

Information, Calcul et Communication

CS-119(k) ICC – Théorie Semaine 13

Rafael Pires
rafael.pires@epfl.ch

Évaluation approfondie

- Jusqu'au dimanche 8 juin
 - Connectez-vous à Moodle et restez sur la page d'accueil (tableau de bord, pas la page du cours)
 - Cliquer sur la flèche en haut à droite de l'écran qui fera apparaître un bloc contenant la tuile intitulée « Évaluation approfondie »
- Donnez un retour sur le cours
 - Feedback anonyme
 - Dites ce qui vous a plu, ce qui vous a déplu, et vos suggestions d'amélioration
 - Soyez constructifs : c'est votre opportunité d'aider à améliorer ce cours à l'avenir

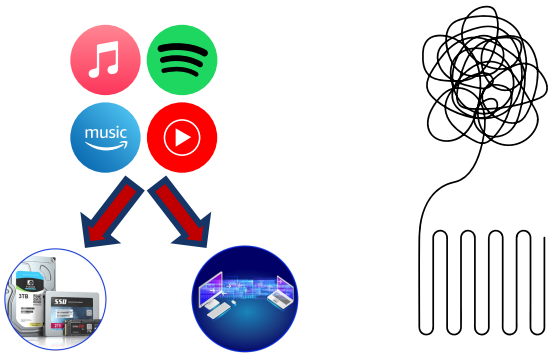
In-depth evaluation

Information,
calcul,
communication

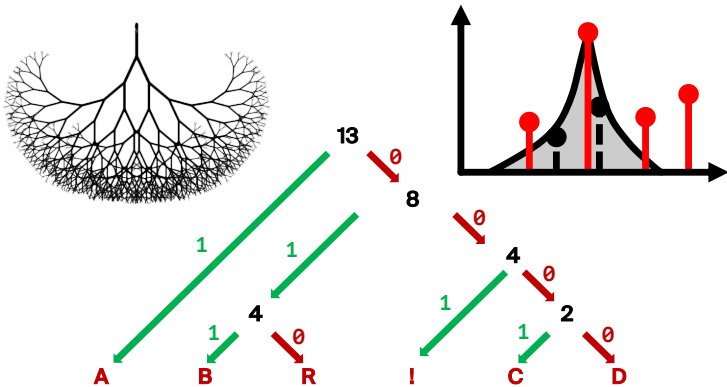
CS-119(k)_SP25

Current Response:
21/121

Précédemment, dans ICC-T 12

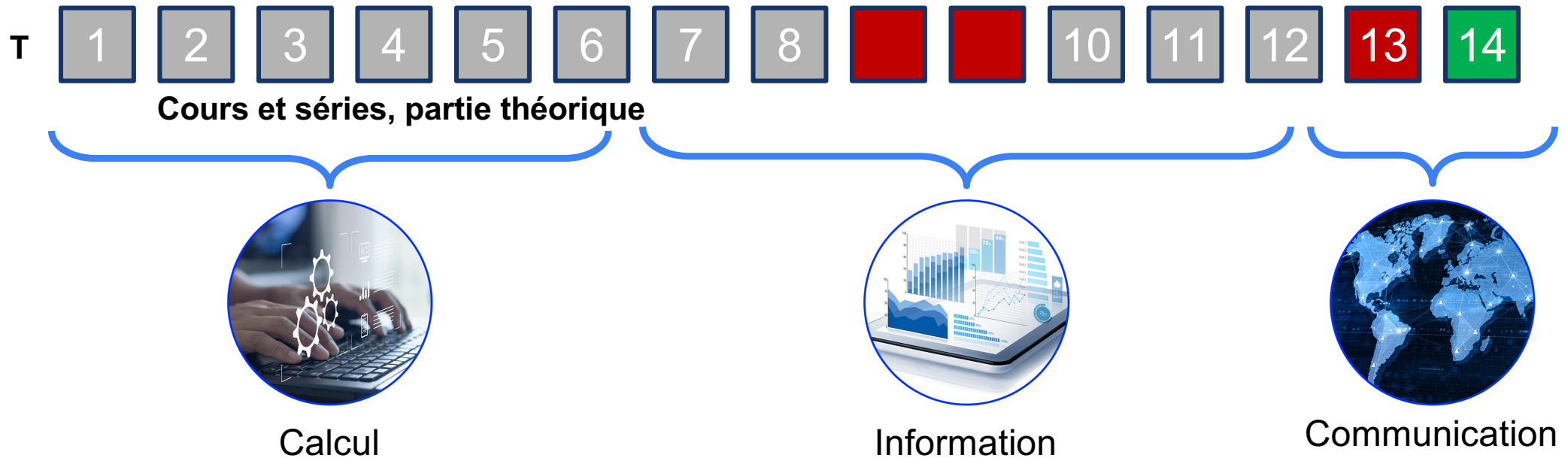


ABRACADABRA!!



- La **compression sans pertes** permet de réduire la taille d'un fichier sans perdre d'information.
- L'**entropie** mesure l'information moyenne par lettre dans une source : $H(X) = \sum p_j \log_2(1/p_j)$
- Les algorithmes de **Shannon-Fano** et **Huffman** construisent des **codes binaires sans préfixe** : codes plus courts aux lettres fréquentes.
- La **longueur moyenne du code** (par lettre) est donnée par $L(C) = \sum p_j \cdot l_j$
- Théorème de Shannon : $H(X) \leq L(C)$
 - L'entropie est une **borne théorique minimale** : on ne peut pas aller en-dessous sans pertes.
- La **compression avec pertes** permet de réduire davantage la taille (images, son, vidéo), au prix d'une **distorsion**.

Programme du cours



ICC



Information

- Représentation de l'information
 - Nombres entiers et réels
 - Circuits logiques et transistors
- Echantillonnage et reconstruction de signaux
- Entropie et compression de données



Calcul

- Algorithmes
- Complexité
- Récursivité
- Algorithmes gloutons / prog. dynamique
- Théorie de la calculabilité



Communication

- Cryptographie
- Réseaux

La sécurité informatique



■ Les besoins :

- Confidentialité
- Intégrité
- Authentification
- Non-répudiation

■ Les briques :

- Chiffrement
- Échange de clés
- Hachage
- Signatures

Les besoins



- **Confidentialité** : seul le destinataire peut lire
 - Réalisée via chiffrement
 - Empêche qu'un message intercepté soit lisible par un tiers non autorisé
- **Intégrité** : le message n'a pas été modifié
 - Détecte toute altération, même minime, du contenu
 - Utilisation typique des fonctions de hachage
 - Éviter les falsifications silencieuses

Les besoins

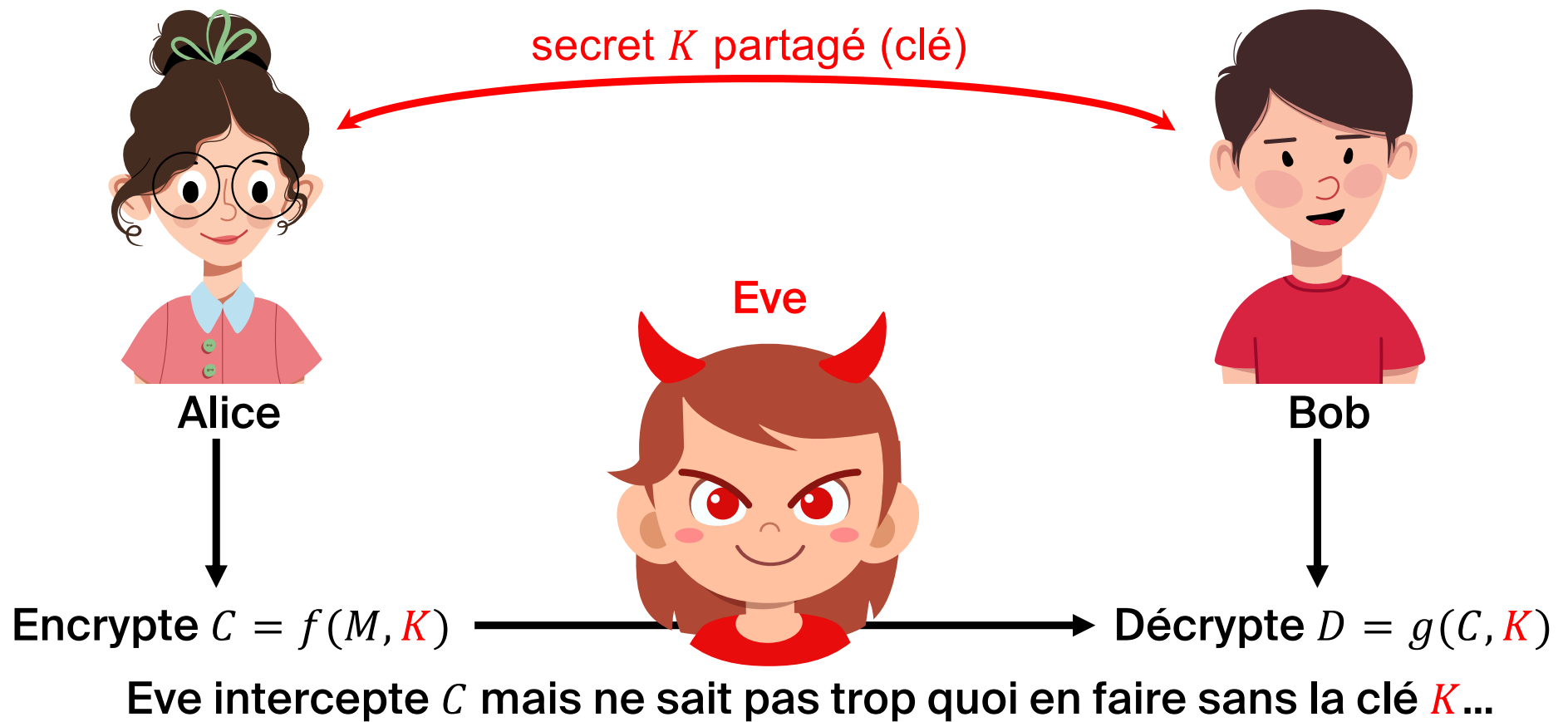


- **Authenticité** : s'assurer de l'identité de l'expéditeur
 - Utilise des signatures numériques ou des mécanismes d'authentification (mot de passe, biométrie)
 - Évite les attaques par usurpation d'identité
- **Non-répudiation** : L'émetteur ne peut pas nier avoir envoyé le message
 - Repose sur les signatures numériques et l'usage de clés privées
 - Important dans les contextes juridiques ou contractuels

Aujourd'hui

- **Chiffrement symétrique**
- **Protocole de Diffie-Hellman**
- **Fonctions de hachage cryptographiques**
- **Chiffrement asymétrique**

Chiffrement symétrique : scénario



Chiffrement de César



- Chiffrement par décalage de lettres dans l'alphabet
- Exemple avec un décalage de 3 :
 - Message : BONJOUR
 - Clé : +3
 - Chiffré : ERQMRXU
- **Facile à casser** : il n'y a que 25 possibilités
- Sert surtout à illustrer le principe de substitution

Clé à usage unique (« *one-time pad* »)

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Message M

C	3
O	15
U	21
C	3
O	15
U	21

+

Clé K

S	19
E	5
C	3
R	18
E	5
T	20

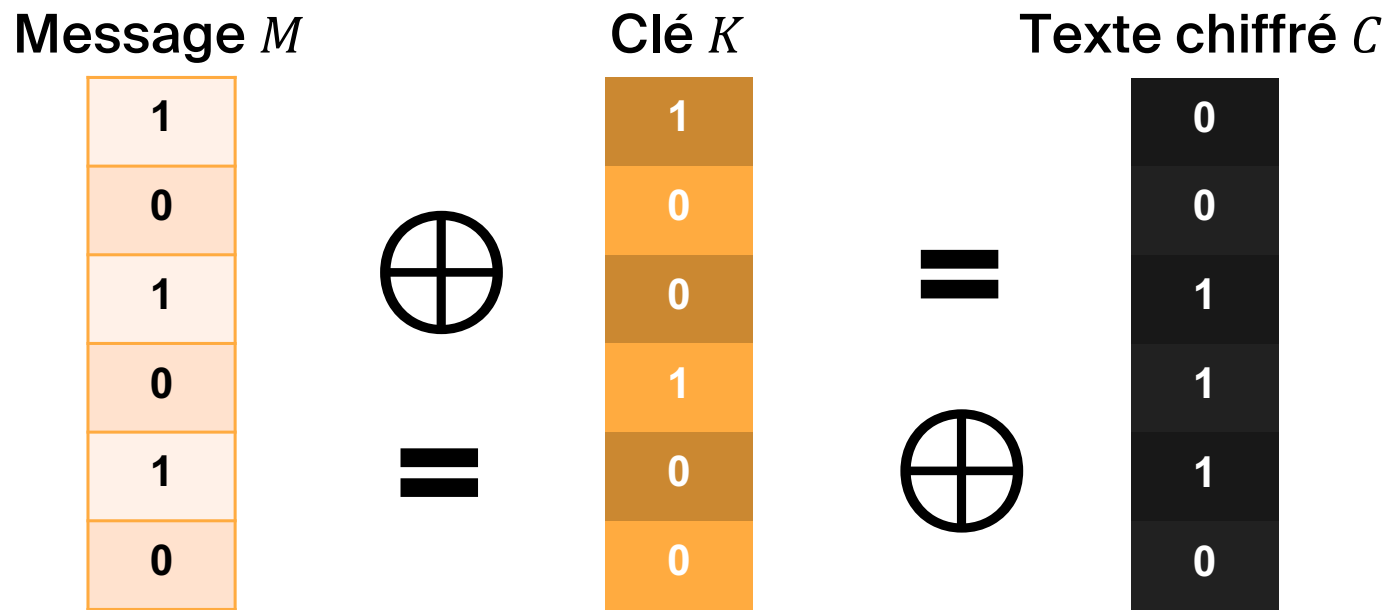
=

Texte chiffré C

V	22
T	20
X	24
U	21
T	20
O	15

$$C = (M + K) \bmod n \longrightarrow D = (C - K) \bmod n$$

Clé à usage unique (« *one-time pad* »)



$$C = M \oplus K \longrightarrow D = C \oplus K$$
$$D = M \oplus K \oplus K$$
$$D = M$$

Pas de débordement 👍

Clé à usage unique (« *one-time pad* »)

	Addition modulaire	XOR
Domaine	Entiers (e.g., 8, 16, 32 ou 64 bits)	Données binaires (bit par bit)
Chiffrement	$C = (M + K) \bmod n$	$C = M \oplus K$
Déchiffrement	$D = (C - K) \bmod n$	$D = C \oplus K$
Remarque	Nécessite une soustraction pour déchiffrer	Plus rapide, auto-inverse

Clé à usage unique (« *one-time pad* »)



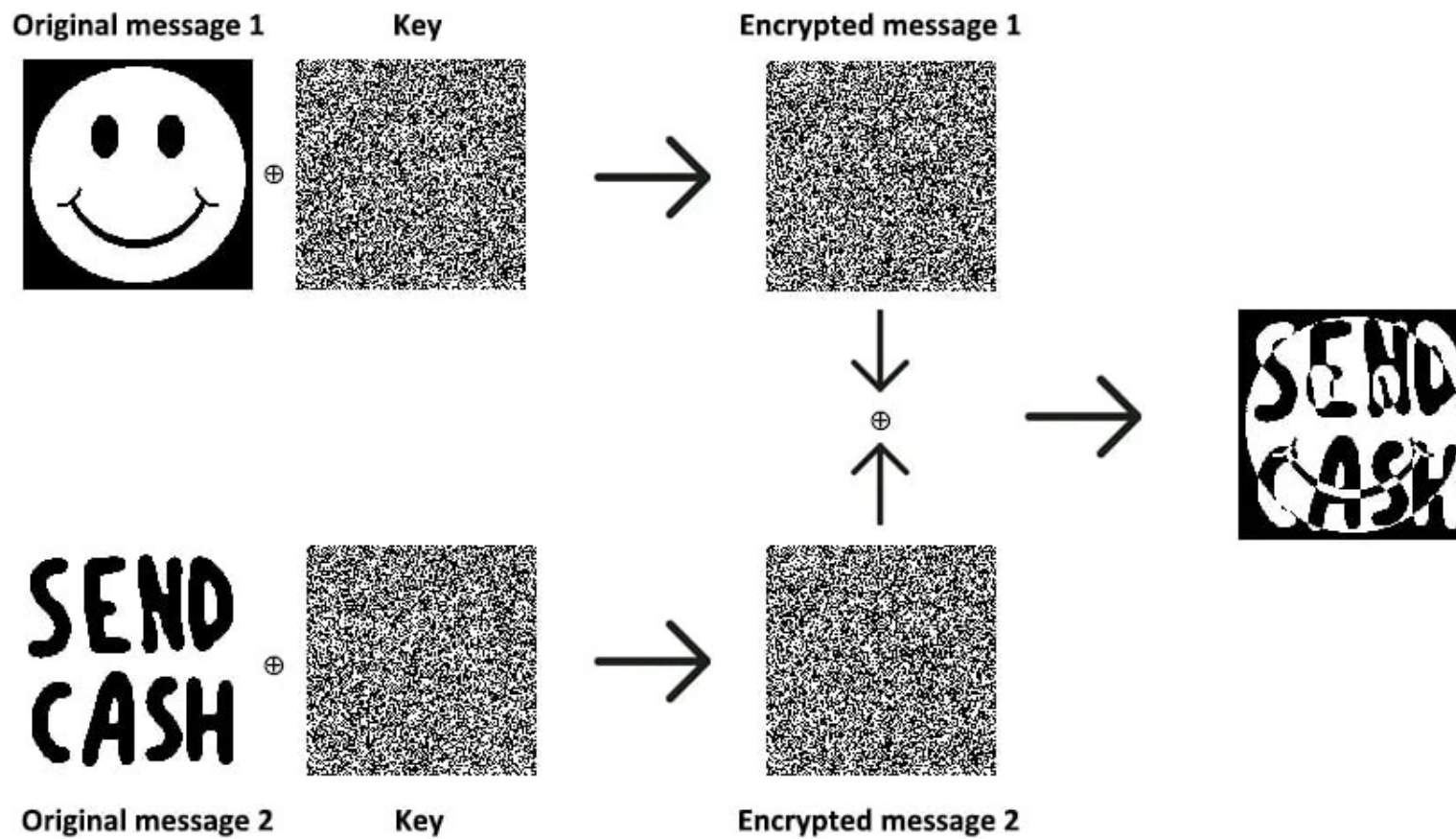
- Le OTP assure une sécurité parfaite **à condition que la clé :**
 - soit parfaitement **aléatoire**
 - ait exactement la **même taille** que le message
 - ne soit **jamais réutilisée**
- Le One-Time Pad ne repose pas sur un problème mathématique difficile, mais sur une **clé parfaitement aléatoire et unique.**
 - Cela le rend **résistant aux attaques quantiques.**

Défauts du OTP



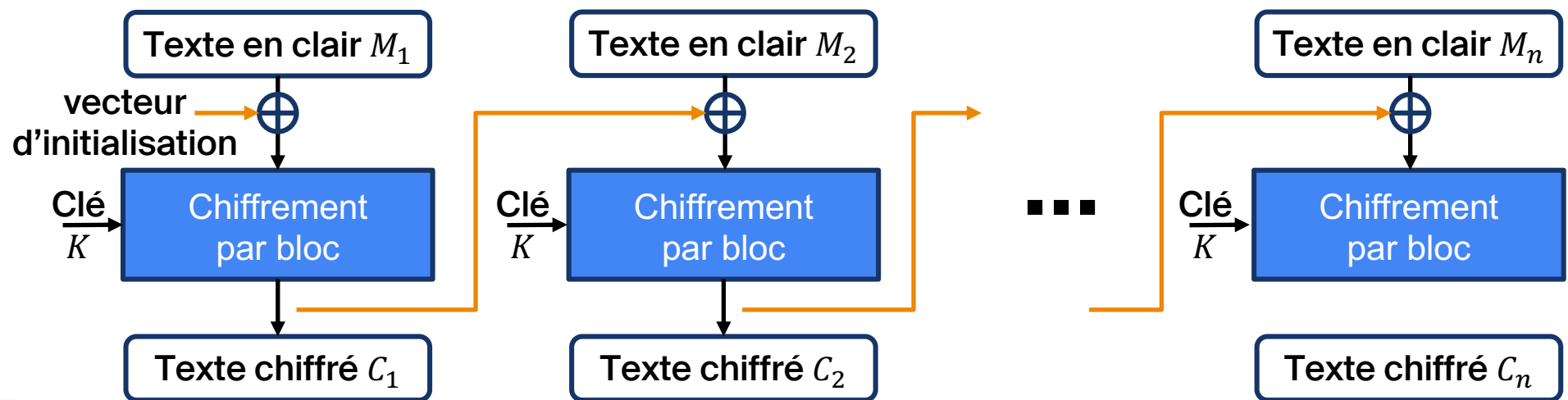
- Pour envoyer un message de longueur n , il faut une **clé aussi longue que le message**
- Cette clé doit être **transmise de façon secrète** avant toute communication
- Si la clé est trop simple (ex. $K = 00000000$), la sécurité est compromise
- Si la même clé est utilisée deux fois :
$$C1 = M1 \oplus K$$
$$C2 = M2 \oplus K$$
$$C1 \oplus C2 = M1 \oplus K \oplus M2 \oplus K = M1 \oplus M2$$
- Eve peut analyser les relations entre $M1$ et $M2$: **plus aucune sécurité !**

Défauts du OTP



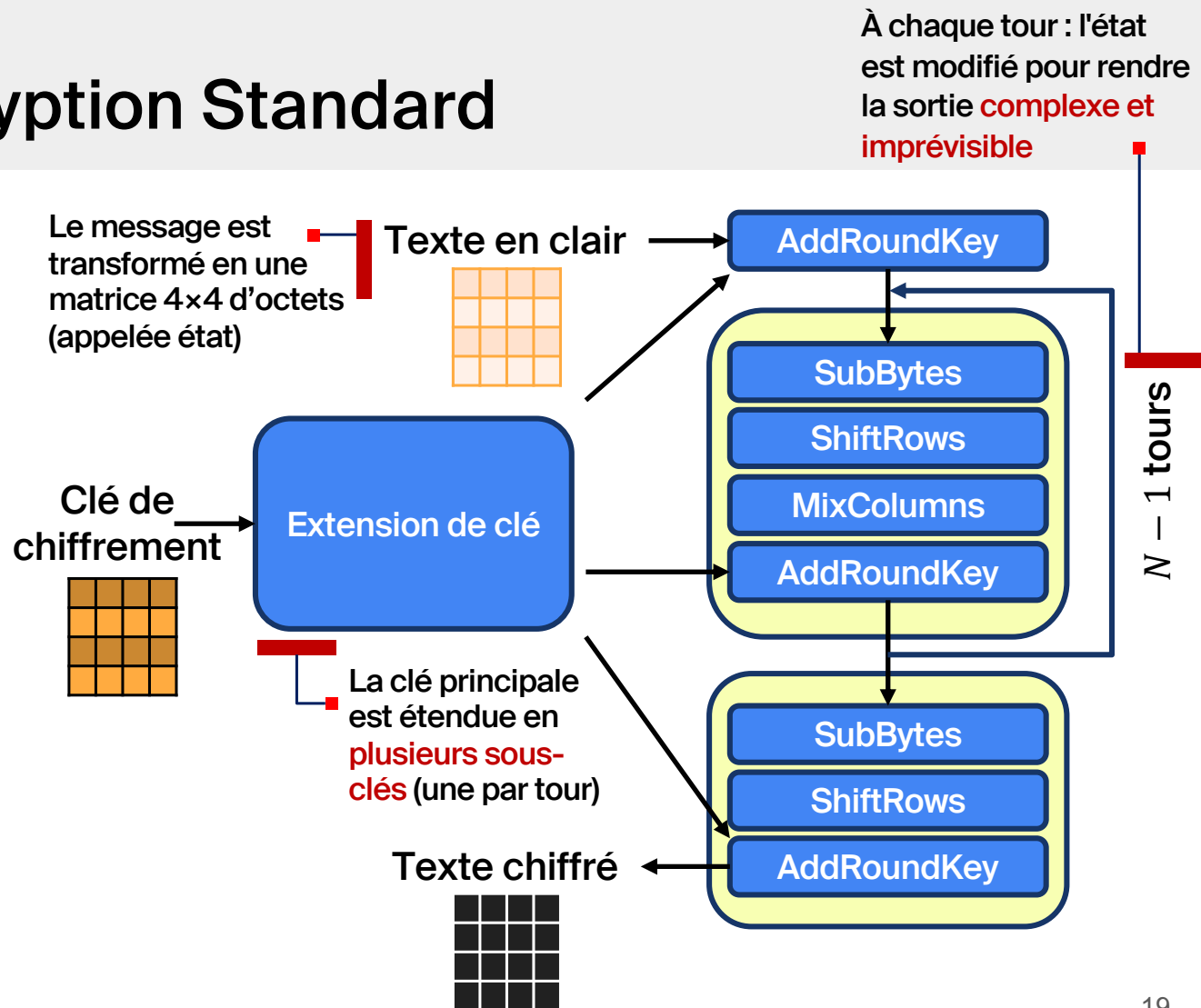
Chiffrements par blocs

- Contrairement au One-Time Pad, on utilise souvent **la même clé plusieurs fois**
- Un block cipher est un algorithme de chiffrement symétrique qui :
 - Prend un bloc de données M_i de taille fixe (par exemple 128 bits)
 - Utilise une clé secrète K
 - Et renvoie un bloc chiffré C_i de la même taille



AES : Advanced Encryption Standard

- Bloc de 128 bits
- Clés de 128, 192 ou 256 bits
- Fonctionne en 10, 12 ou 14 tours
- Chaque tour applique : substitution, permutation, mélange, et ajout de sous-clé
- Utilisé partout : Wi-fi, HTTPS, VPN, disques chiffrés



Bilan : Chiffrement symétrique

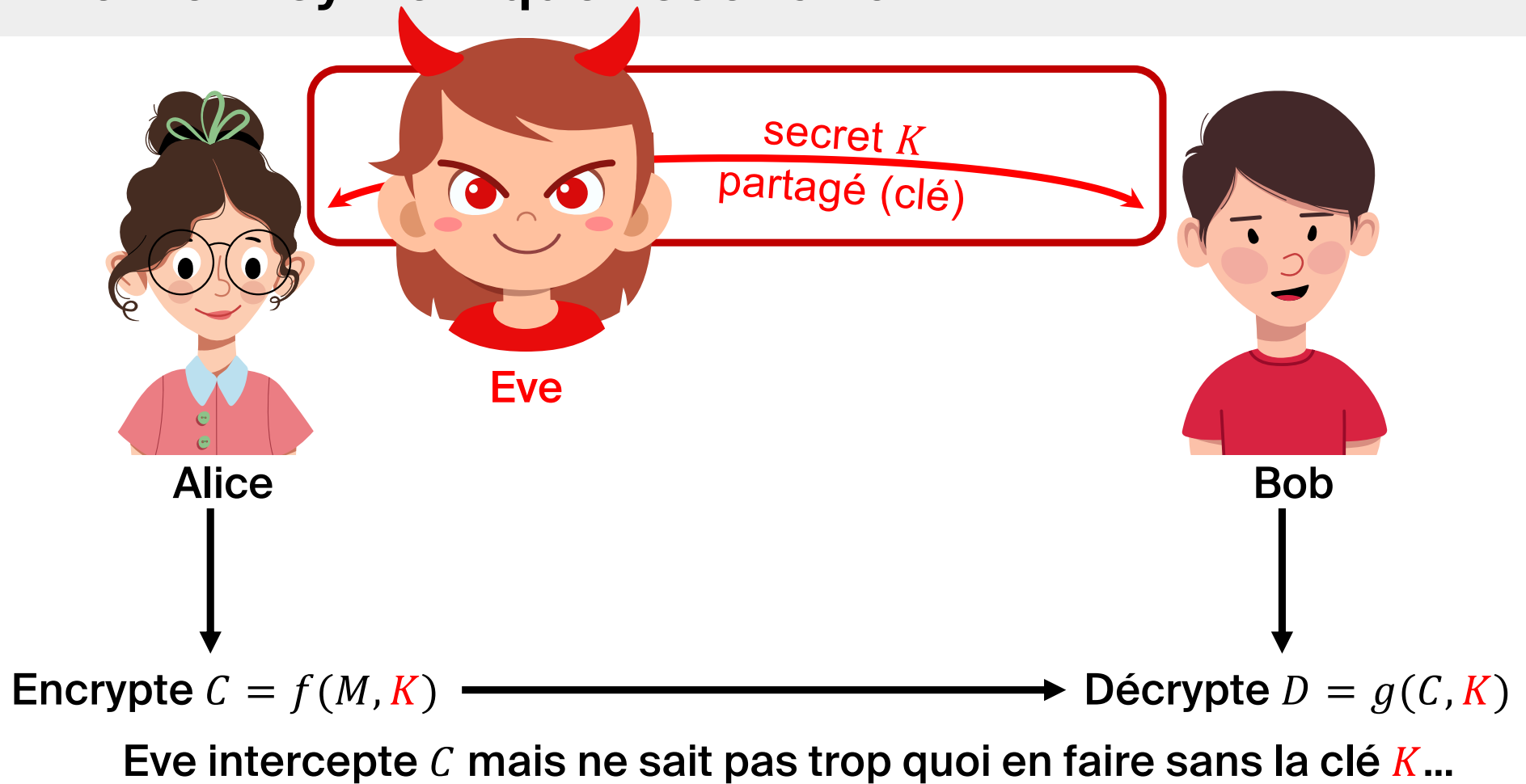


- Une seule clé pour chiffrer / déchiffrer
- Exemples : César, OTP, AES
- La taille du message chiffré est **la même** que celle du message d'origine
- Problème : **comment partager la clé sans la divulguer ?**

Aujourd'hui

- Chiffrement symétrique
- **Protocole de Diffie-Hellman**
- Fonctions de hachage cryptographiques
- Chiffrement asymétrique

Chiffrement symétrique : scénario



Protocole Diffie-Hellman : version simplifiée

- Alice et Bob désirent échanger des informations de manière confidentielle **sans** disposer d'une clé secrète K échangée au préalable.



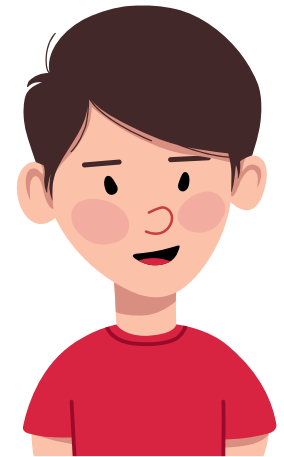
Supposons qu'Eve a raté l'école et n'a jamais appris à diviser. En revanche, elle est super douée pour intercepter toutes les communications entre Alice et Bob. Ils ne peuvent pas échanger un mot sans qu'Eve n'en soit à l'écoute.

Protocole Diffie-Hellman : version simplifiée

On a donc besoin d'**opérations difficilement inversibles**
ou « **opérations à sens unique** ».



Même si Eve intercepte M , N_{AM} et N_{BM} , **tant qu'elle ne sait pas diviser**, le secret reste bien protégé.



1. Ils se mettent d'accord sur un 3^{ème} nombre public M

M

1. Alice choisit un nombre secret N_A

2. Alice calcule $N_{AM} = N_A \cdot M$

3. Alice envoie N_{AM} à Bob

4. Alice calcule $N_A \cdot N_{BM} = N_A \cdot N_B \cdot M = K$

1. Bob choisit un nombre secret N_B

2. Bob calcule $N_{BM} = N_B \cdot M$

3. Bob envoie N_{BM} à Alice

4. Bob calcule $N_B \cdot N_{AM} = N_A \cdot N_B \cdot M = K$

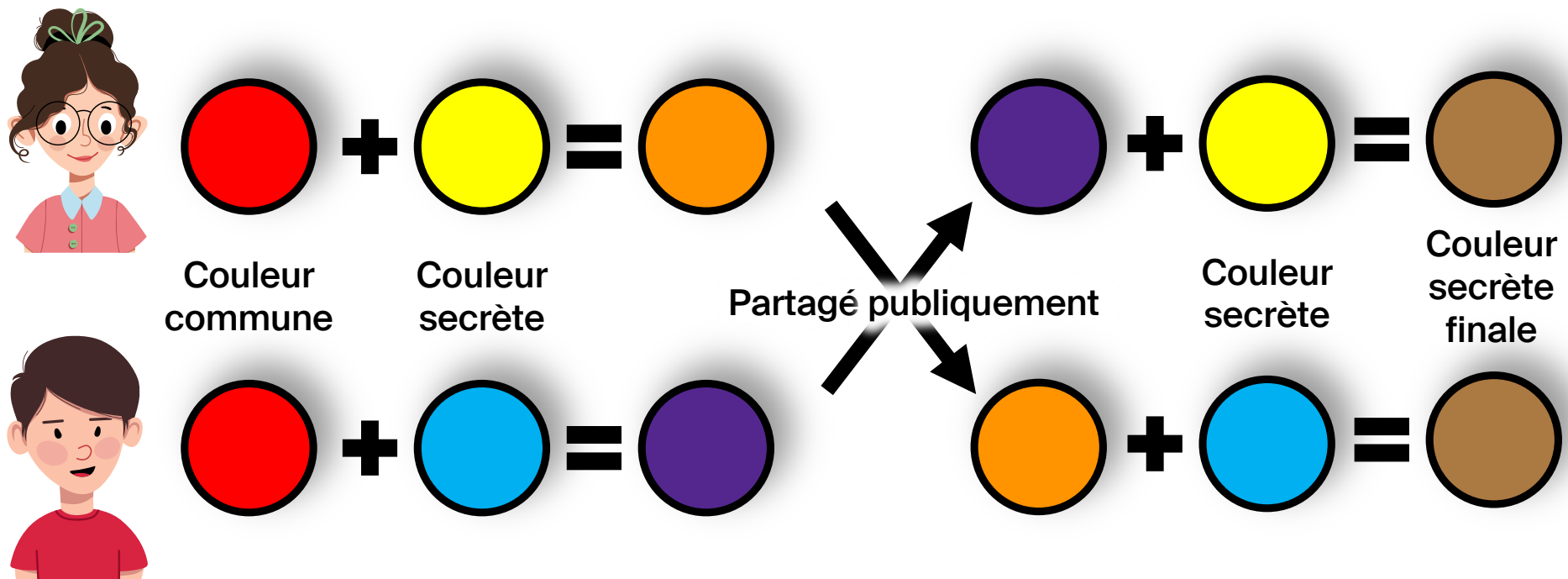
N_{AM}

N_{BM}

égaux = **secret partagé**

Diffie-Hellman avec des couleurs

- Les couleurs sont facilement mélangeables, mais difficilement séparables



Arithmétique modulaire

- Soit P un grand nombre premier. Sur l'ensemble $\{0, 1, \dots, P - 1\}$, on définit l'addition, la multiplication et l'exponentiation **modulo P** :

Exemples (avec $P = 5$) :

$a + b \pmod{P}$	$a \cdot b \pmod{P}$	$a^b \pmod{P}$
$4 + 3 \pmod{5} = 2$	$4 \cdot 3 \pmod{5} = 2$	$4^3 \pmod{5} = 4$

- pas de dépassement de capacité
- toutes des opérations faciles à exécuter

Arithmétique modulaire

$$a^b \pmod{P}$$

$$4^3 \pmod{5} = 4$$

- Il se trouve que l'opération $a^b \pmod{P} = c$ est **difficile à inverser** (i.e., si on nous donne P, a et c , **il est difficile de retrouver b**).

Voilà donc **l'opération à sens unique** que nous allons utiliser.

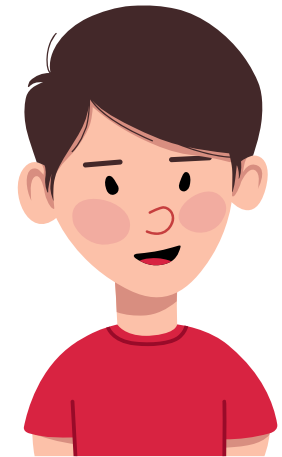
Remarques

- La difficulté de l'opération d'inversion dépend des nombres P et a choisis (c'est le problème du **logarithme discret**).
- Il existe par contre des **algorithmes efficaces** pour trouver de **grands nombres premiers** et donc réaliser concrètement ce qui va suivre !

Protocole Diffie-Hellman



Même si Eve intercepte P , Q , N_{QA} et N_{QB} , tant qu'elle ne sait pas résoudre le **problème du logarithme discret en un temps raisonnable**, le secret reste bien protégé.



1. Alice et Bob choisissent ensemble un grand nombre premier P , ainsi qu'un autre nombre Q entre 1 et $P - 1$.

P, Q

1. Alice choisit un nombre secret $1 < N_A < P - 1$

2. Alice calcule $N_{QA} = Q^{N_A} \pmod{P}$

3. Alice envoie N_{QA} à Bob

4. Alice calcule $N_{QB}^{N_A} = Q^{N_A \cdot N_B} \pmod{P} = K$

1. Bob choisit un nombre secret $1 < N_B < P - 1$

2. Bob calcule $N_{QB} = Q^{N_B} \pmod{P}$

3. Bob envoie N_{QB} à Alice

4. Bob calcule $N_{QA}^{N_B} = Q^{N_A \cdot N_B} \pmod{P} = K$

égaux = **secret partagé**

Aujourd'hui

- Chiffrement symétrique
- Protocole de Diffie-Hellman
- **Fonctions de hachage cryptographiques**
- Chiffrement asymétrique

Fonctions de hachage cryptographiques

- Une fonction de hachage prend **n'importe quelle donnée** (texte, vidéos, musique...) et produit une **empreinte numérique courte** (ex. 256 bits pour SHA-256)
- Propriétés essentielles :
 - **Déterministe** : même entrée → même sortie
 - **Sens unique** : impossible de retrouver l'entrée à partir du hash
 - **Résistance aux collisions** : très difficile de trouver deux entrées différentes ayant le même hash
- Exemples : SHA-256, SHA-3, **MD5** 🙅, **SHA-1** 🙅

Fonctions de hachage cryptographiques

Déterministe : même entrée → même sortie



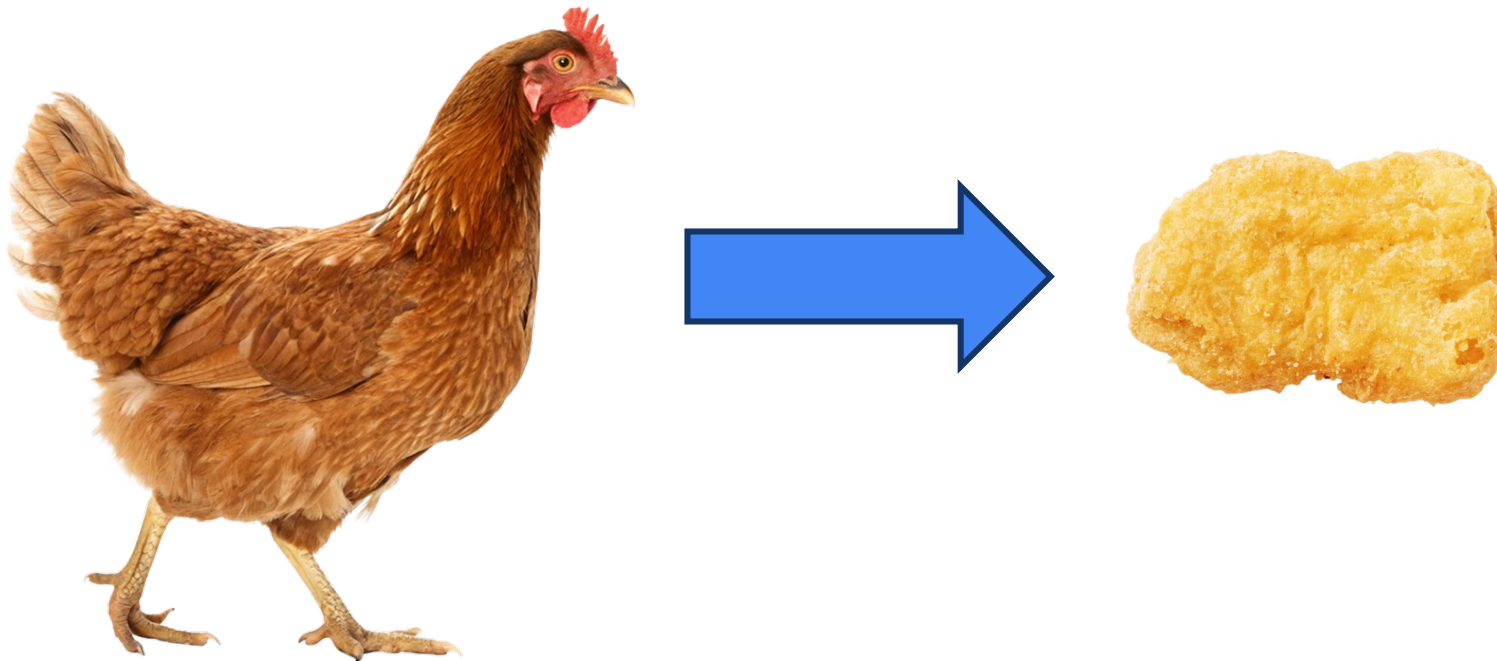
```
$ echo -n "bonjour" | sha256sum
```

```
2cb4b1431b84ec15d35ed83bb927e27e8967d75f4bcd9cc4b25c8d879ae23e18 -
```

Essayez sur votre machine : vous devriez obtenir exactement **le même résultat**.

Fonctions de hachage cryptographiques

Sens unique : impossible de remonter à l'entrée



Fonctions de hachage cryptographiques

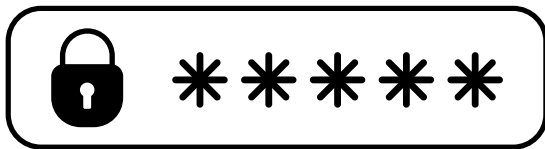
Résistance aux collisions : très difficile de trouver deux entrées différentes ayant le même hash



af8691b9fe48994e5b5637386e0c4d92f2d1a88253f4b2feec61a1ee1f8c69d

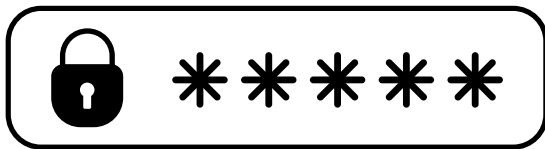
46990b7751ae578ef7de9369e7fa3500ac3a8e2187ab40e94d19128293eedbbf

À quoi servent les fonctions de hachage ?



- **Vérification d'intégrité** : si un fichier est modifié, son hash change
- **Stockage de mots de passe** : on stocke `hash(mot de passe)` au lieu du mot de passe lui-même
- **Signature numérique** : on signe le hash d'un message, pas le message complet

Fonctions de hachage : Stockage de mots de passe



- En tant que fournisseur de service, on ne doit **jamais stocker les mots de passe en clair**
- Une fuite = accès immédiat à tous les comptes
- On stocke seulement le hachage du mot de passe
`stored = hash("azerty123")`
- Lors de la connexion, on compare :
`hash(input_password) == stored`
- Mais **attention** : si deux utilisateurs ont le **même mot de passe**, on obtient le **même hash** – ce qui révèle trop d'informations !

Fonctions de hachage : Stockage de mots de passe



- Ajouter du **sel** (salt) pour renforcer la sécurité
- Le sel est une **valeur aléatoire** propre à chaque utilisateur, ajoutée au mot de passe **avant hachage**

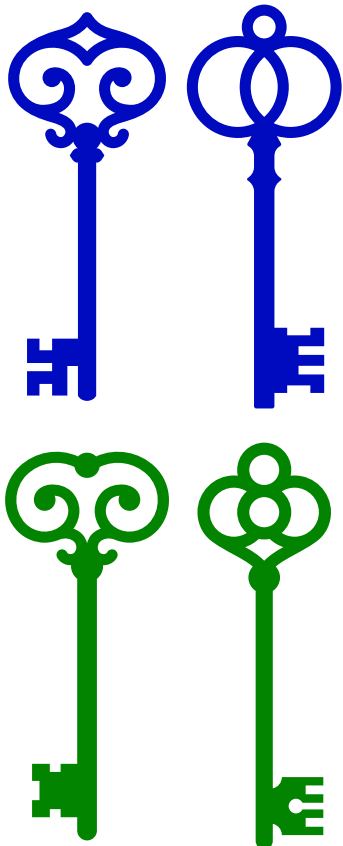
```
salt = random()  
stored = hash(salt + password)
```

- Le sel est **stocké en clair** avec le hash
- Avantages :
 - Rend les **attaques par dictionnaire** ou rainbow tables inutiles
 - **Même** mot de passe → hash **différent** selon l'utilisateur

Aujourd'hui

- Chiffrement symétrique
- Protocole de Diffie-Hellman
- Fonctions de hachage cryptographiques
- **Chiffrement asymétrique**

Cryptographie asymétrique

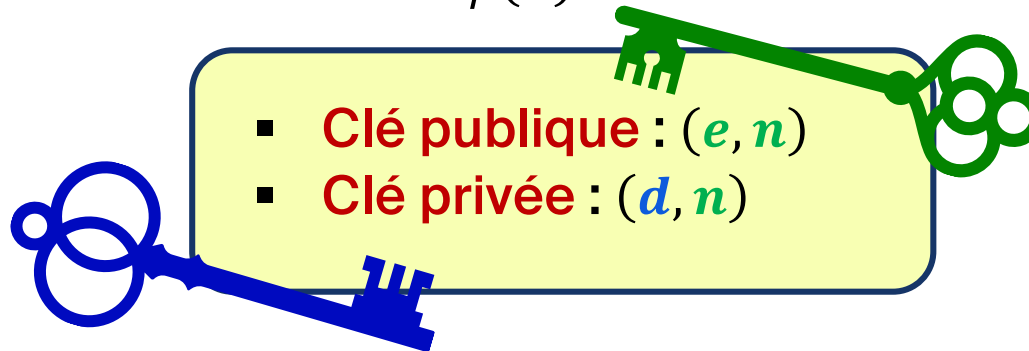


- Chaque utilisateur possède deux clés :
 - Une **clé publique** (diffusée librement)
 - Une **clé privée** (gardée secrète)
- Utilisations typiques :
 - **Chiffrement** → avec la **clé publique du destinataire**
 - **Signature** → avec sa **propre clé privée**
- **Avantage** : plus besoin de partager un secret à l'avance
- **Inconvénient** : calculs plus lourds que les méthodes symétriques
- La **sécurité** repose sur :
 - Multiplier deux grands nombres premiers : **facile**
 - Retrouver les facteurs à partir du produit :
très difficile (en pratique)

Cryptographie asymétrique : RSA



- Alice choisit deux grands nombres premiers p et q
- Elle calcule :
 - $n = p \cdot q$
 - $\varphi(n) = (p - 1) \cdot (q - 1)$
- Elle choisit un exposant public e , tel que :
 - $1 < e < \varphi(n)$, et $\gcd(e, \varphi(n)) = 1$
- Elle calcule ensuite l'exposant privé d , tel que :
 - $e \cdot d \equiv 1 \pmod{\varphi(n)}$



Cryptographie asymétrique : RSA

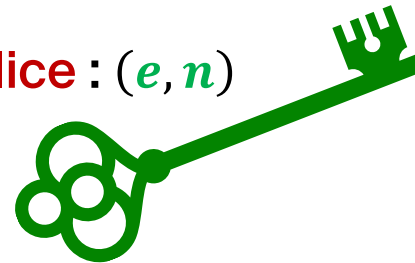
- Bob veut envoyer un message secret à Alice



- Il utilise la **clé publique d'Alice** : (e, n)
- Il chiffre son message m :

$$c = m^e \bmod n$$

c



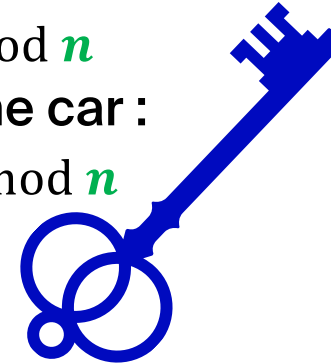
- En **pratique**, on ne chiffre pas directement un message long avec RSA
- On chiffre une **clé de session** → utilisée ensuite avec AES

- Alice reçoit c et utilise sa clé privée (d, n) pour déchiffrer :

$$m = c^d \bmod n$$

- Cela fonctionne car :

$$m^{e \cdot d} \equiv m \bmod n$$



Signature numérique



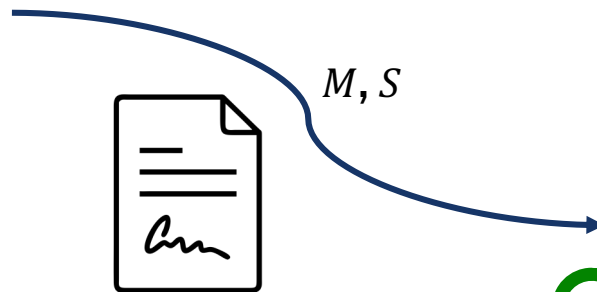
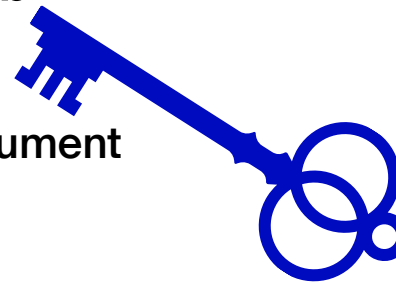
- On ne chiffre pas le message, mais son empreinte (hash) **avec sa clé privée**
- Tout le monde peut **vérifier** avec la clé publique
- Pourquoi c'est utile ?
 - Garantit **l'authenticité** : seul le propriétaire de la clé privée peut signer
 - Garantit **l'intégrité** : toute modification du message entraîne un hash différent
 - Garantit **la non-répudiation** : la signature prouve que l'auteur est bien à l'origine du message

Signature numérique

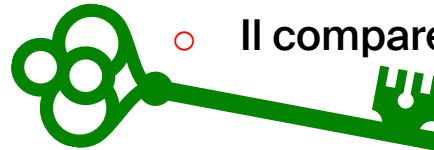
- Alice veut envoyer un document signé à Bob



- Alice calcule $\text{hash}(M)$ où M est le document
- Elle chiffre ce hash avec sa **clé privée**. Cela produit la signature S
- Elle envoie à Bob le document M et la signature S



- Bob vérifie la signature :
 - Il calcule $\text{hash}(M)$ de son côté
 - Il déchiffre S avec la **clé publique** d'Alice
 - Il compare les deux résultats



Aujourd'hui

- Chiffrement symétrique
- Protocole de Diffie-Hellman
- Fonctions de hachage cryptographiques
- Chiffrement asymétrique

Résumé Semaine 13 – ICC-T

- Les besoins en **sécurité** : confidentialité, authenticité, intégrité, non-répudiation
- Les outils cryptographiques :
 - Chiffrement **symétrique** (AES) : rapide, **clé partagée**
 - Diffie-Hellman : permet un **échange de clé sans la transmettre**
 - Fonctions de **hachage** : empreinte numérique, vérification d'intégrité
 - Chiffrement **asymétrique** (RSA) : deux clés, mais plus lent
 - **Signature** numérique : prouve l'identité et l'intégrité d'un message
- Certains cryptosystèmes (ex. : RSA, Diffie–Hellman) reposent sur des **problèmes mathématiques difficiles** (ex. : factorisation, logarithme discret) ; d'autres (AES, OTP) s'appuient sur des **constructions algorithmiques** ou de **l'aléa parfait**.

rafael.pires@epfl.ch

EPFL



Merci