Moyens d'attaque sur une signature

Pour la **sécurité** du schéma, il faut définir un modèle d'attaquant (moyens) et un modèle d'attaque (objectifs).

On distingue les moyens suivants :

1. **Attaque à clé seule** (*key-only attack*) : l'attaquant ne dispose que de la clé publique
2. **Attaque à message connu** (*known-message attack*) : l'attaquant dispose d'une liste de messages déjà signés $(\boldsymbol{m}_1, s_1), \ldots, (\boldsymbol{m}_\ell, s_\ell)$. Les signatures sont valides et réalisées avec la même clé.

Moyens d'attaque sur une signature

Pour la **sécurité** du schéma, il faut définir un modèle d'attaquant (moyens) et un modèle d'attaque (objectifs).

On distingue les moyens suivants :

1. **Attaque à clé seule** (*key-only attack*) : l'attaquant ne dispose que de la clé publique
2. **Attaque à message connu** (*known-message attack*) : l'attaquant dispose d'une liste de messages déjà signés $(\boldsymbol{m}_1, s_1), \ldots, (\boldsymbol{m}_\ell, s_\ell)$. Les signatures sont valides et réalisées avec la même clé.
3. **Attaque à message choisi** (*chosen-message attack*) : l'attaquant choisit des messages $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_\ell$ et demande les signatures $s_1, \ldots, s_\ell$ associées. Les signatures sont valides et réalisées avec la même clé.

Types d'attaques sur une signature

Les modèles d'attaque sont les suivants :

1. **Cassage total** : l'attaquant détermine une clé privée équivalente à celle d'Alice.

Types d'attaques sur une signature

Les modèles d'attaque sont les suivants :

1. **Cassage total** : l'attaquant détermine une clé privée équivalente à celle d'Alice.
2. **Falsification universelle** : avec probabilité non-négligeable, l'attaquant peut falsifier une signature d'un message choisi par quelqu'un d'autre. Ce message n'aura pas été signé précédemment.

Types d'attaques sur une signature

Les modèles d'attaque sont les suivants :

1. **Cassage total** : l'attaquant détermine une clé privée équivalente à celle d'Alice.
2. **Falsification universelle** : avec probabilité non-négligeable, l'attaquant peut falsifier une signature d'un message choisi par quelqu'un d'autre. Ce message n'aura pas été signé précédemment.
3. **Falsification existentielle** : avec probabilité non-négligeable, l'attaquant peut créer un couple ($\boldsymbol{m}$, s) où s est une signature valide de $\boldsymbol{m}$. Ce message n'aura pas été signé précédemment.

Sécurité d'une signature

On combine les définitions précédentes pour définir la sécurité d'un schéma de signature.

Par exemple :

Définition (exemple). On dit qu'un schéma satisfait la propriété d'**infalsification existentielle sous une attaque à message choisi** (EUF-CMA, *Existential UnForgeability under Chosen Message Attack*) si :

tout attaquant ayant accès à $\begin{cases} \text{la clé publique } \texttt{pk}, \\ \text{une liste de messages } \boldsymbol{m}_1, \ldots, \boldsymbol{m}_\ell \text{ qu'il a choisis}, \\ \text{et les signatures associées } s_1, \ldots, s_\ell, \end{cases}$

a une probabilité négligeable de retourner un message $\boldsymbol{m}' \notin \{m_i\}$ et une signature s' tels que $\textbf{Verif}(\boldsymbol{m}', s', \texttt{sk}) = \texttt{True}$.

$\Longrightarrow$ EUF-CMA est le standard de sécurité usuellement requis.

Plan

Plan

Un peu d'histoire

Historique.

- En 1976, Diffie et Hellman publient leur article fondateur de la cryptographie à clé publique.

  New directions in cryptography. W. Diffie, M. Hellman. IEEE Trans. Inf. Theory. **1976**.

Un peu d'histoire

Historique.

- En 1976, Diffie et Hellman publient leur article fondateur de la cryptographie à clé publique.

  New directions in cryptography. W. Diffie, M. Hellman. IEEE Trans. Inf. Theory. **1976**.

- L'année suivante naît le schéma de chiffrement RSA (Rivest, Shamir, Adleman). Publié dans :

  A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Rivest, Shamir, Adleman. Comm. ACM. **1978**. `[lien]`

Un peu d'histoire

Historique.

- En 1976, Diffie et Hellman publient leur article fondateur de la cryptographie à clé publique.

  New directions in cryptography. W. Diffie, M. Hellman. IEEE Trans. Inf. Theory. **1976**.

- L'année suivante naît le schéma de chiffrement RSA (Rivest, Shamir, Adleman). Publié dans :

  A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Rivest, Shamir, Adleman. Comm. ACM. **1978**. `[lien]`

- Breveté par le MIT en 1983. Brevet expiré en 2000.

Un peu d'histoire

Historique.

- En 1976, Diffie et Hellman publient leur article fondateur de la cryptographie à clé publique.

  New directions in cryptography. W. Diffie, M. Hellman. IEEE Trans. Inf. Theory. **1976**.

- L'année suivante naît le schéma de chiffrement RSA (Rivest, Shamir, Adleman). Publié dans :

  A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Rivest, Shamir, Adleman. Comm. ACM. **1978**. `[lien]`

- Breveté par le MIT en 1983. Brevet expiré en 2000.
- En 1973, le britannique Cocks décrit un algorithme similaire, « trop coûteux » pour l'époque.

Un peu d'histoire

Historique.

- En 1976, Diffie et Hellman publient leur article fondateur de la cryptographie à clé publique.

  New directions in cryptography. W. Diffie, M. Hellman. IEEE Trans. Inf. Theory. **1976**.

- L'année suivante naît le schéma de chiffrement RSA (Rivest, Shamir, Adleman). Publié dans :

  A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Rivest, Shamir, Adleman. Comm. ACM. **1978**. `[lien]`

- Breveté par le MIT en 1983. Brevet expiré en 2000.
- En 1973, le britannique Cocks décrit un algorithme similaire, « trop coûteux » pour l'époque.

- Actuellement utilisé dans beaucoup de protocoles et spécifications : SSL/TLS, IPSec, S/MIME (email)

Un peu d'histoire

Historique.

- En 1976, Diffie et Hellman publient leur article fondateur de la cryptographie à clé publique.
  New directions in cryptography. W. Diffie, M. Hellman. IEEE Trans. Inf. Theory. **1976**.

- L'année suivante naît le schéma de chiffrement RSA (Rivest, Shamir, Adleman). Publié dans :
  A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Rivest, Shamir, Adleman. Comm. ACM. **1978**. `[lien]`
- Breveté par le MIT en 1983. Brevet expiré en 2000.
- En 1973, le britannique Cocks décrit un algorithme similaire, « trop coûteux » pour l'époque.

- Actuellement utilisé dans beaucoup de protocoles et spécifications : SSL/TLS, IPSec, S/MIME (email)
- Certifié et standardisé (PKCS # 1, ANSI X9)

RSA : génération de clés

KeyGen : GÉNÉRATION DE LA PAIRE DE CLÉS RSA D'ALICE

RSA : génération de clés

KeyGen : GÉNÉRATION DE LA PAIRE DE CLÉS RSA D'ALICE

1. Choisir aléatoirement deux grands nombres premiers p et q.

RSA : génération de clés

KeyGen : GÉNÉRATION DE LA PAIRE DE CLÉS RSA D'ALICE

1. Choisir aléatoirement deux grands nombres premiers p et q.
2. Calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$.

RSA : génération de clés

KeyGen : GÉNÉRATION DE LA PAIRE DE CLÉS RSA D'ALICE

1. Choisir aléatoirement deux grands nombres premiers p et q.
2. Calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$.
3. Choisir un élément $e \in \{2, \ldots, \phi(n) - 1\}$ premier avec $\phi(n)$.

RSA : génération de clés

KeyGen : GÉNÉRATION DE LA PAIRE DE CLÉS RSA D'ALICE

1. Choisir aléatoirement deux grands nombres premiers p et q.
2. Calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$.
3. Choisir un élément $e \in \{2, \ldots, \phi(n)-1\}$ premier avec $\phi(n)$.
4. Calculer d tel que $de \equiv 1 \mod \phi(n)$.

RSA : génération de clés

KeyGen : GÉNÉRATION DE LA PAIRE DE CLÉS RSA D'ALICE

1. Choisir aléatoirement deux grands nombres premiers p et q.
2. Calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$.
3. Choisir un élément $e \in \{2, \ldots, \phi(n)-1\}$ premier avec $\phi(n)$.
4. Calculer d tel que $de \equiv 1 \mod \phi(n)$.
5. Retourner les clés suivantes :
   - clé publique $\mathtt{pk} = (n, e)$
   - clé privée $\mathtt{sk} = d$

RSA : génération de clés

KeyGen : GÉNÉRATION DE LA PAIRE DE CLÉS RSA D'ALICE

1. Choisir aléatoirement deux grands nombres premiers p et q.
2. Calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$.
3. Choisir un élément $e \in \{2, \ldots, \phi(n)-1\}$ premier avec $\phi(n)$.
4. Calculer d tel que $de \equiv 1 \mod \phi(n)$.
5. Retourner les clés suivantes :
   - clé publique $\mathtt{pk} = (n, e)$
   - clé privée $\mathtt{sk} = d$

Remarques.

- Dans ce cadre, pour Alice il n'est pas nécessaire de garder p et q.
- Il n'est pas nécessaire que e soit choisi aléatoirement : on prend $2^{16} + 1 = 65537$ dès qu'on le peut, pour des raisons d'efficacité.
- Il y a des contraintes supplémentaires sur la génération de p, q, e et d afin que le cryptosystème soit sûr.

RSA : chiffrement et déchiffrement

L'espace des textes clairs et celui des chiffrés sont $\mathbb{Z}/n\mathbb{Z}$ (les entiers modulo n).

RSA : chiffrement et déchiffrement

L'espace des textes clairs et celui des chiffrés sont $\mathbb{Z}/n\mathbb{Z}$ (les entiers modulo n).

Pour chiffrer $m \in \mathbb{Z}/n\mathbb{Z}$, Bob utilise la clé publique $\mathtt{pk} = (n, e)$ d'Alice :

CHIFFREMENT RSA : $\mathrm{Enc}(\cdot, (n, e))$

1. Calculer $c = m^e \mod n$
2. **Retourner** $\mathrm{Enc}(m, (n, e)) = c$

RSA : chiffrement et déchiffrement

L'espace des textes clairs et celui des chiffrés sont $\mathbb{Z}/n\mathbb{Z}$ (les entiers modulo n).

Pour chiffrer $m \in \mathbb{Z}/n\mathbb{Z}$, Bob utilise la clé publique $\mathtt{pk} = (n, e)$ d'Alice :

CHIFFREMENT RSA : $\mathrm{Enc}(\cdot, (n, e))$

1. Calculer $c = m^e \mod n$
2. **Retourner** $\mathrm{Enc}(m, (n, e)) = c$

Pour déchiffrer $c \in \mathbb{Z}/n\mathbb{Z}$, Alice utilise sa clé secrète $\mathtt{sk} = d$:

DÉCHIFFREMENT RSA : $\mathrm{Dec}(\cdot, d)$

1. Calculer $m' = c^d \mod n$
2. **Retourner** $\mathrm{Dec}(c, d) = m'$

RSA : chiffrement et déchiffrement

L'espace des textes clairs et celui des chiffrés sont $\mathbb{Z}/n\mathbb{Z}$ (les entiers modulo n).

Pour chiffrer $m \in \mathbb{Z}/n\mathbb{Z}$, Bob utilise la clé publique $\mathtt{pk} = (n, e)$ d'Alice :

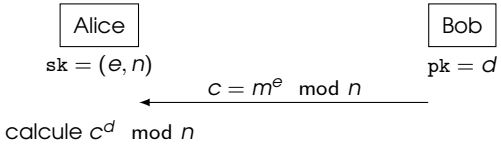
CHIFFREMENT RSA : $\mathrm{Enc}(\cdot, (n, e))$

1. Calculer $c = m^e \mod n$
2. **Retourner** $\mathrm{Enc}(m, (n, e)) = c$

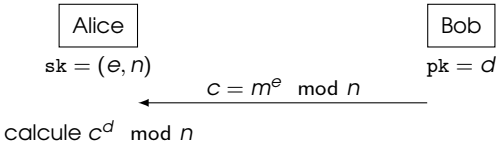
Pour déchiffrer $c \in \mathbb{Z}/n\mathbb{Z}$, Alice utilise sa clé secrète $\mathtt{sk} = d$:

DÉCHIFFREMENT RSA : $\mathrm{Dec}(\cdot, d)$

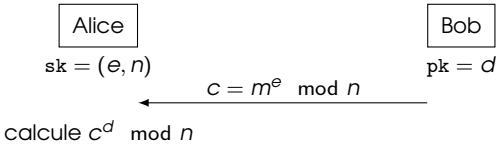
1. Calculer $m' = c^d \mod n$
2. **Retourner** $\mathrm{Dec}(c, d) = m'$



RSA est valide



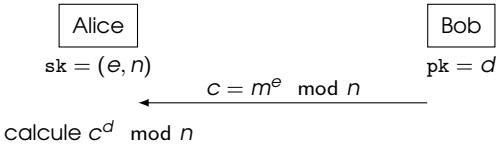
RSA est valide



Vérification de la validité de RSA. On a

$$m' \equiv c^d \equiv m^{ed} \mod n.$$

RSA est valide

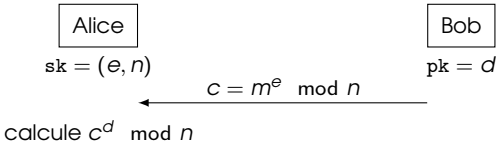


Vérification de la validité de RSA. On a

$$m' \equiv c^d \equiv m^{ed} \mod n.$$

Si $m = 0$, la validité est évidente.

RSA est valide

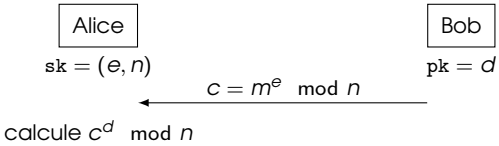


Vérification de la validité de RSA. On a

$$m' \equiv c^d \equiv m^{ed} \mod n.$$

Si $m = 0$, la validité est évidente. Sinon, on distingue deux cas :

RSA est valide



Vérification de la validité de RSA. On a

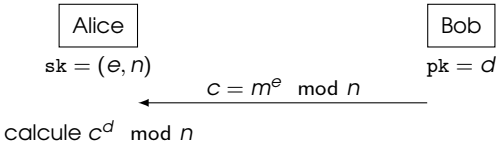
$$m' \equiv c^d \equiv m^{ed} \mod n.$$

Si $m = 0$, la validité est évidente. Sinon, on distingue deux cas :

- **Si** m est premier avec n, par le **théorème d'Euler**, on a $x^{\phi(n)} \equiv 1 \mod n$. Puis, comme $ed \equiv 1 \mod \phi(n)$, on obtient

$$m' \equiv m^{ed} \mod \phi(n) \equiv m \mod n.$$

RSA est valide



Vérification de la validité de RSA. On a

$$m' \equiv c^d \equiv m^{ed} \mod n.$$

Si $m = 0$, la validité est évidente. Sinon, on distingue deux cas :

- **Si** m est premier avec n, par le **théorème d'Euler**, on a $x^{\phi(n)} \equiv 1 \mod n$. Puis, comme $ed \equiv 1 \mod \phi(n)$, on obtient
$$m' \equiv m^{ed \mod \phi(n)} \equiv m \mod n.$$
- **Sinon :** ou bien p, ou bien q divise m. Prenons par exemple p. Alors :
  - m est premier avec q donc $m^{q-1} \equiv 1 \mod q$;
  - comme $\phi(n) = (p-1)(q-1)$, on a $m^{ed} \equiv m \mod q$ car $ed \equiv 1 \mod \phi(n)$;
  - pour finir, p et q divisent $m^{ed} - m$, ce qui implique que $m^{ed} \equiv m \mod n$.

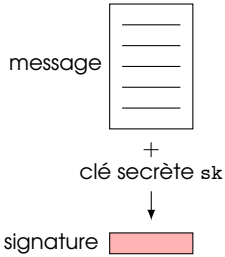
Exemple-jouet pour RSA

Alice choisit $n = 23 \times 41 = 943$ et $e = 3$.
Sa clé publique est $\mathtt{pk} = (e, n)$, sa clé privée est $\mathtt{sk} = d = e^{-1} \bmod \phi(n) = 587$.
Pour chiffrer $m = 111$, Bob calcule $111^3 \bmod 943$ qui vaut $c = 281$.
Pour déchiffrer $c = 281$, Alice calcule $281^{587} \bmod 943$, qui vaut bien $111 = m$.

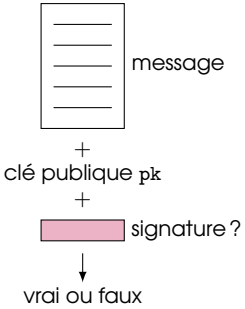
Plan

Première idée

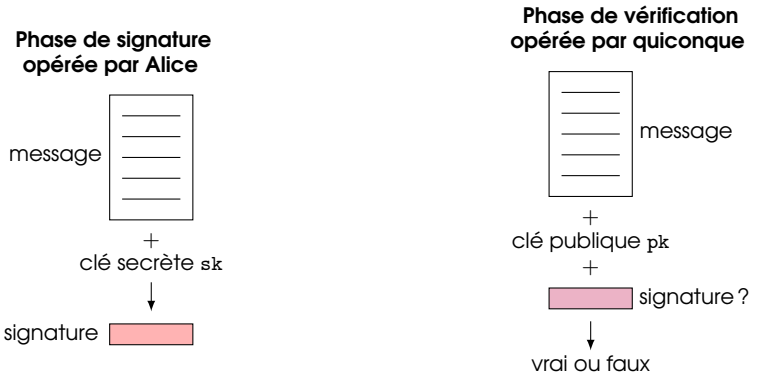
Phase de signature opérée par Alice



Phase de vérification opérée par quiconque



Première idée



Idée informelle : dans RSA, grâce à sa trappe d, Alice est la seule à savoir inverser la fonction à sens-unique $f : x \mapsto x^e \mod n$. Mais tout le monde sait calculer f !

Donc, pour un message m :

- la signature sera $s = f^{-1}(m)$
- on va vérifier publiquement que $f(s) = m$.

Signature RSA

Signature RSA : **KeyGen**

1. Calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$, où p et q sont deux grands nombres premiers aléatoires
2. Choisir e et d tels que $ed \equiv 1 \mod \phi(n)$
3. La clé publique est $\mathtt{pk} = (e, n)$, la clé privée est $\mathtt{sk} = (d, \phi(n))$.

Signature RSA

Signature RSA : **KeyGen**

1. Calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$, où p et q sont deux grands nombres premiers aléatoires
2. Choisir e et d tels que $ed \equiv 1 \mod \phi(n)$
3. La clé publique est $\mathtt{pk} = (e, n)$, la clé privée est $\mathtt{sk} = (d, \phi(n))$.

L'espace des messages est $\mathcal{M} = \mathbb{Z}/n\mathbb{Z}$ et celui des signatures est $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$.

Signature RSA

Signature RSA : **KeyGen**

1. Calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$, où p et q sont deux grands nombres premiers aléatoires
2. Choisir e et d tels que $ed \equiv 1 \mod \phi(n)$
3. La clé publique est $\mathtt{pk} = (e, n)$, la clé privée est $\mathtt{sk} = (d, \phi(n))$.

L'espace des messages est $\mathcal{M} = \mathbb{Z}/n\mathbb{Z}$ et celui des signatures est $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$.

Signature RSA : **Sign**$(m, \mathtt{sk})$

1. Calculer $s = m^d \mod n$.
2. Retourner s.

Signature RSA

Signature RSA : **KeyGen**

1. Calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$, où p et q sont deux grands nombres premiers aléatoires
2. Choisir e et d tels que $ed \equiv 1 \mod \phi(n)$
3. La clé publique est $\mathtt{pk} = (e, n)$, la clé privée est $\mathtt{sk} = (d, \phi(n))$.

L'espace des messages est $\mathcal{M} = \mathbb{Z}/n\mathbb{Z}$ et celui des signatures est $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$.

Signature RSA : **Sign**$(m, \mathtt{sk})$

1. Calculer $s = m^d \mod n$.
2. Retourner s.

Signature RSA : **Verif**$(m, s, \mathtt{pk})$

1. Calculer $m' = s^e \mod n$.
2. Faire le test $m' = m$ et retourner le booléen associé.

Signature RSA

Signature RSA : **KeyGen**

1. Calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$, où p et q sont deux grands nombres premiers aléatoires
2. Choisir e et d tels que $ed \equiv 1 \mod \phi(n)$
3. La clé publique est $\mathtt{pk} = (e, n)$, la clé privée est $\mathtt{sk} = (d, \phi(n))$.

L'espace des messages est $\mathcal{M} = \mathbb{Z}/n\mathbb{Z}$ et celui des signatures est $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$.

Signature RSA : **Sign**$(m, \mathtt{sk})$

1. Calculer $s = m^d \mod n$.
2. Retourner s.

Signature RSA : **Verif**$(m, s, \mathtt{pk})$

1. Calculer $m' = s^e \mod n$.
2. Faire le test $m' = m$ et retourner le booléen associé.

Validité. $m' \equiv s^e \equiv m^{ed} \equiv m \mod n$.

Résumé (signature RSA).

Clés : $\mathtt{pk} = (n, e)$, $\mathtt{sk} = d$,
Signature : $s = m^d \mod n$

Vérification : $(s^e \mod n) \stackrel{?}{\equiv} (m \mod n)$

Problème. Il existe une **attaque de falsification existentielle** sur la signature RSA « brute » avec comme moyens la clé publique seule.

Résumé (signature RSA).

Clés : $\mathtt{pk} = (n, e)$, $\mathtt{sk} = d$,
Signature : $s = m^d \mod n$

Vérification : $(s^e \mod n) \stackrel{?}{\equiv} (m \mod n)$

Problème. Il existe une **attaque de falsification existentielle** sur la signature RSA « brute » avec comme moyens la clé publique seule.

À partir de la clé publique $\mathtt{pk} = (n, e)$, on peut forger un message m' et une signature s' valide pour la clé privée $\mathtt{sk} = d$ d'Alice :

Résumé (signature RSA).

Clés : $\mathtt{pk} = (n, e)$, $\mathtt{sk} = d$,
Signature : $s = m^d \mod n$

Vérification : $(s^e \mod n) \overset{?}{\equiv} (m \mod n)$

Problème. Il existe une **attaque de falsification existentielle** sur la signature RSA « brute » avec comme moyens la clé publique seule.

À partir de la clé publique $\mathtt{pk} = (n, e)$, on peut forger un message m' et une signature s' valide pour la clé privée $\mathtt{sk} = d$ d'Alice :

1. Choisir s' aléatoirement dans $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$,
2. Calculer $m' = (s')^e \mod n$.

Résumé (signature RSA).

Clés : $\mathtt{pk} = (n, e)$, $\mathtt{sk} = d$,
Signature : $s = m^d \mod n$

Vérification : $(s^e \mod n) \overset{?}{\equiv} (m \mod n)$

Problème. Il existe une **attaque de falsification existentielle** sur la signature RSA « brute » avec comme moyens la clé publique seule.

À partir de la clé publique $\mathtt{pk} = (n, e)$, on peut forger un message m' et une signature s' valide pour la clé privée $\mathtt{sk} = d$ d'Alice :

1. Choisir s' aléatoirement dans $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$,
2. Calculer $m' = (s')^e \mod n$.

La signature RSA « brute » admet donc plusieurs **inconvénients :**

1. La sécurité : elle n'est pas EUF-KOA (*key-only attack*).
2. On ne peut pas signer un fichier de taille quelconque ($\mathcal{M} = \mathbb{Z}/n\mathbb{Z}$).

Résumé (signature RSA).

Clés : $\mathtt{pk} = (n, e)$, $\mathtt{sk} = d$,
Signature : $s = m^d \mod n$

Vérification : $(s^e \mod n) \overset{?}{\equiv} (m \mod n)$

Problème. Il existe une **attaque de falsification existentielle** sur la signature RSA « brute » avec comme moyens la clé publique seule.

À partir de la clé publique $\mathtt{pk} = (n, e)$, on peut forger un message m' et une signature s' valide pour la clé privée $\mathtt{sk} = d$ d'Alice :

1. Choisir s' aléatoirement dans $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$,
2. Calculer $m' = (s')^e \mod n$.

La signature RSA « brute » admet donc plusieurs **inconvénients :**

1. La sécurité : elle n'est pas EUF-KOA (*key-only attack*).
2. On ne peut pas signer un fichier de taille quelconque ($\mathcal{M} = \mathbb{Z}/n\mathbb{Z}$).

Solution : fonction de hachage !

Fonctions de hachage

Définition. Une famille de **fonctions de hachage cryptographiques** est un ensemble de fonctions $H_\kappa : \{0,1\}^* \to \mathcal{H}$, où $\mathcal{H}$ est un ensemble de taille fixe, et κ est un paramètre de la famille. L'élément $H_\kappa(\boldsymbol{m})$ est appelé **haché** de $\boldsymbol{m}$.

Définition. Une famille de **fonctions de hachage cryptographiques** est un ensemble de fonctions $H_\kappa : \{0,1\}^* \to \mathcal{H}$, où $\mathcal{H}$ est un ensemble de taille fixe, et κ est un paramètre de la famille. L'élément $H_\kappa(\boldsymbol{m})$ est appelé **haché** de $\boldsymbol{m}$.

Une fonction de hachage $H = H_\kappa$ est **résistante aux collisions** si pour tout algorithme polynomial probabiliste $\mathcal{A}$, la probabilité

$$\mathbb{P}\left[\boldsymbol{m} \neq \boldsymbol{m}' \text{ et } H_\kappa(\boldsymbol{m}) = H_\kappa(\boldsymbol{m}') \mid (\boldsymbol{m}, \boldsymbol{m}') \leftarrow \mathcal{A}\right]$$

est négligeable devant un paramètre de sécurité donné.

Fonctions de hachage

Définition. Une famille de **fonctions de hachage cryptographiques** est un ensemble de fonctions $H_\kappa : \{0,1\}^* \to \mathcal{H}$, où $\mathcal{H}$ est un ensemble de taille fixe, et κ est un paramètre de la famille. L'élément $H_\kappa(\boldsymbol{m})$ est appelé **haché** de $\boldsymbol{m}$.

Une fonction de hachage $H = H_\kappa$ est **résistante aux collisions** si pour tout algorithme polynomial probabiliste $\mathcal{A}$, la probabilité

$$\mathbb{P}\left[\boldsymbol{m} \neq \boldsymbol{m}' \text{ et } H_\kappa(\boldsymbol{m}) = H_\kappa(\boldsymbol{m}') \mid (\boldsymbol{m}, \boldsymbol{m}') \leftarrow \mathcal{A}\right]$$

est négligeable devant un paramètre de sécurité donné.

Exemple : les fonctions SHA-3 (*secure hash algorithm*), dont les sorties sont de taille 224, 256, 384 ou 512 bits (au choix).

Fonctions de hachage

Définition. Une famille de **fonctions de hachage cryptographiques** est un ensemble de fonctions $H_\kappa : \{0,1\}^* \to \mathcal{H}$, où $\mathcal{H}$ est un ensemble de taille fixe, et κ est un paramètre de la famille. L'élément $H_\kappa(\boldsymbol{m})$ est appelé **haché** de $\boldsymbol{m}$.

Une fonction de hachage $H = H_\kappa$ est **résistante aux collisions** si pour tout algorithme polynomial probabiliste $\mathcal{A}$, la probabilité

$$\mathbb{P}\left[\boldsymbol{m} \neq \boldsymbol{m}' \text{ et } H_\kappa(\boldsymbol{m}) = H_\kappa(\boldsymbol{m}') \mid (\boldsymbol{m}, \boldsymbol{m}') \leftarrow \mathcal{A}\right]$$

est négligeable devant un paramètre de sécurité donné.

Exemple : les fonctions SHA-3 (*secure hash algorithm*), dont les sorties sont de taille 224, 256, 384 ou 512 bits (au choix).

Important. Les « anciennes » fonctions MD5 et SHA-1 ne sont pas résistantes aux collisions, mais elles restent utilisées pour des applications non-cryptographiques (à éviter néanmoins).

Fonctions de hachage

Définition. Une famille de **fonctions de hachage cryptographiques** est un ensemble de fonctions $H_\kappa : \{0,1\}^* \to \mathcal{H}$, où $\mathcal{H}$ est un ensemble de taille fixe, et κ est un paramètre de la famille. L'élément $H_\kappa(\boldsymbol{m})$ est appelé **haché** de $\boldsymbol{m}$.

Une fonction de hachage $H = H_\kappa$ est **résistante aux collisions** si pour tout algorithme polynomial probabiliste $\mathcal{A}$, la probabilité

$$\mathbb{P}\left[\boldsymbol{m} \neq \boldsymbol{m}' \text{ et } H_\kappa(\boldsymbol{m}) = H_\kappa(\boldsymbol{m}') \mid (\boldsymbol{m}, \boldsymbol{m}') \leftarrow \mathcal{A}\right]$$

est négligeable devant un paramètre de sécurité donné.

Exemple : les fonctions SHA-3 (*secure hash algorithm*), dont les sorties sont de taille 224, 256, 384 ou 512 bits (au choix).

Important. Les « anciennes » fonctions MD5 et SHA-1 ne sont pas résistantes aux collisions, mais elles restent utilisées pour des applications non-cryptographiques (à éviter néanmoins).

Dans toute la suite, on prend $H = H_\kappa : \{0,1\}^* \to \mathcal{H}$ une fonction de hachage résistante aux collisions.

RSA-FDH : *full domain hash*

Idée : au lieu de signer directement le message $\boldsymbol{m}$, on signe le haché $H(\boldsymbol{m})$ avec l'algorithme de signature RSA « brut » vu précédemment.

RSA-FDH : *full domain hash*

Idée : au lieu de signer directement le message $\boldsymbol{m}$, on signe le haché $H(\boldsymbol{m})$ avec l'algorithme de signature RSA « brut » vu précédemment.

La **génération de clés** est identique : on a $\mathtt{pk} = (n, e)$ et $\mathtt{sk} = d$.

RSA-FDH : *full domain hash*

Idée : au lieu de signer directement le message $\boldsymbol{m}$, on signe le haché $H(\boldsymbol{m})$ avec l'algorithme de signature RSA « brut » vu précédemment.

La **génération de clés** est identique : on a $\mathtt{pk} = (n, e)$ et $\mathtt{sk} = d$.

L'espace des messages est maintenant $\mathcal{M} = \{0,1\}^*$ et celui des signatures reste $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$. On suppose que H est à valeurs dans $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$.

RSA-FDH : *full domain hash*

Idée : au lieu de signer directement le message $\boldsymbol{m}$, on signe le haché $H(\boldsymbol{m})$ avec l'algorithme de signature RSA « brut » vu précédemment.

La **génération de clés** est identique : on a $\mathtt{pk} = (n, e)$ et $\mathtt{sk} = d$.

L'espace des messages est maintenant $\mathcal{M} = \{0,1\}^*$ et celui des signatures reste $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$. On suppose que H est à valeurs dans $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$.

Signature RSA-FDH : **Sign**($\boldsymbol{m}$, $\mathtt{sk}$)

1. Hacher $\boldsymbol{m}$, c'est-à-dire calculer $h := H(\boldsymbol{m})$.
2. Calculer et retourner $s = h^d \mod n$.

RSA-FDH : *full domain hash*

Idée : au lieu de signer directement le message $\boldsymbol{m}$, on signe le haché $H(\boldsymbol{m})$ avec l'algorithme de signature RSA « brut » vu précédemment.

La **génération de clés** est identique : on a $\mathtt{pk} = (n, e)$ et $\mathtt{sk} = d$.

L'espace des messages est maintenant $\mathcal{M} = \{0,1\}^*$ et celui des signatures reste $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$. On suppose que H est à valeurs dans $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$.

Signature RSA-FDH : **Sign**($\boldsymbol{m}$, $\mathtt{sk}$)

1. Hacher $\boldsymbol{m}$, c'est-à-dire calculer $h := H(\boldsymbol{m})$.
2. Calculer et retourner $s = h^d \mod n$.

Signature RSA-FDH : **Verif**($\boldsymbol{m}$, s, $\mathtt{pk}$)

1. Calculer $h' = s^e \mod n$.
2. Hacher $\boldsymbol{m}$, c'est-à-dire calculer $h := H(\boldsymbol{m})$
3. Faire le test $h' = h$ et retourner le booléen associé.

RSA-FDH : *full domain hash*

Idée : au lieu de signer directement le message $\boldsymbol{m}$, on signe le haché $H(\boldsymbol{m})$ avec l'algorithme de signature RSA « brut » vu précédemment.

La **génération de clés** est identique : on a $\mathtt{pk} = (n, e)$ et $\mathtt{sk} = d$.

L'espace des messages est maintenant $\mathcal{M} = \{0,1\}^*$ et celui des signatures reste $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$. On suppose que H est à valeurs dans $\mathcal{S} = \mathbb{Z}/n\mathbb{Z}$.

Signature RSA-FDH : **Sign**($\boldsymbol{m}$, $\mathtt{sk}$)

1. Hacher $\boldsymbol{m}$, c'est-à-dire calculer $h := H(\boldsymbol{m})$.
2. Calculer et retourner $s = h^d \mod n$.

Signature RSA-FDH : **Verif**($\boldsymbol{m}$, s, $\mathtt{pk}$)

1. Calculer $h' = s^e \mod n$.
2. Hacher $\boldsymbol{m}$, c'est-à-dire calculer $h := H(\boldsymbol{m})$
3. Faire le test $h' = h$ et retourner le booléen associé.

Validité. $h' \equiv s^e \equiv h^{ed} \equiv h \mod n$.

En pratique, on peut utiliser la fonction de hachage **SHA-3**, de sortie 224 ou 256 bits par exemple.

En pratique, on peut utiliser la fonction de hachage **SHA-3**, de sortie 224 ou 256 bits par exemple.

Pour obtenir une sécurité maximale, il faut que l'espace de définition de la fonction RSA ($\mathbb{Z}/n\mathbb{Z}$) soit le même que l'espace des hachés : « *full domain hash* ».

En pratique, on peut utiliser la fonction de hachage **SHA-3**, de sortie 224 ou 256 bits par exemple.

Pour obtenir une sécurité maximale, il faut que l'espace de définition de la fonction RSA ($\mathbb{Z}/n\mathbb{Z}$) soit le même que l'espace des hachés : « *full domain hash* ».

Problème : pour RSA, il faut choisir n de 2048 bits minimum, mais SHA-3 a une sortie de 224-256 bits.

En pratique, on peut utiliser la fonction de hachage **SHA-3**, de sortie 224 ou 256 bits par exemple.

Pour obtenir une sécurité maximale, il faut que l'espace de définition de la fonction RSA ($\mathbb{Z}/n\mathbb{Z}$) soit le même que l'espace des hachés : « *full domain hash* ».

Problème : pour RSA, il faut choisir n de 2048 bits minimum, mais SHA-3 a une sortie de 224-256 bits.

Plusieurs solutions :

- faire du remplissage aléatoire (*padding*),
- concaténer des hachés successifs $H^{(i)}(\boldsymbol{m})$
- concaténer des hachés avec incrément $H^{(i)}(\boldsymbol{m} \| \text{ctr})$.

En pratique, on peut utiliser la fonction de hachage **SHA-3**, de sortie 224 ou 256 bits par exemple.

Pour obtenir une sécurité maximale, il faut que l'espace de définition de la fonction RSA ($\mathbb{Z}/n\mathbb{Z}$) soit le même que l'espace des hachés : « *full domain hash* ».

Problème : pour RSA, il faut choisir n de 2048 bits minimum, mais SHA-3 a une sortie de 224-256 bits.

Plusieurs solutions :

- faire du remplissage aléatoire (*padding*),
- concaténer des hachés successifs $H^{(i)}(\boldsymbol{m})$
- concaténer des hachés avec incrément $H^{(i)}(\boldsymbol{m} \| \text{ctr})$.

Exemple :

$$FDH(\boldsymbol{m}, \text{IV}) = H(\boldsymbol{m} \| n \| \text{IV} + 0) \| H(\boldsymbol{m} \| n \| \text{IV} + 1) \| H(\boldsymbol{m} \| n \| \text{IV} + 2) \| \cdots$$

où IV est un vecteur d'initialisation public apposé au message.

Paradoxe des anniversaires

L'attaque par **paradoxe des anniversaires** s'emploie notamment pour obtenir une **collision** sur une fonction de hachage. Le « paradoxe » est le suivant :

> Quelle est la taille typique d'une classe d'étudiant.e.s pour qu'avec bonne probabilité, il existe deux etudiant.e.s ayant leur anniversaire le même jour ?

Pour les fonctions de hachages :

- supposons que les hachés sont de taille $t = 256$ bits ; alors il y a hachés possibles
- dans une liste de $\sqrt{t} = 2^{128}$ messages hachés, la probabilité que deux soient identiques est $\geq 1/2$.

Paradoxe des anniversaires

L'attaque par **paradoxe des anniversaires** s'emploie notamment pour obtenir une **collision** sur une fonction de hachage. Le « paradoxe » est le suivant :

> Quelle est la taille typique d'une classe d'étudiant.e.s pour qu'avec bonne probabilité, il existe deux etudiant.e.s ayant leur anniversaire le même jour ?

Pour les fonctions de hachages :

- supposons que les hachés sont de taille $t = 256$ bits ; alors il y a $2^t = 2^{256}$ hachés possibles
- dans une liste de $\sqrt{t} = 2^{128}$ messages hachés, la probabilité que deux soient identiques est $\geq 1/2$.

Performance

La signature RSA-FDH est une brique de base du standard RSA PKCS#1 v2.1.

Performance

La signature RSA-FDH est une brique de base du standard RSA PKCS#1 v2.1.

Performances :

- *Calcul efficace :* un haché + $O(1)$ exponentiations modulaires (les clés sont de taille indépendante de celle du fichier).
- *Taille de clés :*

  clé publique $2 \log_2 n \simeq 4096$ bits minimum
  clé privée $\log_2 n \simeq 2048$ bits minimum
- *Taille de signature :* $\log_2 n = 2048$ bits minimum

Plan

Le problème de l'échange de clés

Objectif. Alice et Bob souhaitent partager un **secret commun**, par exemple un nombre, ou une séquence de bits.

Le problème de l'échange de clés

Objectif. Alice et Bob souhaitent partager un **secret commun**, par exemple un nombre, ou une séquence de bits.

- On suppose qu'**avant** d'initier le protocole, ils ne détiennent aucune information commune.

Le problème de l'échange de clés

Objectif. Alice et Bob souhaitent partager un **secret commun**, par exemple un nombre, ou une séquence de bits.

- On suppose qu'**avant** d'initier le protocole, ils ne détiennent aucune information commune.
- Un **attaquant observe** les échanges entre Alice et Bob, et souhaite obtenir des informations sur le secret.

Le problème de l'échange de clés

Objectif. Alice et Bob souhaitent partager un **secret commun**, par exemple un nombre, ou une séquence de bits.

- On suppose qu'**avant** d'initier le protocole, ils ne détiennent aucune information commune.
- Un **attaquant observe** les échanges entre Alice et Bob, et souhaite obtenir des informations sur le secret.
- Des paramètres publics sont accessibles à Alice et Bob, mais également à l'attaquant.

Le problème de l'échange de clés

Objectif. Alice et Bob souhaitent partager un **secret commun**, par exemple un nombre, ou une séquence de bits.

- On suppose qu'**avant** d'initier le protocole, ils ne détiennent aucune information commune.
- Un **attaquant observe** les échanges entre Alice et Bob, et souhaite obtenir des informations sur le secret.
- Des paramètres publics sont accessibles à Alice et Bob, mais également à l'attaquant.

Motivation. Échanger une clé pour initier du chiffrement symétrique.

Pourquoi ?

- en pratique, le chiffrement symétrique est **plus rapide** que le chiffrement asymétrique
- cela permet d'instaurer des clés de session, dites clés **éphémères**

Le protocole de Diffie–Hellman

On prend la fonction à sens unique d'**exponentiation modulaire**. C'est-à-dire qu'on choisit un nombre premier p et un « générateur » g tel que :

1. calculer $g^x \mod p$ est facile pour tout x ;
2. retrouver x à partir de $g^x \mod p$ est difficile pour presque tout x.

Le **protocole d'échange de clefs de Diffie–Hellman** est alors :

Alice	Bob

Le protocole de Diffie–Hellman

On prend la fonction à sens unique d'**exponentiation modulaire**. C'est-à-dire qu'on choisit un nombre premier p et un « générateur » g tel que :

1. calculer $g^x \mod p$ est facile pour tout x ;
2. retrouver x à partir de $g^x \mod p$ est difficile pour presque tout x.

Le **protocole d'échange de clefs de Diffie–Hellman** est alors :

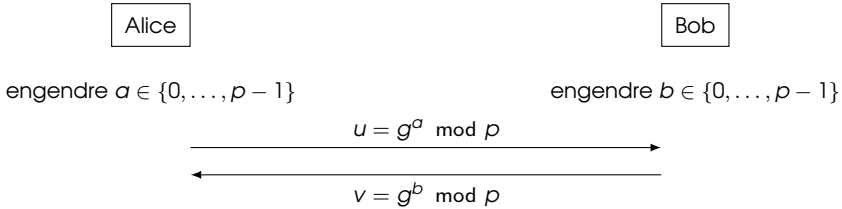
Alice	Bob
engendre $a \in \{0, \ldots, p-1\}$	engendre $b \in \{0, \ldots, p-1\}$

Le protocole de Diffie–Hellman

On prend la fonction à sens unique d'**exponentiation modulaire**. C'est-à-dire qu'on choisit un nombre premier p et un « générateur » g tel que :

1. calculer $g^x \mod p$ est facile pour tout x ;
2. retrouver x à partir de $g^x \mod p$ est difficile pour presque tout x.

Le **protocole d'échange de clefs de Diffie–Hellman** est alors :

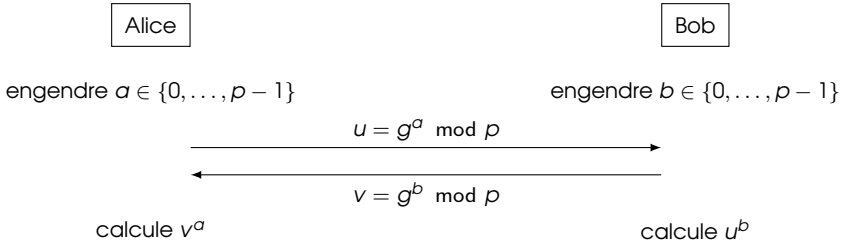


Le protocole de Diffie–Hellman

On prend la fonction à sens unique d'**exponentiation modulaire**. C'est-à-dire qu'on choisit un nombre premier p et un « générateur » g tel que :

1. calculer $g^x \mod p$ est facile pour tout x ;
2. retrouver x à partir de $g^x \mod p$ est difficile pour presque tout x.

Le **protocole d'échange de clefs de Diffie–Hellman** est alors :

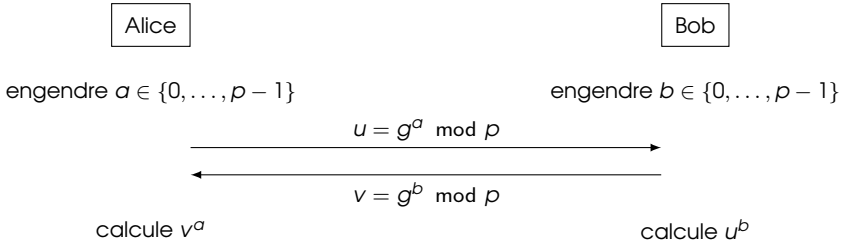


Le protocole de Diffie–Hellman

On prend la fonction à sens unique d'**exponentiation modulaire**. C'est-à-dire qu'on choisit un nombre premier p et un « générateur » g tel que :

1. calculer $g^x \mod p$ est facile pour tout x ;
2. retrouver x à partir de $g^x \mod p$ est difficile pour presque tout x.

Le **protocole d'échange de clefs de Diffie–Hellman** est alors :



On observe qu'Alice et Bob obtiennent une **valeur commune** :

$$k = v^a = u^b = g^{ab} \mod p\,.$$

Diffie–Hellman : exemple dans $(\mathbb{Z}/p\mathbb{Z})^\times$

Exemple avec $p = 83$.

Diffie–Hellman : exemple dans $(\mathbb{Z}/p\mathbb{Z})^\times$

Exemple avec $p = 83$.

Un générateur de $(\mathbb{Z}/p\mathbb{Z})^\times$ est $g = 2$.

Diffie–Hellman : exemple dans $(\mathbb{Z}/p\mathbb{Z})^\times$

Exemple avec $p = 83$.

Un générateur de $(\mathbb{Z}/p\mathbb{Z})^\times$ est $g = 2$.

Alice	Bob
engendre $a = 22$	engendre $b = 37$

Diffie–Hellman : exemple dans $(\mathbb{Z}/p\mathbb{Z})^\times$

Exemple avec $p = 83$.

Un générateur de $(\mathbb{Z}/p\mathbb{Z})^\times$ est $g = 2$.

Alice		Bob
engendre $a = 22$		engendre $b = 37$
	$u = 2^a = 65 \mod 83$ $\longrightarrow$	
	$\longleftarrow$ $v = 2^b = 57 \mod 83$	

Diffie–Hellman : exemple dans $(\mathbb{Z}/p\mathbb{Z})^\times$

Exemple avec $p = 83$.

Un générateur de $(\mathbb{Z}/p\mathbb{Z})^\times$ est $g = 2$.

Alice	Bob
engendre $a = 22$	engendre $b = 37$
$u = 2^a = 65 \mod 83$ →	
	← $v = 2^b = 57 \mod 83$
calcule $57^a = 48 \mod 83$	calcule $65^b = 48 \mod 83$

Diffie–Hellman : exemple dans $(\mathbb{Z}/p\mathbb{Z})^\times$

Exemple avec $p = 83$.

Un générateur de $(\mathbb{Z}/p\mathbb{Z})^\times$ est $g = 2$.

Alice	Bob
engendre $a = 22$	engendre $b = 37$
$u = 2^a = 65 \mod 83$	
$\longrightarrow$	
	$\longleftarrow$
	$v = 2^b = 57 \mod 83$
calcule $57^a = 48 \mod 83$	calcule $65^b = 48 \mod 83$

Le secret commun est 48.

Difficulté du calcul de logarithme discret

En pratique, le calcul du logarithme discret, c'est-à-dire $g^x \mod p \mapsto x$, est « assez » difficile :

Difficulté du calcul de logarithme discret

En pratique, le calcul du logarithme discret, c'est-à-dire $g^x \mod p \mapsto x$, est « assez » difficile :

- On ne connaît pas d'algorithme de résolution avec une complexité polynomiale en $\log p$.

Difficulté du calcul de logarithme discret

En pratique, le calcul du logarithme discret, c'est-à-dire $g^x \mod p \mapsto x$, est « assez » difficile :

- On ne connaît pas d'algorithme de résolution avec une complexité polynomiale en $\log p$.
- On pense généralement qu'il n'en existe pas.

Difficulté du calcul de logarithme discret

En pratique, le calcul du logarithme discret, c'est-à-dire $g^x \mod p \mapsto x$, est « assez » difficile :

- On ne connaît pas d'algorithme de résolution avec une complexité polynomiale en $\log p$.
- On pense généralement qu'il n'en existe pas.
- Le meilleur algorithme pour le résoudre est le **crible algébrique** (*number field sieve, NFS*), et a pour complexité :

$$2^e \quad \text{avec } e \simeq 1.92\ (\log p)^{1/3}\ (\log \log p)^{2/3}$$

Difficulté du calcul de logarithme discret

En pratique, le calcul du logarithme discret, c'est-à-dire $g^x \mod p \mapsto x$, est « assez » difficile :

- On ne connaît pas d'algorithme de résolution avec une complexité polynomiale en $\log p$.
- On pense généralement qu'il n'en existe pas.
- Le meilleur algorithme pour le résoudre est le **crible algébrique** (*number field sieve, NFS*), et a pour complexité :

$$2^e \quad \text{avec } e \simeq 1.92\ (\log p)^{1/3}\ (\log \log p)^{2/3}$$

Conséquences :

Difficulté du calcul de logarithme discret

En pratique, le calcul du logarithme discret, c'est-à-dire $g^x \mod p \mapsto x$, est « assez » difficile :

- On ne connaît pas d'algorithme de résolution avec une complexité polynomiale en $\log p$.
- On pense généralement qu'il n'en existe pas.
- Le meilleur algorithme pour le résoudre est le **crible algébrique** (*number field sieve, NFS*), et a pour complexité :

$$2^e \quad \text{avec } e \simeq 1.92\ (\log p)^{1/3}\ (\log\log p)^{2/3}$$

Conséquences :

Pour p de taille 2048 bits (c'est-à-dire $p \simeq 2^{2048}$), on obtient une sécurité correcte.

Difficulté du calcul de logarithme discret

En pratique, le calcul du logarithme discret, c'est-à-dire $g^x \mod p \mapsto x$, est « assez » difficile :

- On ne connaît pas d'algorithme de résolution avec une complexité polynomiale en $\log p$.
- On pense généralement qu'il n'en existe pas.
- Le meilleur algorithme pour le résoudre est le **crible algébrique** (*number field sieve, NFS*), et a pour complexité :

$$2^e \quad \text{avec } e \simeq 1.92\ (\log p)^{1/3}\ (\log\log p)^{2/3}$$

Conséquences :

Pour p de taille 2048 bits (c'est-à-dire $p \simeq 2^{2048}$), on obtient une sécurité correcte.

Il faut prendre p de taille 3072 bits pour une sécurité à long terme.

À quoi ressemble un premier de 2048 bits

Remarque :

Voici un nombre premier de 2048 bits :

$$p = 434857633695560884272208234064699791102789294786191770153853903557827027082130908706362119852587131908582796021351414878873195710356227447285701299986303947454400689904754064697663494002738078569276393891444301755782571687485181550912057385711172359602402830023282134250891119516190492386074898014927266738644017335834547384784651775784342884901739169865781616469518286339988579388655700220961757111910950132973379585844700414805378281952686404246201151361986284331220060965022805477217732443933185667826603635424734428259687861890467896776794851167170041960187730741024366771434388433346272922603504134260171776994111```

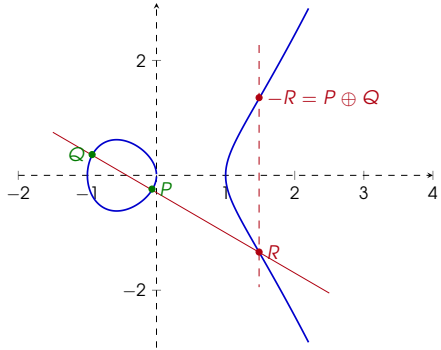
Remarque avancée : courbes ellipiques

On peut faire des calculs sur d'autres objets mathématiques que des **restes d'entiers**. Par exemple, des calculs géométriques sur les **points d'une courbe elliptique** procure une meilleure sécurité.

Dans de tels groupes, les meilleurs algorithmes résolvent le problème DL en temps

$$O(\sqrt{p})$$

où p est le plus grand nombre premier qui divise l'ordre du groupe.



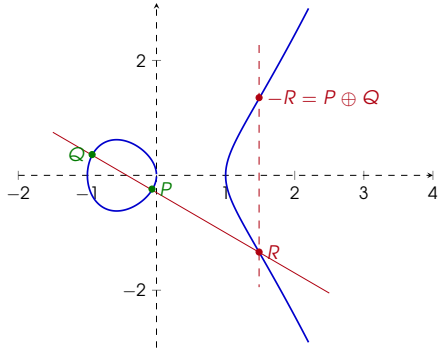
Remarque avancée : courbes ellipiques

On peut faire des calculs sur d'autres objets mathématiques que des **restes d'entiers**. Par exemple, des calculs géométriques sur les **points d'une courbe elliptique** procure une meilleure sécurité.

Dans de tels groupes, les meilleurs algorithmes résolvent le problème DL en temps

$$O(\sqrt{p})$$

où p est le plus grand nombre premier qui divise l'ordre du groupe.



Conséquence. Il est recommandé de choisir une courbe elliptique $\mathcal{E}$ et un entier q tel que l'ordre du groupe $\mathcal{E}(\mathbb{F}_q)$ **admette un facteur premier de taille au moins** 256 **bits**.

Attaque par le milieu (*man-in-the-middle*)

Question : en s'introduisant dans le protocole, Ève peut-elle anéantir la confidentialité du secret ?

Attaque par le milieu (*man-in-the-middle*)

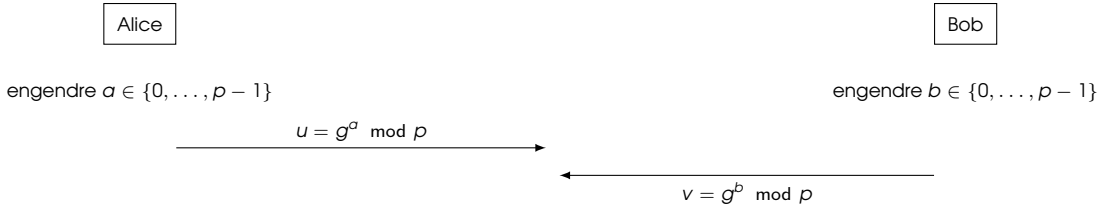
Question : en s'introduisant dans le protocole, Ève peut-elle anéantir la confidentialité du secret ?

On suppose ici qu'Ève est une adversaire **active**. Alors elle peut opérer une attaque « par le milieu » (*man-in-the-middle*).

Attaque par le milieu (*man-in-the-middle*)

Question : en s'introduisant dans le protocole, Ève peut-elle anéantir la confidentialité du secret ?

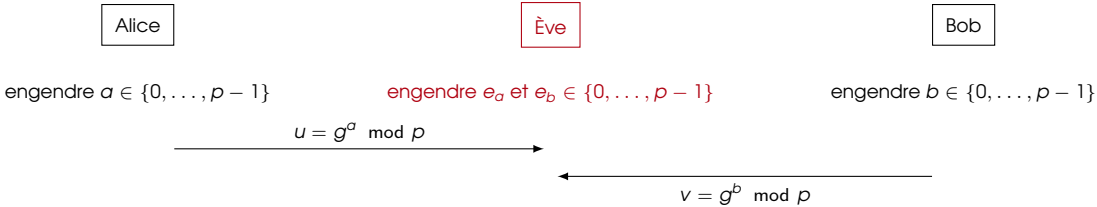
On suppose ici qu'Ève est une adversaire **active**. Alors elle peut opérer une attaque « par le milieu » (*man-in-the-middle*).



Attaque par le milieu (*man-in-the-middle*)

Question : en s'introduisant dans le protocole, Ève peut-elle anéantir la confidentialité du secret ?

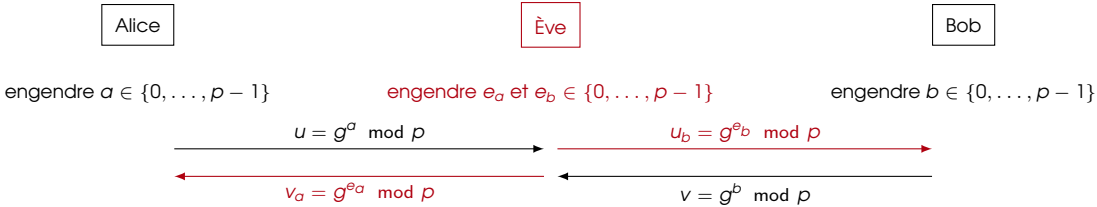
On suppose ici qu'Ève est une adversaire **active**. Alors elle peut opérer une attaque « par le milieu » (*man-in-the-middle*).



Attaque par le milieu (*man-in-the-middle*)

Question : en s'introduisant dans le protocole, Ève peut-elle anéantir la confidentialité du secret ?

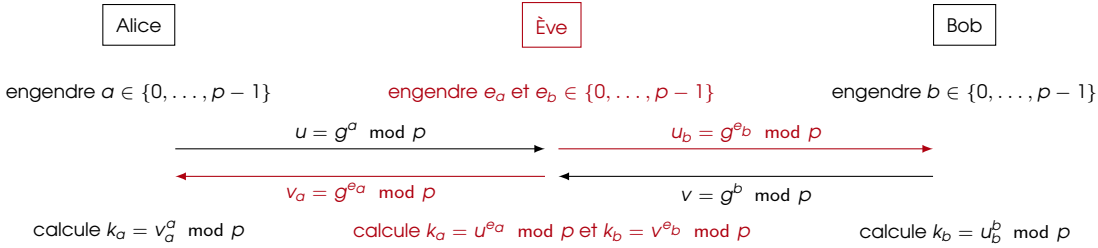
On suppose ici qu'Ève est une adversaire **active**. Alors elle peut opérer une attaque « par le milieu » (*man-in-the-middle*).



Attaque par le milieu (*man-in-the-middle*)

Question : en s'introduisant dans le protocole, Ève peut-elle anéantir la confidentialité du secret ?

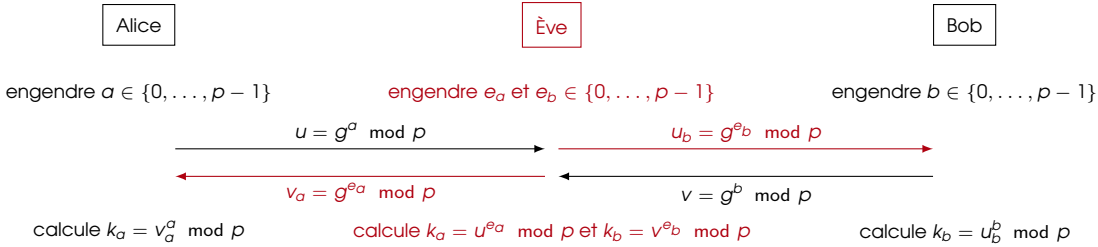
On suppose ici qu'Ève est une adversaire **active**. Alors elle peut opérer une attaque « par le milieu » (*man-in-the-middle*).



Attaque par le milieu (*man-in-the-middle*)

Question : en s'introduisant dans le protocole, Ève peut-elle anéantir la confidentialité du secret ?

On suppose ici qu'Ève est une adversaire **active**. Alors elle peut opérer une attaque « par le milieu » (*man-in-the-middle*).

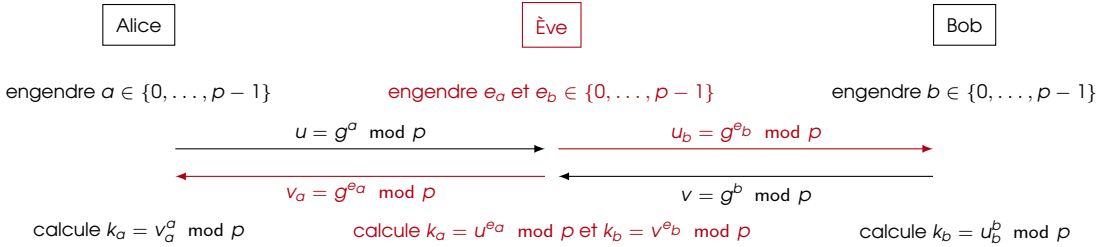


Alors, Ève possède deux clés k_a et k_b qu'elle peut utiliser respectivement avec Alice et Bob, **sans qu'ils ne s'en aperçoivent**.

Attaque par le milieu (*man-in-the-middle*)

Question : en s'introduisant dans le protocole, Ève peut-elle anéantir la confidentialité du secret ?

On suppose ici qu'Ève est une adversaire **active**. Alors elle peut opérer une attaque « par le milieu » (*man-in-the-middle*).



Alors, Ève possède deux clés k_a et k_b qu'elle peut utiliser respectivement avec Alice et Bob, **sans qu'ils ne s'en aperçoivent**.

Contre-mesure. L'idée est de **signer** les valeurs échangées avec des clés **certifiées** par une autorité de confiance.

Attaques par canaux auxiliaires

Au prochain cours...