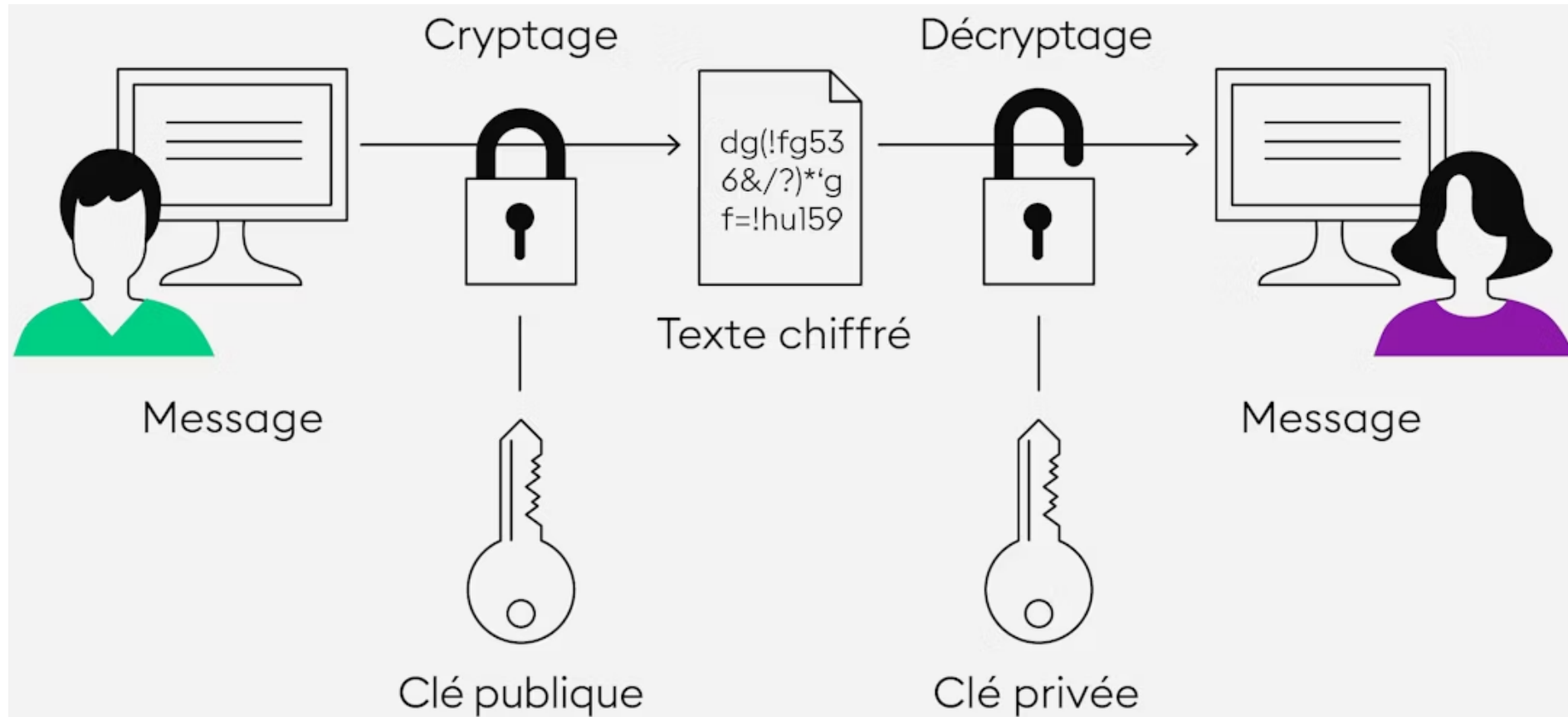


Chiffrement asymétrique

Le **chiffrement asymétrique**, également appelé **cryptographie à clé publique**, est un système de cryptographie qui utilise **deux clés distinctes** :

1. Une **clé publique** (connue de tous) pour **chiffrer les données**.
2. Une **clé privée** (gardée secrète) pour **déchiffrer les données**.

Ces deux clés sont **mathématiquement liées**, mais il est pratiquement **impossible de deviner la clé privée à partir de la clé publique**.



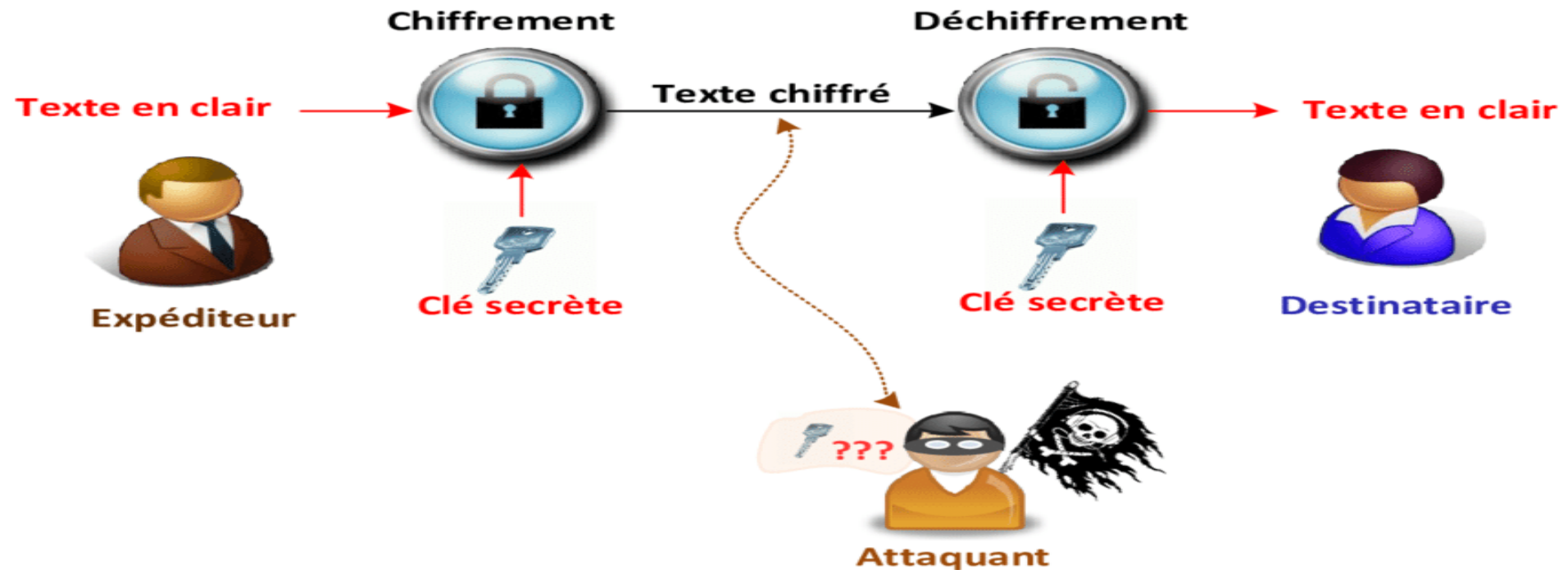
Fonctionnement de chiffrement asymétrique

1. Chiffrement des données :

1. L'expéditeur utilise la **clé publique** du destinataire pour chiffrer un message.
2. Une fois chiffré, ce message ne peut être déchiffré qu'avec la **clé privée** correspondante.
3. Exemple : Vous voulez envoyer un message sécurisé à **Alice**.
Vous utilisez sa clé publique pour le chiffrer.

2. Déchiffrement des données :

Alice reçoit le message et utilise sa **clé privée** pour le déchiffrer.



Étapes détaillées :

1. Génération des clés par le récepteur (Alice) :

Alice utilise un algorithme cryptographique (comme RSA ou ECC) pour créer une paire de clés :

- **Clé publique** : K_{pub_Alice} → Elle sera distribuée à tout le monde.
- **Clé privée** : K_{priv_Alice} → Elle reste sur son appareil, en sécurité.

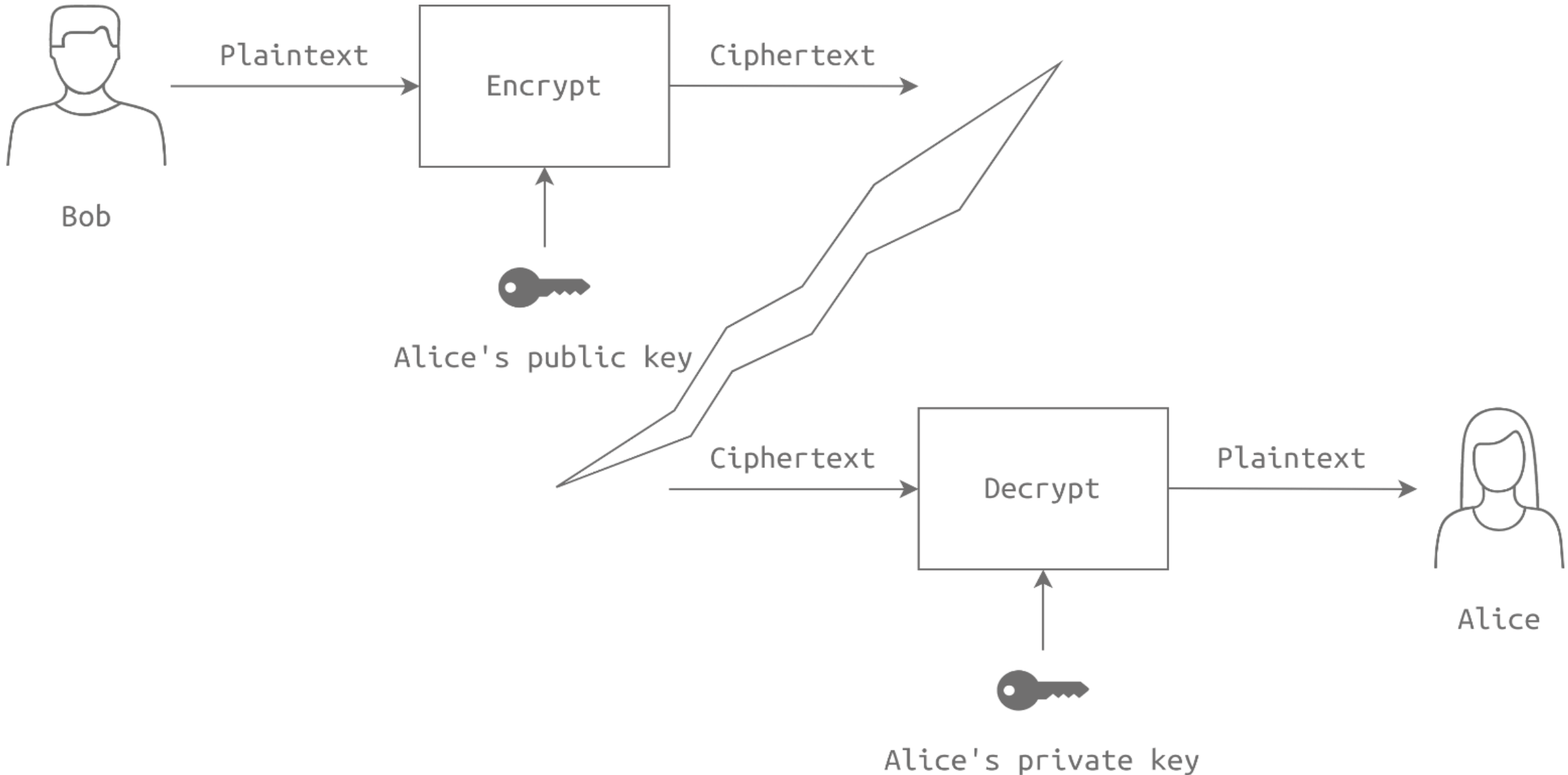
2. Partage de la clé publique :

- Alice envoie sa clé publique (K_{pub_Alice}) à Bob par un canal non sécurisé (par exemple, un e-mail ou un site web).
- **Même si quelqu'un intercepte cette clé publique, cela ne compromet pas la sécurité, car seule la clé privée (K_{priv_Alice}) peut déchiffrer les messages.**

3. Envoi du message chiffré par l'émetteur (Bob) :

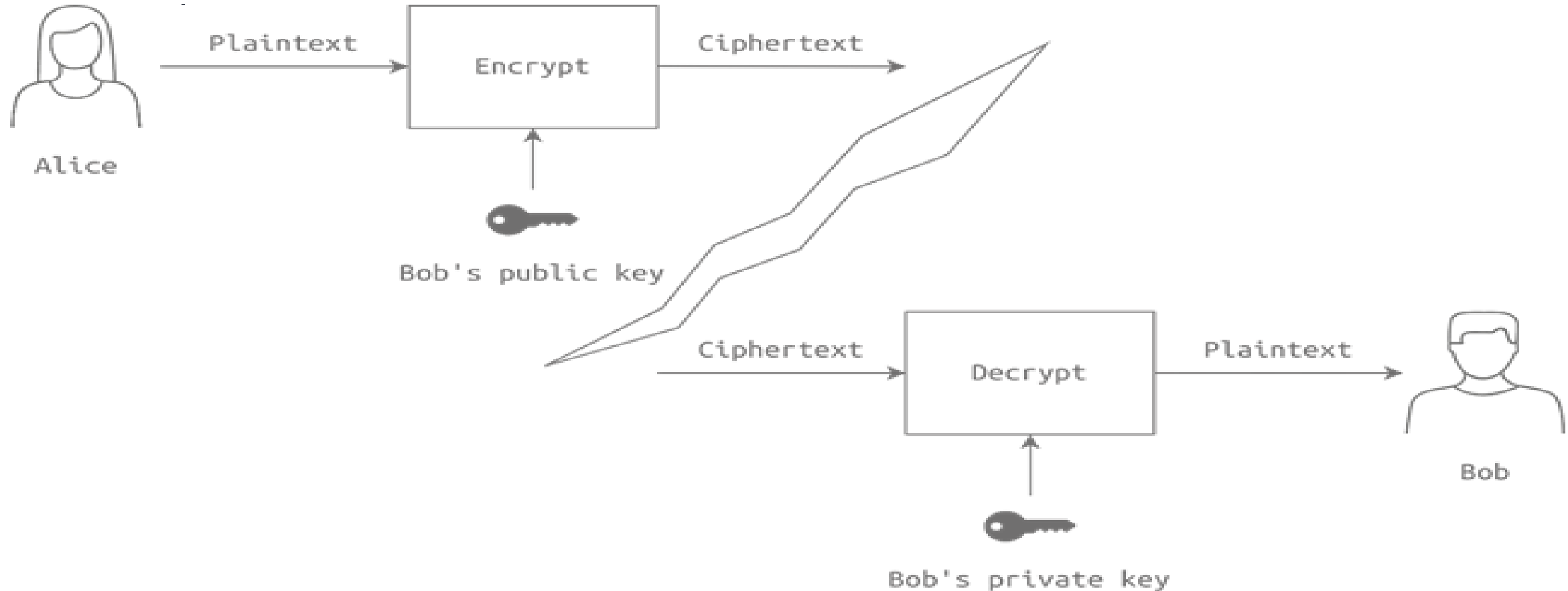
- Bob utilise la clé publique d'Alice pour chiffrer son message.
- Une fois chiffré, seul Alice peut le déchiffrer, car elle est la seule à posséder la clé privée correspondante.

Lorsque Bob veut répondre à Alice, il crypte ses messages en utilisant la clé publique d'Alice, et Alice peut les décrypter en utilisant sa clé privée.



Alice souhaite **garantir la confidentialité** de **ses échanges avec Bob**.
Elle **crypte le message** à l'aide de la **clé publique de Bob**,
et **Bob le décrypte** à l'aide de sa **clé privée**.

La clé publique de Bob est censée être **publiée** dans une **base de données publique** ou sur son **site Web**



Avantages clés de chiffrement asymétrique :

- **Pas de partage préalable de secret**

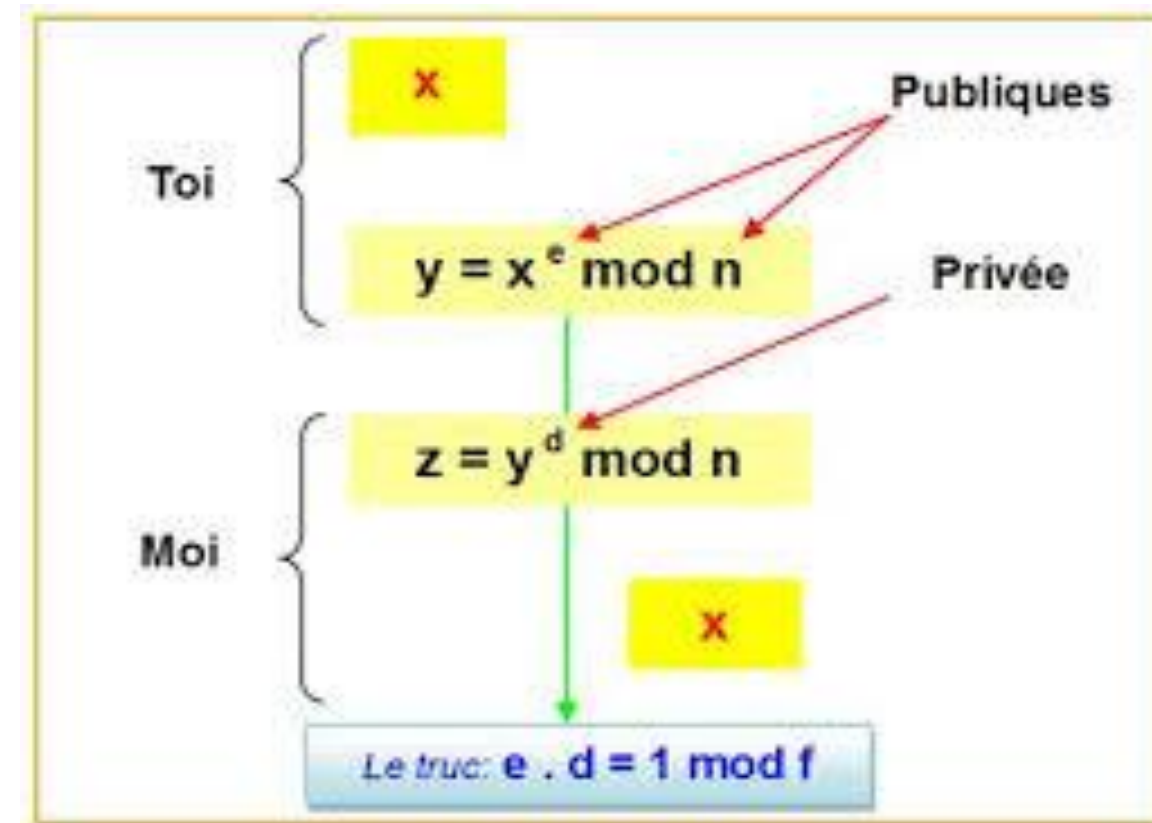
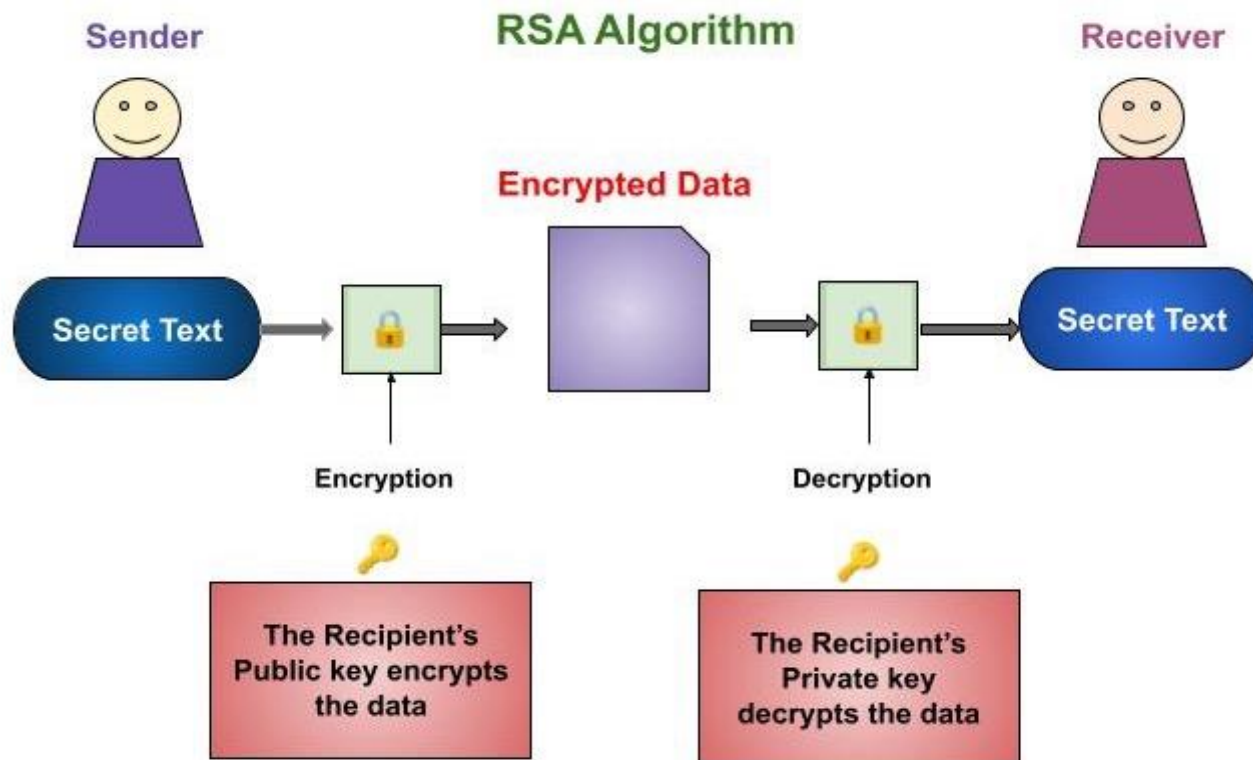
Sécurise l'échange de clés (problème du partage initial en symétrique).

- **Authentification possible (via signature)**
- **Particulièrement adapté à la distribution sécurisée de clés symétriques**

RSA

RSA (du nom de ses inventeurs Rivest, Shamir et Adleman) est un **algorithme de cryptographie asymétrique** utilisé pour **sécuriser les communications**.

RSA repose sur des **principes mathématiques**, notamment la **difficulté de factoriser de grands nombres premiers**, ce qui rend **cet algorithme robuste face aux attaques**.



Étapes principales de RSA

Génération des clés :

1. Choisir deux grands nombres premiers p et q .
 2. Calculer $n = p \times q$. n est la base de la clé publique et de la clé privée.
 3. Calculer $\phi(n) = (p - 1) \times (q - 1)$, qui est la fonction indicatrice d'Euler.
 4. Choisir un entier e tel que $1 < e < \phi(n)$ et e soit premier avec $\phi(n)$.
 5. Calculer d , l'inverse modulaire de e modulo $\phi(n)$: $d \times e \bmod \phi(n) = 1$.
- Clé publique : (e, n) .
 - Clé privée : (d, n) .

Exemple pratique pas à pas

on prend de petits nombres premiers : $p=61$ et $q=53$

Étape 1 : calcul de n $n = p \times q = 61 \times 53 = 3233$

Étape 2 : calcul de $\varphi(n)$

$$\varphi(n) = (p - 1) \times (q - 1) = 60 \times 52 = 3120$$

1. Choix de deux grands nombres premiers

- Soient p et q deux nombres premiers distincts.

2. Calcul de n

- $n = p \times q$.
- n servira de **module** pour la clé publique et la clé privée

3. Calcul de $\varphi(n)$ (fonction indicatrice d'Euler)


- $\varphi(n) = (p - 1) \times (q - 1)$.

4. Choix de l'exposant public e

- Choisir un entier e tel que $1 < e < \varphi(n)$
- et $\gcd(e, \varphi(n)) = 1$ (c'est-à-dire que e est premier avec $\varphi(n)$).

Étape 3 : choix de e

On cherche un e premier avec 3120, par exemple :

- Tester $e = 17$
- $\gcd(17, 3120) = 1$ 

Donc on fixe $e = 17$.

Comment????

Lister quelques candidats e

En pratique on teste d'abord des petits nombres premiers :

$$e \in \{3, 5, 17, 257, 65537, \dots\}$$

Ces valeurs sont souvent choisies parce qu'elles facilitent les calculs (surtout 65537).

Vérifier $\gcd(e, \varphi(n))$ par l'algorithme d'Euclide

Exemple avec $\varphi(n) = 3120$ et $e = 17$

On veut $\gcd(17, 3120)$. On pose des divisions euclidiennes successives :

1. $3120 = 17 \times 183 + 9$

2. $17 = 9 \times 1 + 8$

3. $9 = 8 \times 1 + 1$

4. $8 = 1 \times 8 + 0$

Lorsque le reste devient 0, le dernier **reste non-nul** (ici 1) est le gcd.

Ici $\gcd(17, 3120) = 1$, donc **17 est accepté** comme exposant.

5. Calcul de l'exposant privé d

- Déterminer d tel que $d \times e \equiv 1 \pmod{\varphi(n)}$
- Autrement dit, d est l'inverse modulaire de e modulo $\varphi(n)$.

Étape 4 : calcul de d

Il faut résoudre l'équation : $d \times 17 \equiv 1 \pmod{3120}$.

Autrement dit, il existe un entier k tel que :

$$17d - 3120k = 1,$$

On applique l'algorithme d'Euclide étendu :

$$3120 = 17 \times 183 + 9$$

$$17 = 9 \times 1 + 8$$

$$9 = 8 \times 1 + 1$$

$$8 = 1 \times 8 + 0$$

Puis on remonte pour exprimer 1 comme combinaison :

$$1 = 9 - 8 \times 1$$

$$= 9 - (17 - 9 \times 1) = 2 \times 9 - 17$$

$$= 2 \times (3120 - 17 \times 183) - 17$$

$$= 2 \times 3120 - 367 \times 17$$

Donc $-367 \times 17 \equiv 1 \pmod{3120}$,

d'où $d \equiv -367 \equiv 3120 - 367 = 2753$.

On prend $d = 2753$.

6. Clés

- Clé publique : (e, n)
- Clé privée : (d, n)

Récapitulatif des clés

- Clé publique $(e, n) = (17, 3233)$
- Clé privée $(d, n) = (2753, 3233)$

Déchiffrement avec la clé privée :

OpenSSL est une boîte à outils open-source très répandue qui implémente les protocoles SSL/TLS et fournit une bibliothèque complète de fonctions cryptographiques. Elle inclut à la fois :

- **Une bibliothèque** (libcrypto) utilisable depuis vos programmes (C, Python via bindings, etc.)
- **Une interface en ligne de commande** (openssl) permettant de générer des clés, chiffrer/déchiffrer, signer, vérifier, gérer des certificats, etc.

Chiffrement et déchiffrement RSA

1. Générer une clé privée RSA 2048 bits

```
openssl genpkey \  
-algorithm RSA \  
-out rsa_private.pem \  
-pkeyopt rsa_keygen_bits:2048
```

- **openssl genpkey** Lance la commande de génération de clé privée (utility “genPKEY”).
- **-algorithm RSA** Spécifie qu’on veut générer une paire de clés **RSA**.
- **-out rsa_private.pem** Indique le fichier de sortie (rsa_private.pem) qui contiendra la **clé privée** au format PEM.
- **-pkeyopt rsa_keygen_bits:2048** Option pour définir la taille de la clé à **2048 bits** (niveau de sécurité courant).

Chiffrer et Déchiffrer :


1.Chiffrement avec la clé publique :

Si nous avons la clé publique du destinataire, nous pouvons chiffrer un fichier texte :

bash

openssl pkeyutl -encrypt -in plaintext.txt -out ciphertext -inkey public-key.pem -pubin

- **-encrypt** : Indique que nous souhaitons chiffrer.
- **-in plaintext.txt** : Fichier texte en clair à chiffrer.
- **-out ciphertext** :
Fichier de sortie contenant le texte chiffré.
- **-inkey public-key.pem** :
Utilise la clé publique pour le chiffrement.
- **-pubin** : Spécifie que la clé fournie est publique.

 public-key - Notepad

File Edit Format View Help

-----BEGIN PUBLIC KEY-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1r7ZrNs7HbpNUwX4DnFi
6o09TqllxrubL2vgzVLEiB329Meh27AhYlVtpiaNKKYLoUYAS4i87Ep4EkGwH8Dy
sos2GWBEi92EfAhcmYEJpfwvLGB4vHhHx4kbjrkHjQhC3CUFsiox6l7HaInV7TF/
rgBEV8xSbzXKRgbX7nNnyVnZv2E2XidJTNxFuTjLxTImT9AnvBL1TaTg7u1kyClD
bYgEERNGXF04F2v2yk50kUf+P0qqfQZTDrac5A2RvaypSeS1HH1wF6J2ItLowVfk
juhuYf3UTuos8dTDW4LT+S4gS0Es13/DxlfA21kEiegd18o35SJi0HG6Ik+PoQDu
OwIDAQAB

-----END PUBLIC KEY-----

Déchiffrement avec la clé privée :

Le destinataire peut déchiffrer le fichier avec sa clé privée :

openssl pkeyutl -decrypt -in ciphertext -inkey private-key.pem -out decrypted.txt

- **-decrypt** : Indique que nous souhaitons déchiffrer.

- **-in ciphertext** : Fichier chiffré à déchiffrer.

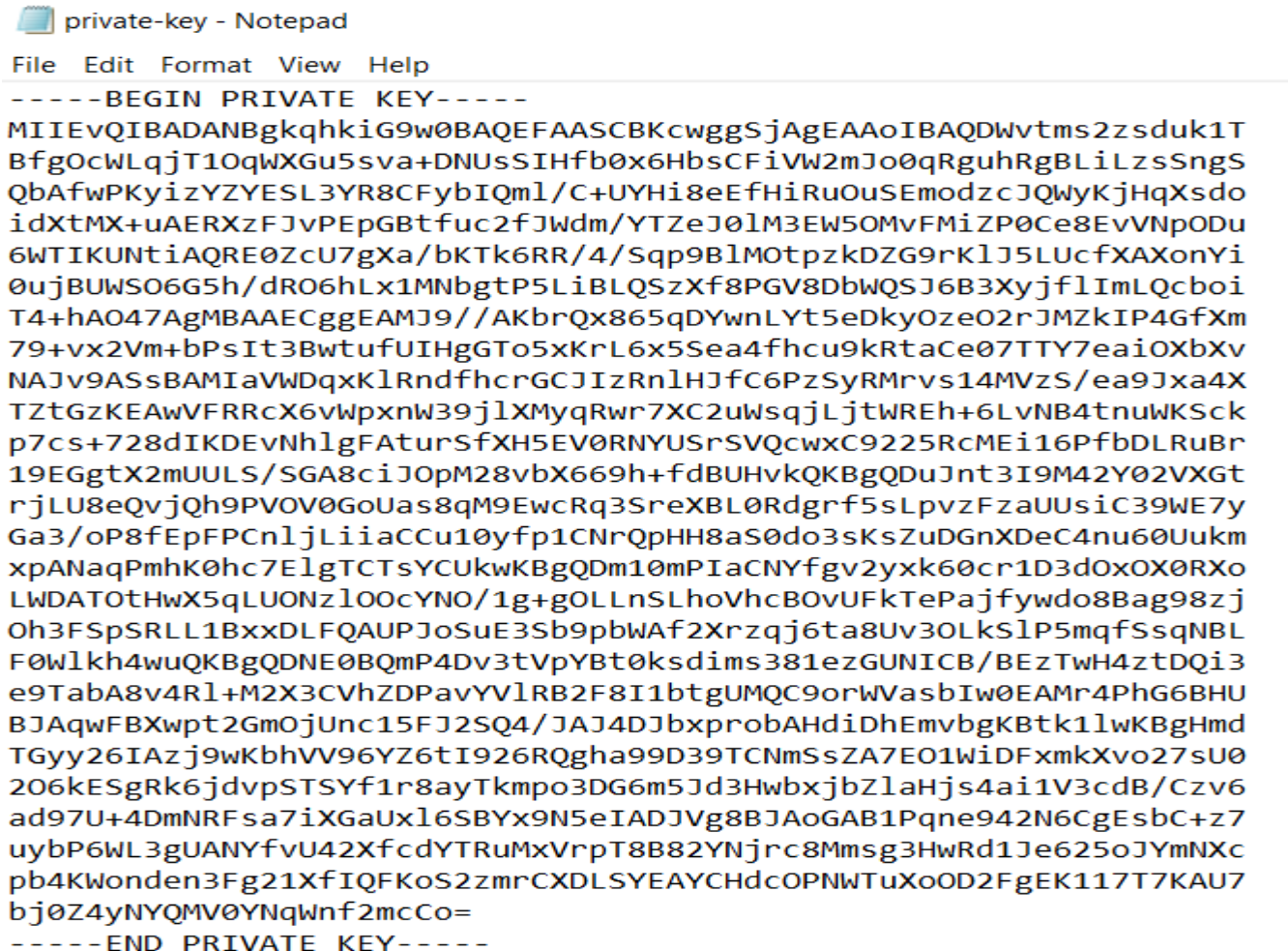
- **-inkey private-key.pem** :

Utilise la clé privée du destinataire.

- **-out decrypted.txt** :

Fichier de sortie contenant le texte en clair après déchiffrement.

Voir Lab2 Chiffrement RSA



```
private-key - Notepad
File Edit Format View Help
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBAwggSjAgEAAoIBAQDWvtms2zsduk1T
BfgOcWLqjT1OqWXGu5sva+DNUsSIHfb0x6HbsCFiVW2mJo0qRguhRgBLiLzsSngS
QbAfwPKyizYZYESL3YR8CFybIQm1/C+UYHi8eEfHiRuOuSEmodzcJQwyKjHqXsdo
idXtMX+uAERXzFJvPEpGBtfuc2fJWdm/YTZeJ0lM3EW5OMvFMiZP0Ce8EvVNpODu
6WTIKUNTiAQRE0Zcu7gXa/bKTK6RR/4/Sqp9BlM0tpzkDZG9rKlJ5LUCfXAXonYi
0ujBUWSO6G5h/dRO6hLx1MNBgtP5LiBLQSZxf8PGV8DbWQsJ6B3Xyjf1ImLQcboi
T4+hAO47AgMBAAECggEAMJ9//AKbrQx865qDYwnLYt5eDkyOzeO2rJMzKIP4GfXm
79+vx2Vm+bPsIt3BwtufUIHGGTo5xKrL6x5Sea4fhcu9kRtaCe07TTY7eaiOXbXv
NAJv9ASsBAMiavWDqxKlRndfhcrGCJIZRn1HJfC6PzSyRMrvs14MVzS/ea9Jxa4X
TZtGzKEAwVFRRcX6vWpxnW39j1XMyqRwr7XC2uwsqjLjtwREh+6LvNB4tnuWKSck
p7cs+728dIKDevNhlGfAtursfXH5EV0RNYUSrSVQcwx9225RcMEi16PfbDLRuBr
19EGgtX2mUULS/SGA8ciJOpM28vbX669h+fdBUHvkQKBgQDuJnt3I9M42Y02VXGt
rjLU8eQvjQh9PVOV0GoUas8qM9EwcRq3SreXBL0Rdgrf5sLpvzFzaUUsiC39WE7y
Ga3/oP8fEpFPCnljLiiaCCu10yfp1CNrQpHH8aS0do3sKsZuDgnXDeC4nu60Uukm
xpANaQpMhK0hc7ElgTCTsYCUkwKBgQDm10mPIaCNYfgv2yxk60cr1D3d0xOX0RXo
LWDAT0tHwX5qLUONz100cYNO/1g+gOLLnSLhoVhcB0vUFkTePajfywdo8Bag98zj
Oh3FSpSRL1BxxDLFQAUPJoSuE3Sb9pbWaf2Xrzqj6ta8Uv30Lks1P5mqfSsqNBL
F0Wlkh4wuQKBgQDNE0BQmP4Dv3tVpYBt0ksdms381ezGUNICB/BezTWh4ztDQi3
e9TabA8v4Rl+M2X3CVhZDPavYVlRB2F8I1btgUMQC9orwVasbIw0EAMr4PhG6BHU
BJAqWFBXwpt2GmOjUnc15FJ2SQ4/JAJ4DJbxprobAHdiDhEmvbgKBtk1lwKBgHmd
TGyy26IAzj9wKbhVV96YZ6tI926RQgha99D39TCNmSsZA7E01WiDFxmKXvo27sU0
206kESgRk6jdvpSTSYf1r8ayTkmpo3DG6m5Jd3HwbxjbZlaHjs4ai1V3cdB/Czv6
ad97U+4DmNRFsa7iXGaUx16SBYx9N5eIADJVg8BJAoGAB1Pqne942N6CgEsbc+z7
uybP6WL3gUANYfvU42XfcdYTRuMxVrpt8B82YNjrc8Mmsg3HwRd1Je625oJYmNXc
pb4KWonden3Fg21XfIQFKoS2zmrCXDLSEAYCHdcOPNWTuXoOD2FgEK117T7KAU7
bj0Z4yNYQMv0YNqWnf2mcCo=
-----END PRIVATE KEY-----
```

Protocole Diffie-Hellman pour l'échange de clé symétrique

En 1976, **Whitfield Diffie** et **Martin Hellman** proposent d'utiliser le **problème du logarithme discret** dans un **corps fini**, un problème calculatoire considéré difficile, comme base pour construire un **protocole d'échange de clé**.

Le problème est le suivant :

Alice et Bob veulent s'échanger un message crypté en utilisant un **algorithme nécessitant une clé K**.

Ils veulent **s'échanger cette clé K**, mais ils **ne disposent pas de canal sécurisé** pour cela.

Le protocole d'échange de clés de Diffie et Hellman répond à ce problème lorsque **K est un entier**.

Le protocole Diffie -Hellman est un **algorithme de chiffrement asymétrique**.

Il permet **l'échange d'un secret** sur un **canal public** (même si cet canal est potentiellement non sécurisé).

Le protocole **Diffie-Hellman (DH)** est largement utilisé en **cryptographie symétrique** car il répond à un **problème fondamental** dans ce domaine :
comment partager une clé secrète entre deux parties sur un canal non sécurisé?

La **cryptographie symétrique** nécessite une **clé secrète** partagée pour **chiffrer et déchiffrer** les données, et **Diffie-Hellman** offre une **solution élégante et sécurisée** à ce problème.

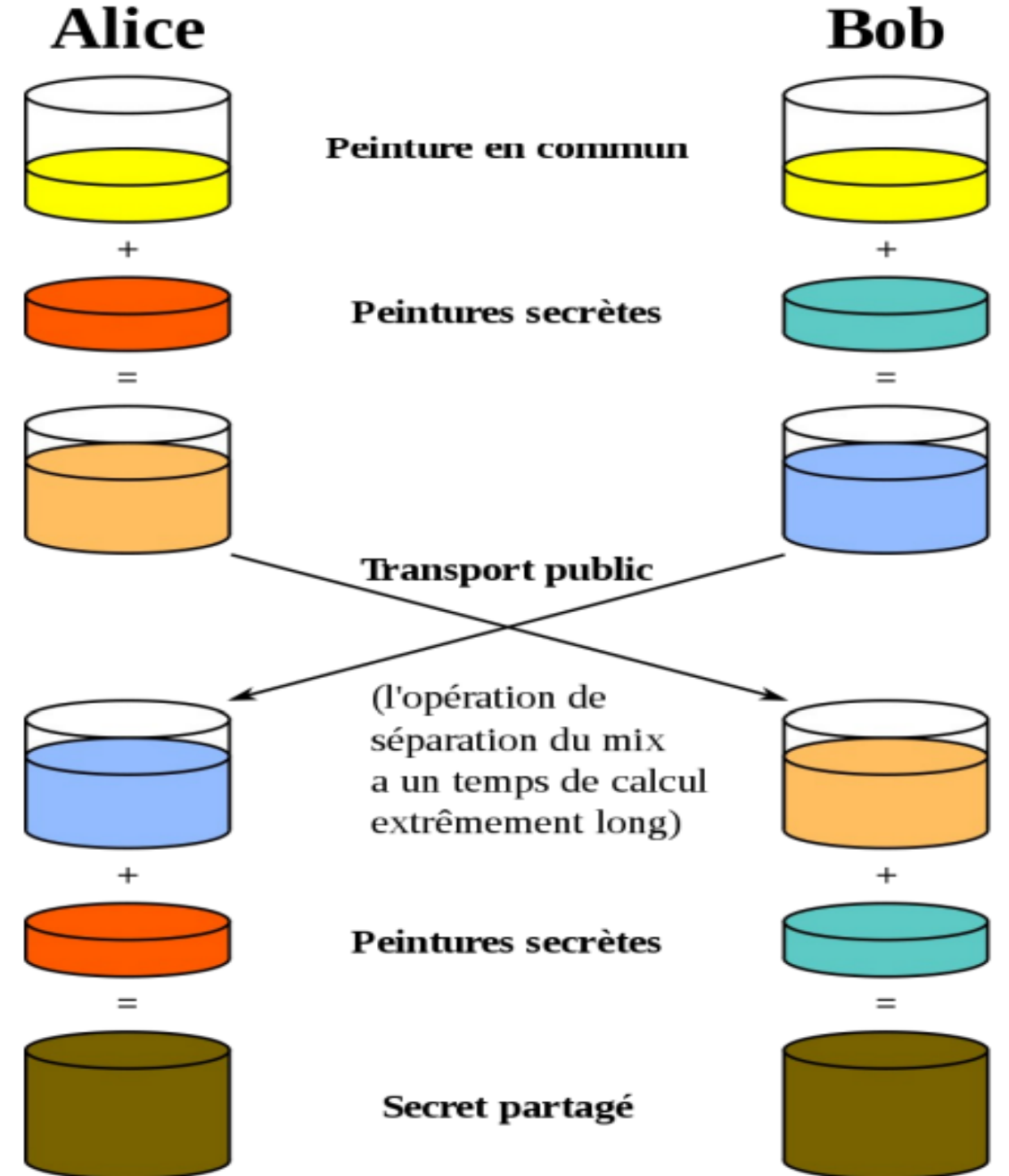


Illustration conceptuelle d'un échange de clés Diffie-Hellman

Ce protocole repose sur l'arithmétique modulaire et sur le postulat suivant :

Étant donné des entiers p (p premier), a (avec $1 \leq a \leq p-1$), x :

- il est facile de calculer l'entier y tel que $y \equiv a x [p]$
- si on connaît a, p et $y \equiv a x [p]$, il est très difficile de retrouver x , pourvu que p soit assez grand.

Retrouver x s'appelle résoudre le *problème du logarithme discret*, pour lequel on ne dispose pas d'algorithme efficace.

Voici comment Alice et Bob font pour s'échanger la clé secrète.
Ils font des actions en parallèle, que l'on décrit dans le tableau suivant :

	Alice	Bob
Étape 1	Alice et Bob choisissent un nombre premier p et un entier a tel que $1 \leq a \leq p-1$. L'échange n'est pas sécurisé.	
Étape 2	Alice choisit secrètement un nombre x_1 .	Bob choisit secrètement un nombre x_2 .
Étape 3	Alice calcule y_1 tel que : $y_1 \equiv a^{x_1} [p]$.	Bob calcule y_2 tel que : $y_2 \equiv a^{x_2} [p]$.
Étape 4	Alice et Bob s'échangent les valeurs de y_1 et y_2 . L'échange n'est pas sécurisé.	
Étape 5	Alice calcule la clé secrète $y_2^{x_1} [p]$	Bob calcule la clé secrète $y_1^{x_2} [p]$.

Le mécanisme d'échange de clés Diffie-Hellman :

1. Valeurs publiques :

Les deux parties conviennent d'un nombre premier q et d'une base g (également appelée générateur). Ces deux valeurs sont publiques et connues de tous.

2. Choix des secrets :

- Chaque partie choisit un nombre secret (par exemple, Alice choisit a et Bob choisit b).
- Ces nombres restent confidentiels.

3. Calcul des valeurs intermédiaires :

- Alice calcule $A = g^a \bmod q$ et envoie A à Bob.
- Bob calcule $B = g^b \bmod q$ et envoie B à Alice.

4. Création de la clé secrète :

- Alice utilise la valeur B reçue pour calculer la clé secrète $s = B^a \bmod q$
- Bob utilise la valeur A reçue pour calculer la clé secrète $s = A^b \bmod q$
- Grâce aux propriétés des puissances modulo q , les deux calculs donnent le même résultat.

5. Sécurité :

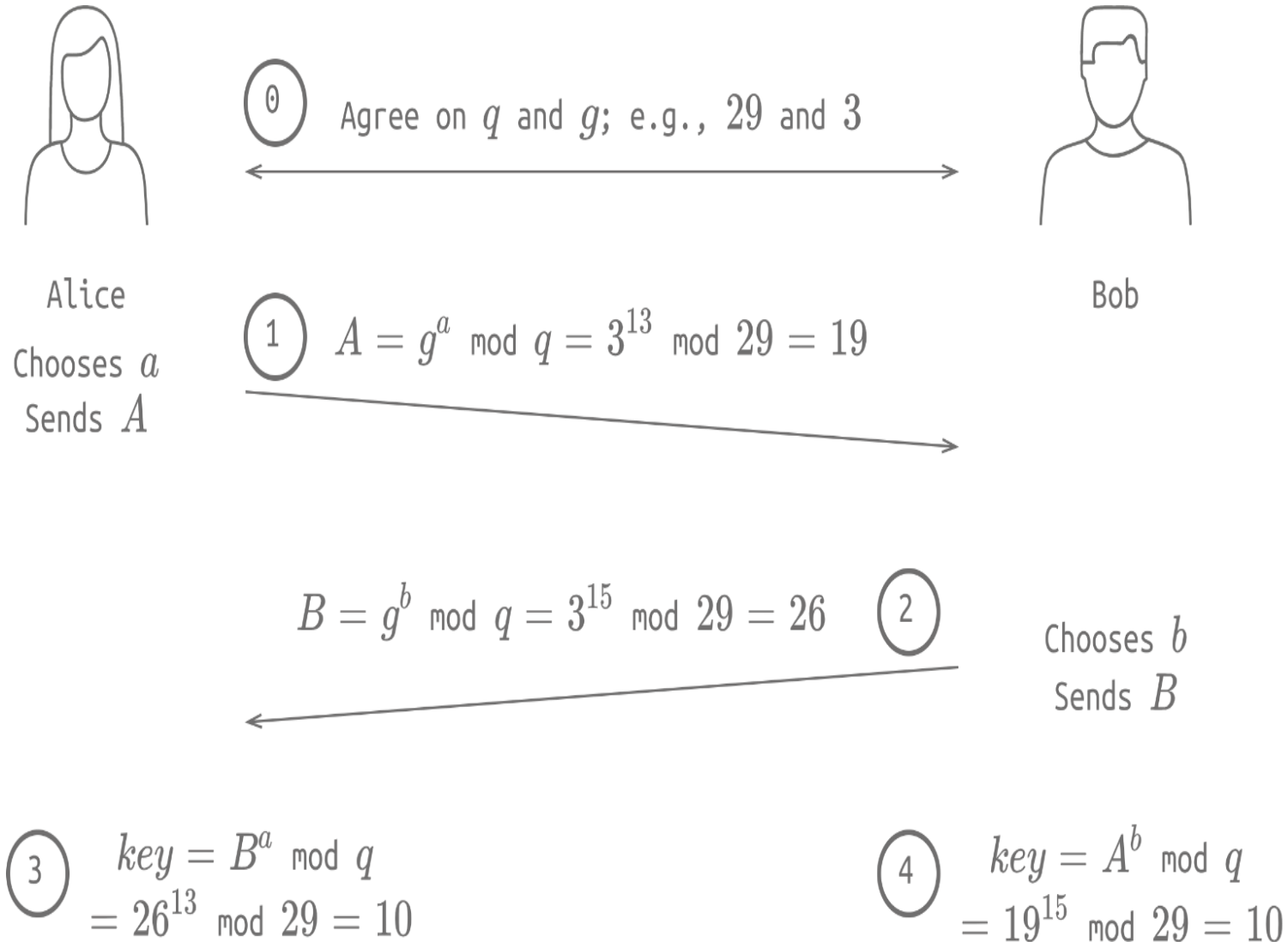
La sécurité du protocole repose sur la difficulté de résoudre le problème du logarithme discret. Même si un attaquant connaît q, g, A et B , il est extrêmement difficile de retrouver les valeurs secrètes a ou b , et donc la clé s .

Étape 0 : Choix des paramètres publics (qqq et ggg)

Alice et Bob conviennent publiquement de deux paramètres :

- **q=29** (un grand nombre premier).
- **g=3** (la base ou générateur).

Ces paramètres sont partagés et accessibles à tous, même à un éventuel attaquant.



Étape 1 : Calcul et envoi de la clé publique par Alice

1. **Alice** choisit un **secret privé a=13**, qui est **gardé secret**.

2. Alice calcule sa clé publique A :

$$A = g^a \mod q = 3^{13} \mod 29 = 19$$

3. Alice envoie A=19 à Bob.

Étape 2 : Calcul et envoi de la clé publique par Bob

1. **Bob** choisit un **secret privé b=15**, qui est **gardé secret**.

2. Bob calcule sa clé publique B

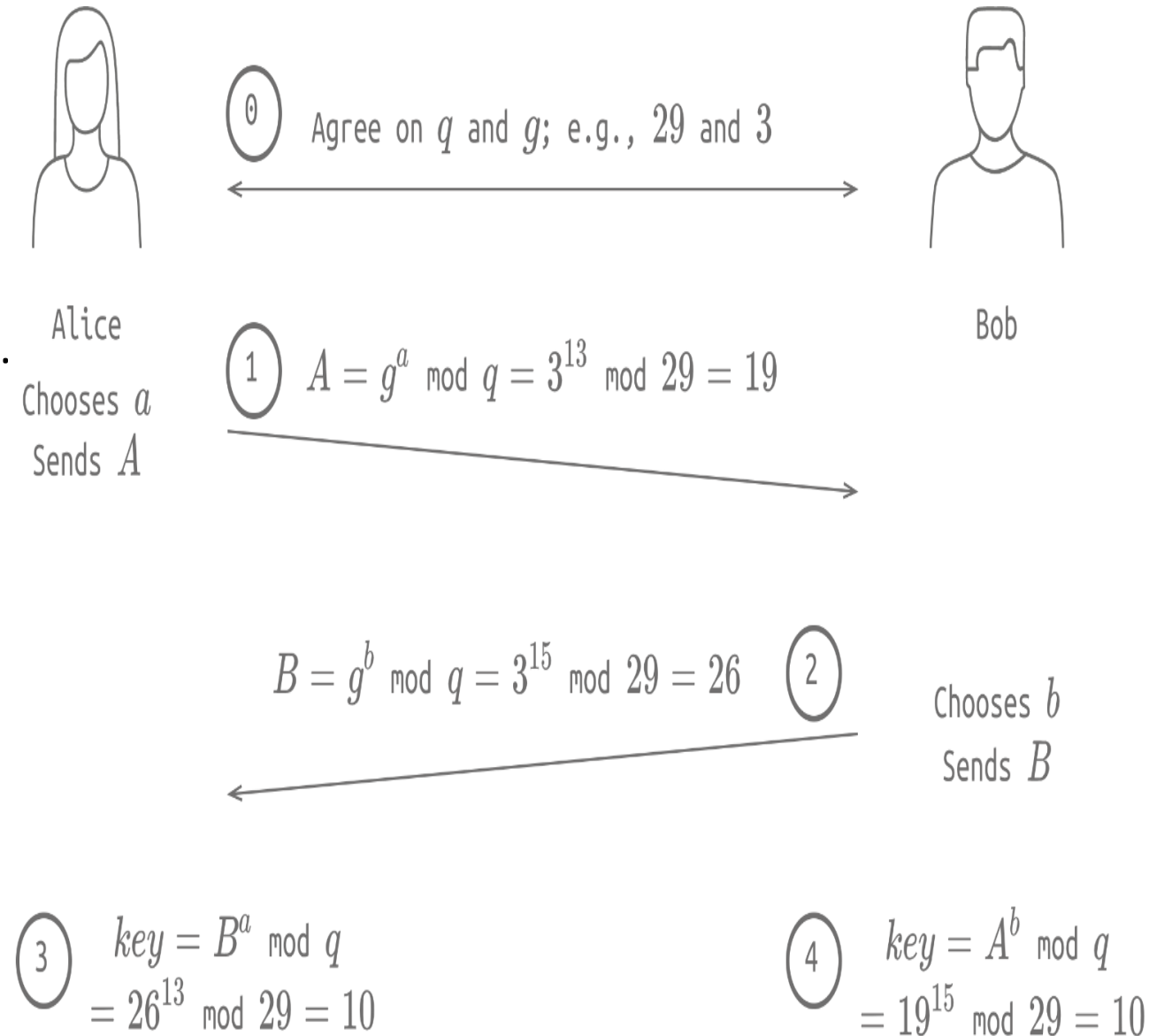
$$B = g^b \mod q = 3^{15} \mod 29 = 26$$

3. Bob envoie B=26 à Alice.

Étape 3 : Calcul de la clé partagée par Alice

Alice utilise la **clé publique** reçue de Bob (B=26) et son secret privé (a=13) pour **calculer la clé partagée K** :

$$K = B^a \mod q = 26^{13} \mod 29 = 10$$



Étape 4 : Calcul de la clé partagée par Bob

- **Bob** utilise la **clé publique** reçue d'Alice ($A=19$) et son **secret privé** ($b=15$) pour **calculer la clé partagée K** :

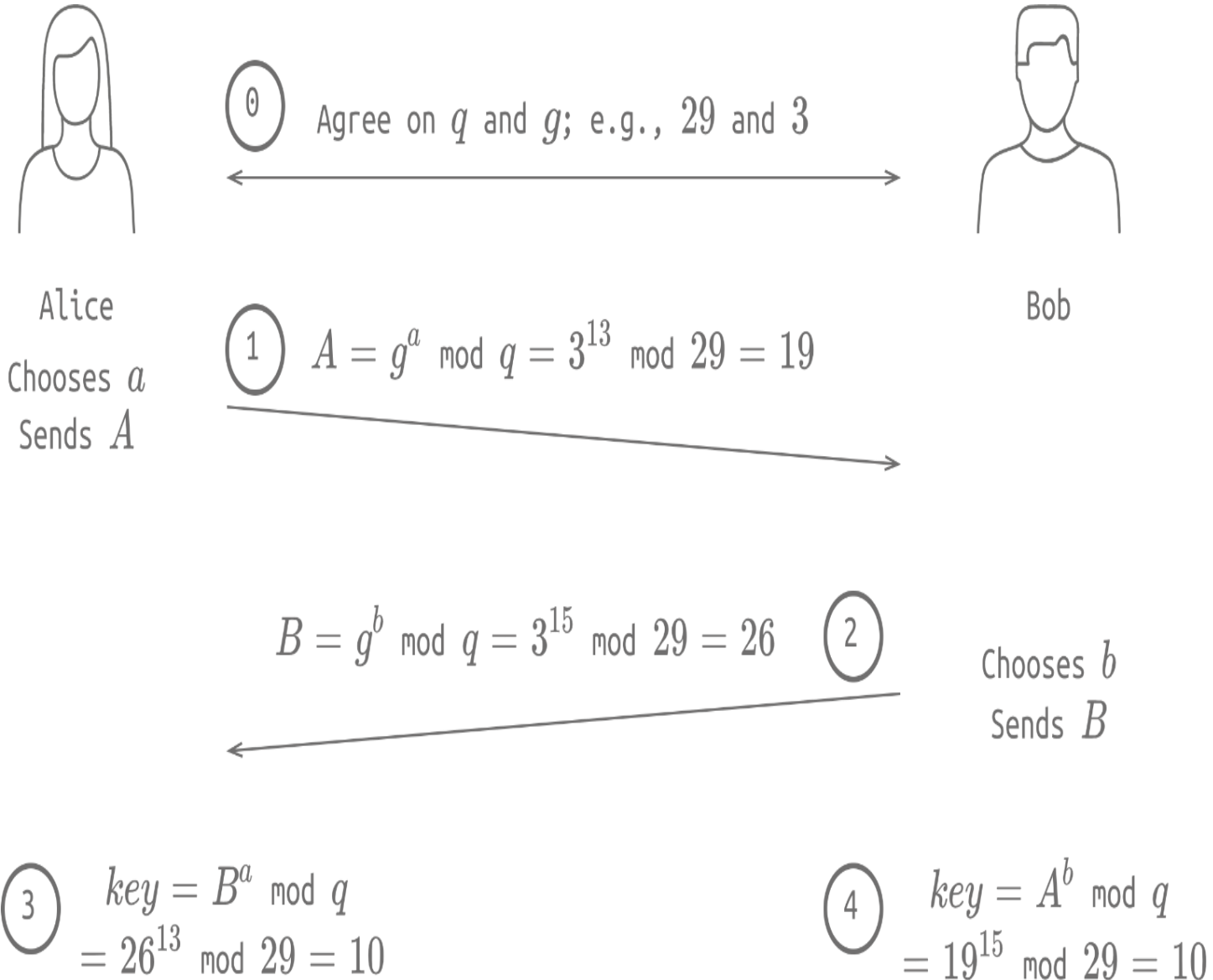
$$K = A^b \mod q = 19^{15} \mod 29 = 10$$

Résultat final : Clé partagée

- **Alice et Bob** obtiennent la **même clé partagée K=10**, qu'ils peuvent **utiliser pour chiffrer** leurs **communications avec un algorithme symétrique** (comme **AES**).

- **Cette clé n'a jamais été transmise** directement, ce qui **garantit sa confidentialité**.

[Voir Lab Diffie-Hellman](#)



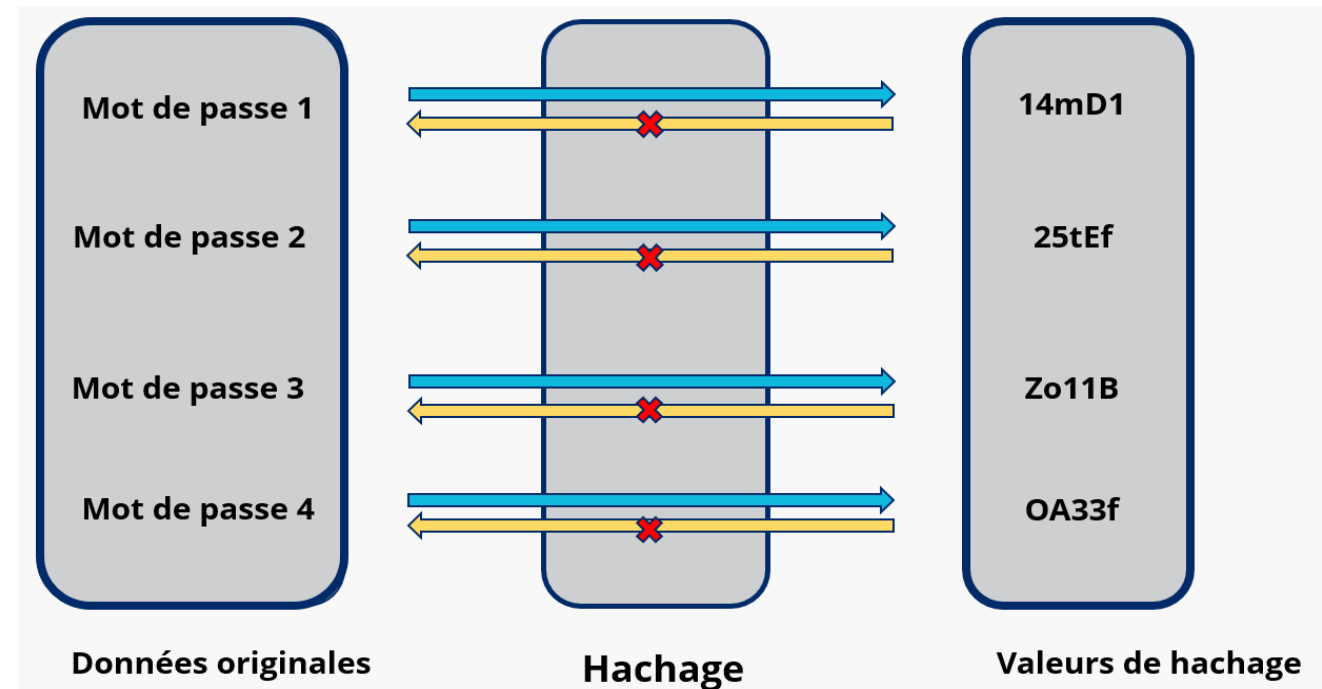
Hachage

Une **fonction de hachage cryptographique** est une **fonction mathématique** qui:

- **prend une donnée en entrée** (appelée "**message**") de **taille variable** et
- **génère une empreinte numérique** (appelée "**haché**") de **taille fixe**.



Cette **empreinte** est souvent représentée sous forme **hexadécimale**.



Propriétés principales d'une fonction de hachage :

1- Unidirectionnelle :

Impossible de retrouver l'entrée originale (le message) à partir de l'empreinte générée.

2- Déterministe :

Pour une même entrée, la fonction produira toujours la même empreinte.

3- Sensibilité aux changements :

Une petite modification dans le message d'entrée change complètement l'empreinte.

4- Rapidité :

Le calcul de l'empreinte est rapide, même pour de grands fichiers.

5- Résistance aux collisions :

Il est pratiquement impossible de trouver deux entrées différentes qui produisent la même empreinte.

Applications et Utilisation des fonctions de hachage en cryptographie :

- Vérification de l'intégrité des données (par exemple, pour détecter une modification non autorisée).
- Signatures numériques.
- Stockage sécurisé des mots de passe (hachage des mots de passe).
- Algorithmes de chiffrement et protocoles (comme TLS, SSL, blockchain, etc.).

Exemple de fonction de hachage courante :

SHA-256 (Secure Hash Algorithm - 256 bits) :

Une fonction de **hachage** largement utilisée dans:
la sécurité informatique,
les blockchains (Bitcoin),
les certificats numériques...

Certaines fonctions de hachage plus anciennes, telles que MD5 (Message Digest 5) et SHA-1, sont **cryptographiquement défectueuses**.

HMAC reste très pertinent pour des protocoles modernes où une clé secrète est utilisée.

Fonction	Taille du haché	Sécurité	Usage actuel
MD5	128 bits	Vulnérable aux collisions	Vérifications non sécurisées (intégrité).
SHA-1	160 bits	Vulnérable aux collisions	Obsolète, remplacé par SHA-256 ou mieux.
HMAC	Dépend de la fonction utilisée (ex. SHA-256)	Sûr (avec une clé secrète).	Protocoles sécurisés (TLS, IPsec, JWT).

[Voir Lab Introduction to Cryptography](#)
[Voir Lab SHA-256](#)

PKI et SSL/TLS

Problème: L'échange de clés via **Diffie-Hellman** nous permet de nous **mettre d'accord** sur une **clé secrète**, utilisée avec un **algorithme de chiffrement symétrique** pour **garantir la confidentialité des communications**.

Cependant, l'échange de clés que nous avons décrit précédemment peut être victime d'une **attaque de l'homme du milieu (MITM)**.

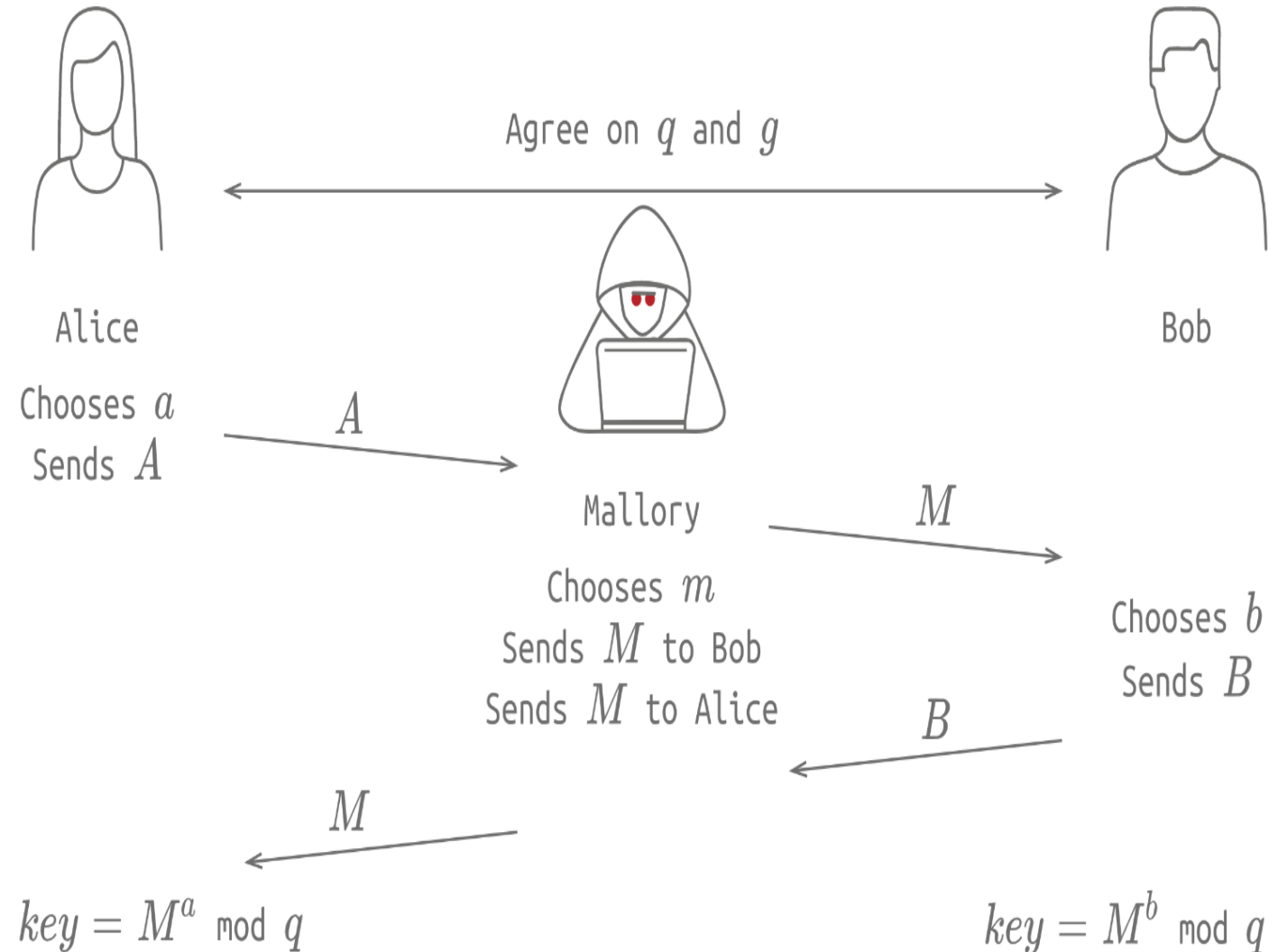
Absence d'authentification :

- **Alice et Bob** n'ont **aucun moyen** de **vérifier qu'ils échangent bien leurs clés** entre eux et **non avec un attaquant (Mallory)**.

- **Mallory intercepte les messages A** (d'Alice) et **B (de Bob)**.

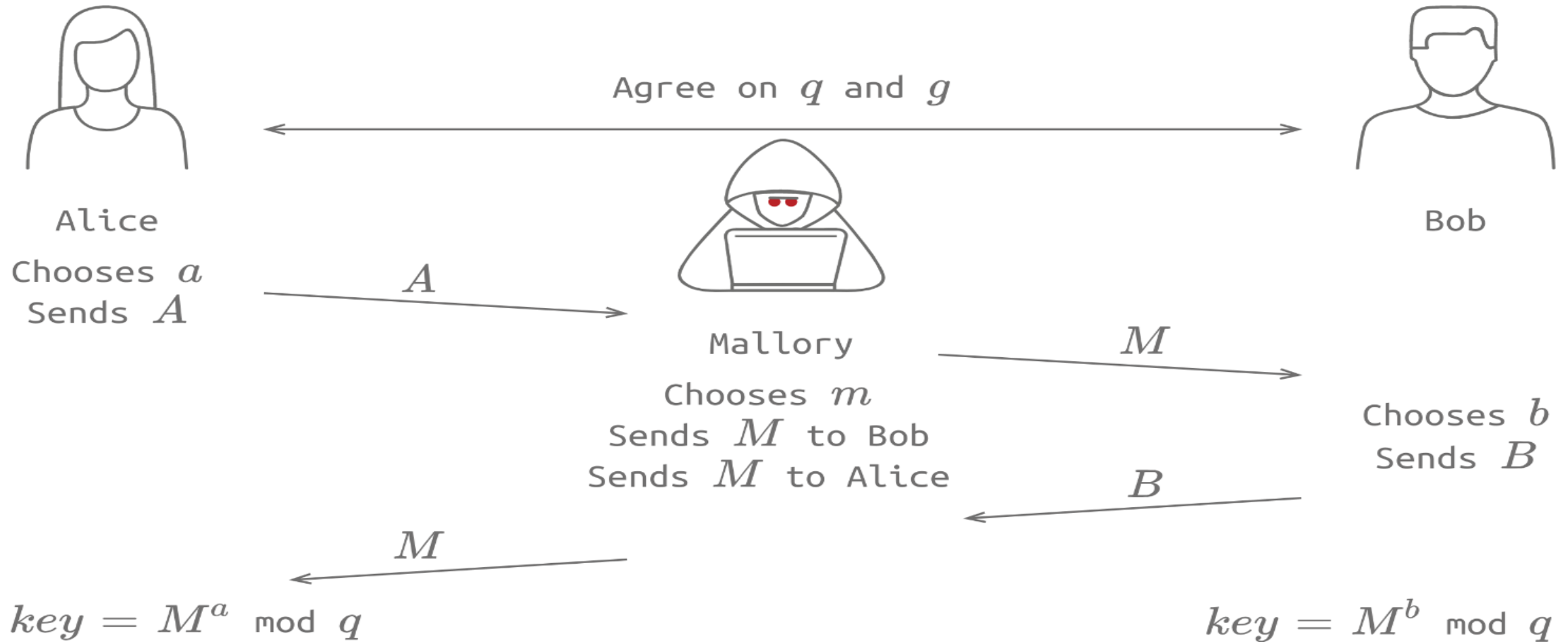
Elle substitue ses propres valeurs M à la place, créant deux clés distinctes :

- Une clé entre Alice et Mallory.
- Une clé entre Bob et Mallory.



Conséquences :

- Alice pense partager une clé avec Bob, mais **en réalité**, elle **communique avec Mallory**.
- **Mallory agit** comme un intermédiaire, lisant et modifiant les messages avant de les transmettre.
- C'est l'attaque dite **Man-in-the-Middle (MitM)**.



PKI (Public Key Infrastructure) : Authentification via certificats numériques

La PKI (Infrastructure à Clé Publique) est principalement réalisée dans le cadre de la **cryptographie asymétrique**.

Un certificat numérique est un **document électronique délivré** par une **autorité de certification (AC)**.

certificat associe la **clé publique** d'un utilisateur à **son identité (nom, organisation, etc.)** et est **signé numériquement** par l'**AC** pour **garantir son authenticité**.

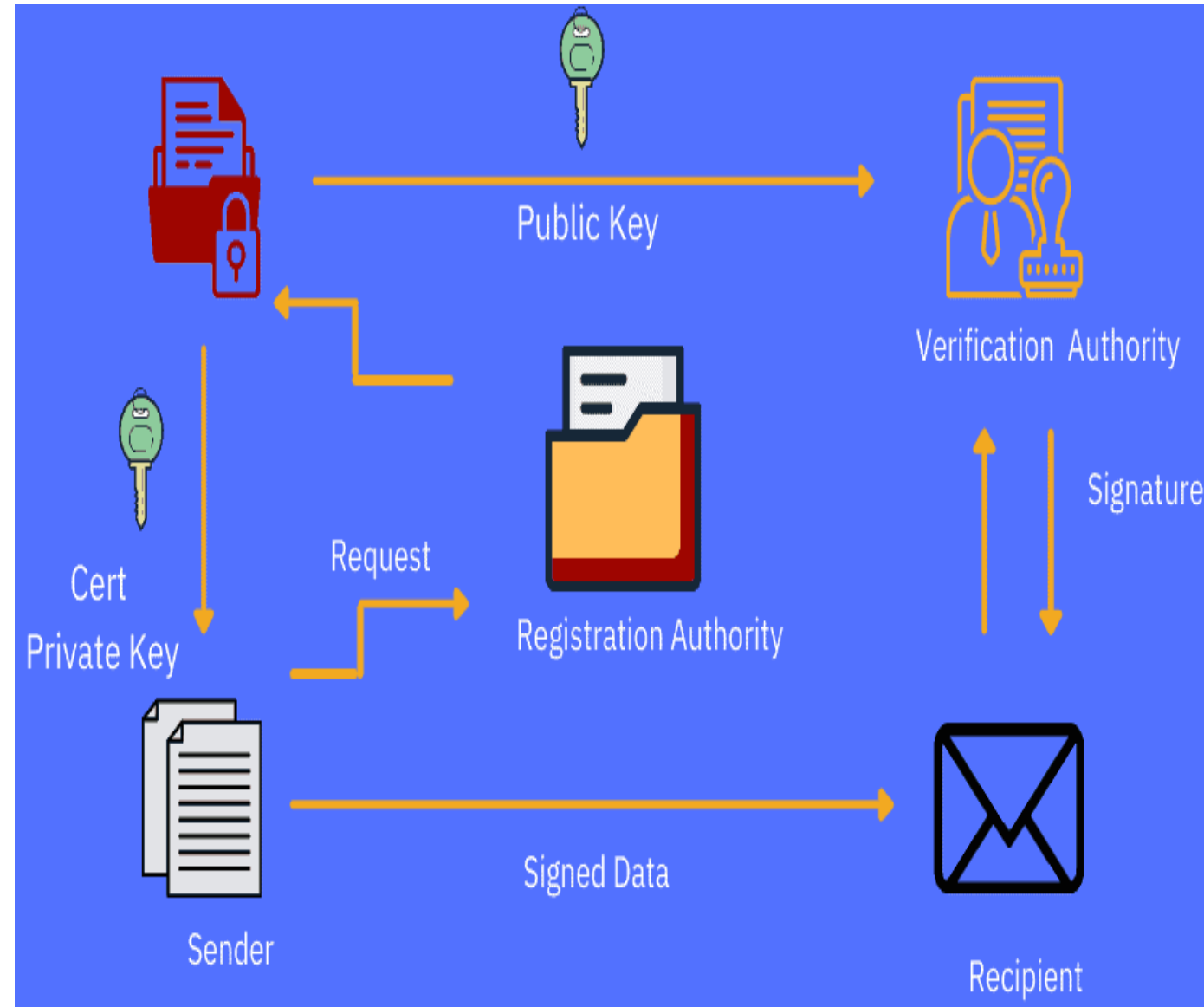
PKI (Public Key Infrastructure) : Authentification via certificats numériques

C'est un système permettant de gérer de manière sécurisée les clés cryptographiques et les certificats numériques.

Le PKI utilise des **certificats numériques** pour garantir l'identité des parties qui communiquent.

- Les **certificats numériques** sont émis par une **autorité de certification (CA)** de confiance.

- Chaque certificat contient la clé publique de son propriétaire et prouve que cette clé appartient bien à l'entité (par exemple, Alice ou Bob).



La PKI est une infrastructure qui repose sur l'utilisation de **clés publiques et privées** et de **certificats numériques** pour garantir :

- **L'authentification** : Vérifier l'identité des parties qui communiquent.
- **La confidentialité** : Assurer que les données échangées ne peuvent être lues que par le destinataire.
- **L'intégrité** : Garantir que les données n'ont pas été modifiées.
- **La non-répudiation** : S'assurer que l'expéditeur ne peut pas nier avoir envoyé le message.

Elle repose sur plusieurs composants clés, comme l'**autorité de certification (CA)**, l'**autorité d'enregistrement (RA)** et les certificats numériques.

Fonctionnement de la PKI :

- **Création de la paire de clés :**

L'utilisateur génère sa paire de clés.

- **Demande de certificat :**

L'utilisateur soumet sa clé publique à une AC avec une preuve d'identité.

- **Délivrance du certificat :**

L'AC vérifie l'identité et délivre un certificat numérique qui lie la clé publique à l'utilisateur.

- **Utilisation :**

Lors de communications sécurisées, le certificat permet à une autre partie de vérifier que la clé publique appartient bien à la personne ou l'entité déclarée.

Applications de la PKI

- **SSL/TLS** : Sécuriser les communications sur Internet (HTTPS).
- **Email sécurisé** : Utilisation de S/MIME pour signer ou chiffrer les emails.
- **VPN** : Authentification des utilisateurs et sécurisation des connexions.
- **Signature électronique** : Garantir l'intégrité des documents.
- **Authentification des utilisateurs** : Utilisation de certificats pour remplacer les mots de passe.

SSL/TLS : Authentification et échange sécurisé de clés

Le protocole SSL/TLS combine PKI avec Diffie-Hellman pour garantir à la fois :
l'authentification et la sécurité des échanges.

PKI fournit l'infrastructure nécessaire pour gérer les **certificats numériques** qui authentifient les parties dans une communication.

SSL/TLS utilise ces certificats numériques (générés grâce à PKI) pour établir des connexions sécurisées en garantissant l'authentification, la confidentialité et l'intégrité des données.

Exemple pratique : HTTPS (basé sur SSL/TLS avec PKI)

1. Un site web utilise un certificat SSL/TLS signé par une **CA** (Autorité de Certification).
2. Le navigateur du client vérifie ce certificat grâce à la PKI.
3. Une fois le certificat validé, SSL/TLS établit une connexion sécurisée.

[Voir Lab Cryptography](#)

Les attaques de l'homme du milieu (Man-in-the-Middle)

L'**attaque Man-In-The-Middle (MITM)** est une attaque dans laquelle un **attaquant écoute et modifie** éventuellement la **communication entre deux hôtes** sur le réseau et vole des données sensibles pour effectuer d'autres attaques.

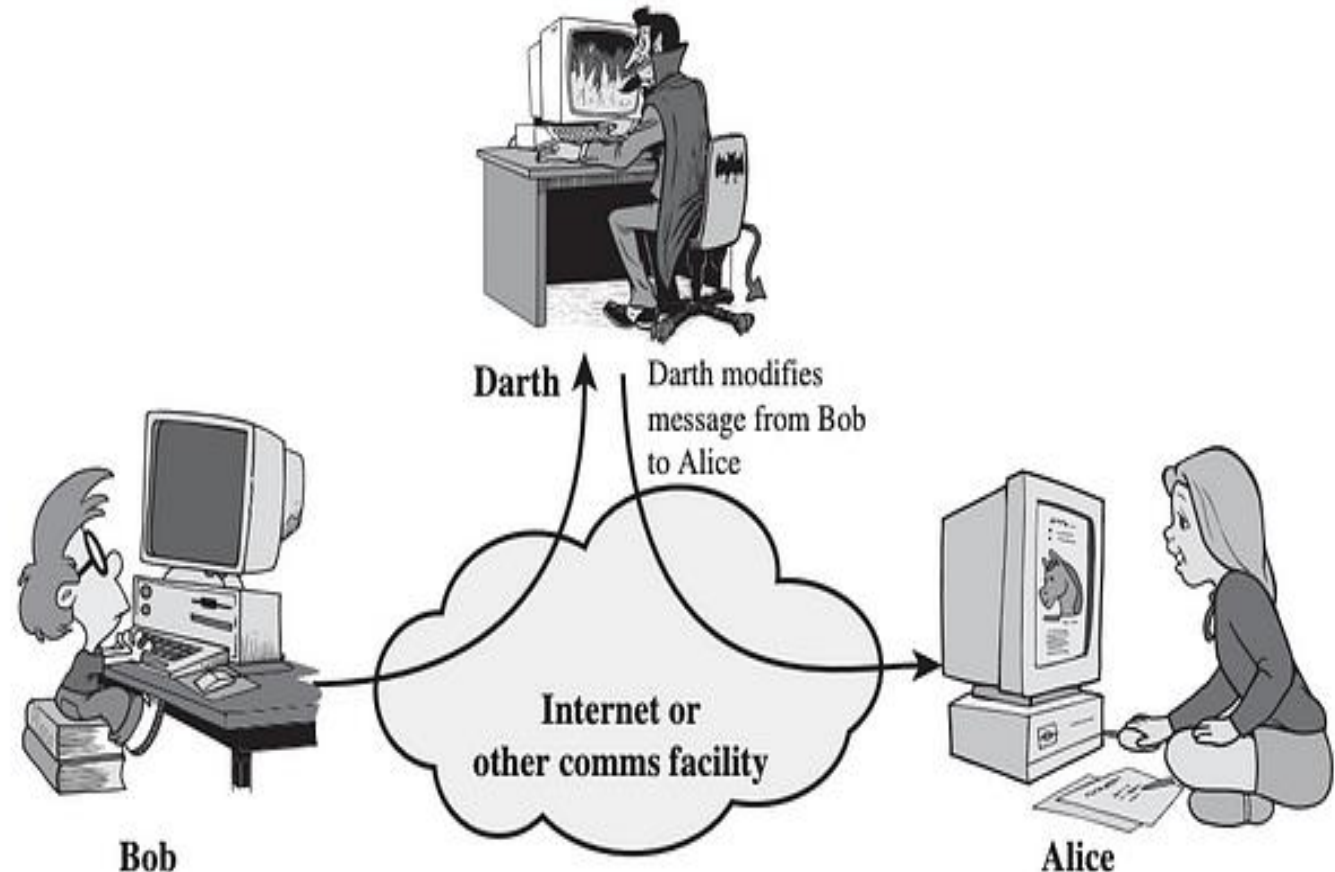
Imaginez Alice et Bob en pleine conversation.

Ils pensent que leur communication est sécurisée,

mais un adversaire nommé Darth intercepte et manipule astucieusement leurs messages à leur insu.

C'est l'essence même d'une attaque de l'homme du milieu (MITM).

L'attaquant se positionne entre Alice et Bob, interceptant et potentiellement altérant les messages échangés.



Étapes de l'Attaque MITM Interception de la clé publique de Bob :

- Lorsqu'Alice demande la clé publique de Bob (ex: via un serveur non sécurisé, un email, un site web), le pirate intercepte la requête.
- Il remplace la **vraie clé publique de Bob** par **sa propre clé publique** (fausse clé).
- Alice reçoit la **fausse clé publique** et croit qu'elle appartient à Bob.

1.Chiffrement par Alice :

- Alice chiffre son message avec la **fausse clé publique** (celle du pirate).
- Elle envoie le message chiffré à Bob.

étape	Alice	Pirate	Bob
1	Demande la clé publique de Bob →	Intercepte, envoie sa fausse clé.	
2	Chiffre avec la fausse clé →	Intercepte le message.	
3		Déchiffre avec sa clé privée.	
4		Rechiffre avec la vraie clé de Bob →	Reçoit le message et le déchiffre.

3. Interception du message chiffré :

- Le pirate intercepte le message **avant qu’il n’atteigne Bob**.
- Comme le message est chiffré avec **sa clé publique**, il le déchiffre avec **sa clé privée** (qu’il contrôle).
- Il peut alors **lire/modifier** le message.

4. Rechiffrement vers Bob :

- Le pirate rechiffre le message (modifié ou non) avec la **vraie clé publique de Bob**.
- Il transfère le message à Bob, qui le déchiffre normalement avec **sa clé privée**.

étape	Alice	Pirate	Bob
1	Demande la clé publique de Bob →	Intercepte, envoie sa fausse clé.	
2	Chiffre avec la fausse clé →	Intercepte le message.	
3		Déchiffre avec sa clé privée.	
4		Rechiffre avec la vraie clé de Bob →	Reçoit le message et le déchiffre.

Solution pour éviter les attaques de l'homme du milieu

Processus a suivre

Action	Objectif	Clé Utilisée	Exemple
Chiffrement	Rendre le message confidentiel	Clé publique du destinataire	Alice chiffre un message pour Bob.
Signature Numérique	Garantir l'origine/intégrité	Clé privée de l'émetteur	Alice signe un contrat pour Bob.

A. Chiffrement (Confidentialité)

Émetteur (Alice) :

- Alice récupère la **clé publique de Bob** (disponible librement).
- Elle **chiffre** le message avec cette clé publique.
- Elle envoie le message chiffré à Bob.

Destinataire (Bob) :

Bob **déchiffre** le message avec sa **clé privée** (secrète).

B. Signature Numérique (Authenticité/Intégrité)

Émetteur (Alice) :

- Alice génère un **hash** du message (ex: SHA-256).
- Elle **signe** ce hash avec sa **clé privée** → **Crée une signature**.
- Elle envoie le message **original + signature** à Bob.

Destinataire (Bob) :

- Bob génère le hash du message reçu.
- Il **vérifie** la signature avec la **clé publique d'Alice** (garantit l'origine et l'intégrité).

Action	Objectif	Clé Utilisée	Exemple
Chiffrement	Rendre le message confidentiel	Clé publique du destinataire	Alice chiffre un message pour Bob.
Signature Numérique	Garantir l'origine/intégrité	Clé privée de l'émetteur	Alice signe un contrat pour Bob.

Résultat

Le **décryptage réussi** à l'aide de la clé publique de Bob conduit à quelques conclusions intéressantes:

1- Premièrement (**validation d'intégrité**)

le message n'a pas été altéré tout au long du canal de communication ;
cela prouve **l'intégrité du message**.

Exemple:

Alice Prépare le Message

Message : "Payer 100€ à Bob".

Hachage (SHA-256) :

Hash = a1b2c3d4... (empreinte unique du message).

Signature : Alice chiffre ce hash avec sa clé privée → Signature SIG123

Deuxièmement (Validation d'authenticité du message),

Sachant que **personne n'a accès à la clé privée de Bob**,
nous **pouvons être sûrs** que ce message **vient bien de Bob** ; cela prouve **l'authenticité du message**.

Enfin (Non-répudiation),

comme **personne d'autre que Bob n'a accès à la clé privée de Bob**,
Bob ne peut pas nier l'envoi de ce message ;
cela établit la **non-répudiation**.

Authentification avec des mots de passe

Comment la cryptographie peut contribuer à renforcer la sécurité des mots de passe?

Grâce à PKI et SSL/TLS, nous pouvons communiquer avec n'importe quel serveur et fournir nos identifiants de connexion tout en garantissant que personne ne peut lire nos mots de passe lorsqu'ils circulent sur le réseau.

Il s'agit d'un exemple de protection des données en transit.

Comment nous pouvons protéger les mots de passe lorsqu'ils sont enregistrés dans une base de données?



Approche 1:

La méthode la moins sûre serait d'enregistrer le nom d'utilisateur et le mot de passe dans une base de données. toute violation de données exposerait les mots de passe des utilisateurs.

Aucun effort n'est requis au-delà de la lecture de la base de données contenant les mots de passe.

Nom d'utilisateur	mot de passe
Alice	qwerty
bob	dragon
Charlie	princesse

Approche 2

L'approche la plus efficace consisterait à **enregistrer le nom d'utilisateur** et une **version hachée** du **mot de passe** dans une base de données.

une violation de données exposerait les **versions hachées** des **mots de passe**.

Comme une **fonction de hachage** est **irréversible**,
l'attaquant doit continuer à essayer différents mots de passe pour trouver celui qui produirait le même hachage.

Le tableau ci-dessous montre la somme MD5 des mots de passe.
(Nous avons choisi MD5 simplement pour garder le champ du mot de passe petit pour l'exemple ; sinon, nous aurions utilisé SHA256 ou quelque chose de plus sécurisé.)

Problème : la disponibilité des **tables arc-en-ciel** a rendu cette approche peu sûre. Une **table arc-en-ciel** est une base de données pré-calculée par les pirates qui associe des mots de passe courants à leurs hachages.

**L'attaquant n'a pas besoin de deviner
ou de recalculer quoi que ce soit :
tout est pré-calculé.**

Nom d'utilisateur	Hachage (<u>mot de passe</u>)
Alice	d8578edf8458ce06fbc5bb76a58c5ca4
bob	8621ffdbc5698829397d97767ac13db3
Charlie	8afa847f50a716e64932d995c8e7435a

Approche 3

Utiliser un "salt" :

Un **salt** est une chaîne aléatoire ajoutée au mot de passe avant de le hacher.

Exemple avec un salt unique pour Alice :

- **Mot de passe** : qwerty
- **Salt** : randomSalt123
- **Entrée hachée** : hash(qwerty + randomSalt123)

Même si deux utilisateurs ont le même mot de passe (qwerty), leurs hachages seront différents car leurs salts seront différents.

Nom d'utilisateur	Hachage (<u>mot</u> de passe + sel)	Sel
Alice	8a43db01d06107fcad32f0bcfa651f2f	12742
bob	aab2b680e6a1cb43c79180b3d1a38beb	22861
Charlie	3a40d108a068cdc8e7951b82d312129b	16056