Architecture et OS	Chap 03	**Sécurisation des communications**

3.1 Pourquoi sécuriser les communications ?

3.1.1 HTTP : un protocole en clair

Le voyage d'un paquet

Lorsqu'on envoie des données sur Internet, ces données ne voyagent pas directement de notre machine au serveur. Elles sont découpées en **paquets**, qui transitent de routeur en routeur jusqu'à destination. Chaque routeur intermédiaire **reçoit le paquet, le lit, puis le retransmet**.

Ce mécanisme, que nous avons étudié dans le chapitre précédent, est très efficace pour acheminer l'information. Mais il soulève une question :

Que se passe-t-il si l'un de ces routeurs intermédiaires est malveillant ?

HTTP : tout passe en clair

Le protocole **HTTP** est le protocole de base du Web. Lorsqu'un navigateur demande une page à un serveur, les données échangées sont envoyées **sans aucune protection** : la requête, la réponse, les **identifiants** et les **mots de passe** saisis dans les formulaires sont tous visibles en clair.

Un outil comme **Wireshark** permet de capturer le trafic réseau qui transite sur une interface et c'est exactement ce que peut faire n'importe qui ayant accès à un routeur intermédiaire ou à un réseau local partagé comme un Wi-Fi public : **lire, voire modifier** les données qui transitent.

Exemple 1 — Ce qu'on peut lire dans une capture HTTP

Sur un réseau non sécurisé, une capture Wireshark d'une connexion HTTP peut révéler des lignes de ce type dans la requête envoyée au serveur :

```
POST /login HTTP/1.1
Host: exemple.com
...
username=aragorn&password=sixseven
```

Wi-Fi public

Se connecter à un Wi-Fi public (café, gare, hôtel…) sans précaution est dangereux. Toute personne connectée au même réseau peut intercepter le trafic non chiffré.

3.1.2 Les trois objectifs de la sécurité

Qu'attend-on d'une communication sécurisée ?

Reprenons la situation précédente. Aragorn envoie son mot de passe à un serveur via HTTP. Un attaquant intercepte le paquet : d'une part il peut le **lire**, de plus rien ne l'empêche de le **modifier** avant de le retransmettre. Enfin, Aragorn ne saura jamais si le serveur qui lui répond **est bien le bon**.

Cette situation met en lumière trois problématiques bien distinctes définies ci-après.

Définition 1 — Confidentialité

La confidentialité. Seuls l'émetteur et le destinataire peuvent lire le message. Un tiers qui intercepte les données ne doit pas pouvoir en comprendre le contenu.

Définition 2 — Intégrité

L'intégrité. Le message reçu est bien celui qui a été envoyé. Personne n'a pu le modifier en transit sans que cela soit détectable.

Définition 3 — Authenticité

L'authenticité. On est certain de l'identité de son interlocuteur. Le serveur avec lequel on communique est bien celui qu'il prétend être.

Exercice 1 — Identifier les objectifs

Pour chaque situation ci-dessous, indiquer quel(s) objectif(s) parmi **confidentialité**, **intégrité** et **authenticité** sont en jeu.

1. Aragorn envoie son numéro de carte bancaire à un site marchand.

   ---

2. Aragorn reçoit un e-mail envoyé par sa banque.

   ---

3. Bilbo reçoit un virement et veut s'assurer que le montant est bien le bon.

   ---

4. Une mise à jour logicielle est téléchargée sur Internet.

3.2 Principes du chiffrement

Rendre le message illisible

Face aux risques que présente HTTP, la solution naturelle est de **chiffrer** les données avant de les envoyer : on les transforme de sorte qu'elles deviennent incompréhensibles pour quiconque intercepterait les paquets en transit.

Remarque 1. L'idée n'est pas nouvelle. Les Spartiates utilisaient déjà au V^{e} siècle av. J.-C. un dispositif appelé **scytale** : un bâton autour duquel on enroulait une lanière de cuir pour écrire un message. Une fois déroulée, la lanière ne montrait qu'une suite de lettres sans signification ; seul le destinataire possédant un bâton de même diamètre pouvait la lire[1].

Scytale avec message enroulé

Lanière vierge pour pourvoir chiffrer

Définition 4 — Message clair

Le message clair est le message original, lisible par quiconque en prend connaissance.

Définition 5 — Chiffrer

Chiffrer consiste à transformer un message clair en message chiffré à l'aide d'une clé.

Définition 6 — Clé

Une **clé** est une information secrète utilisée par l'algorithme de chiffrement pour transformer le message clair en message chiffré, et inversement.

Définition 7 — Message chiffré

Le message chiffré est le message transformé par un algorithme de chiffrement. Il est incompréhensible sans la clé correspondante.

Définition 8 — Déchiffrer

Déchiffrer consiste à retrouver le message clair à partir du message chiffré, en connaissant la clé.

1. https://fr.wikipedia.org/wiki/Scytale

Définition 9 — Attaquant

Un **attaquant** est toute personne cherchant à accéder à des informations qu'elle n'est pas censée connaître, ou à perturber une communication.

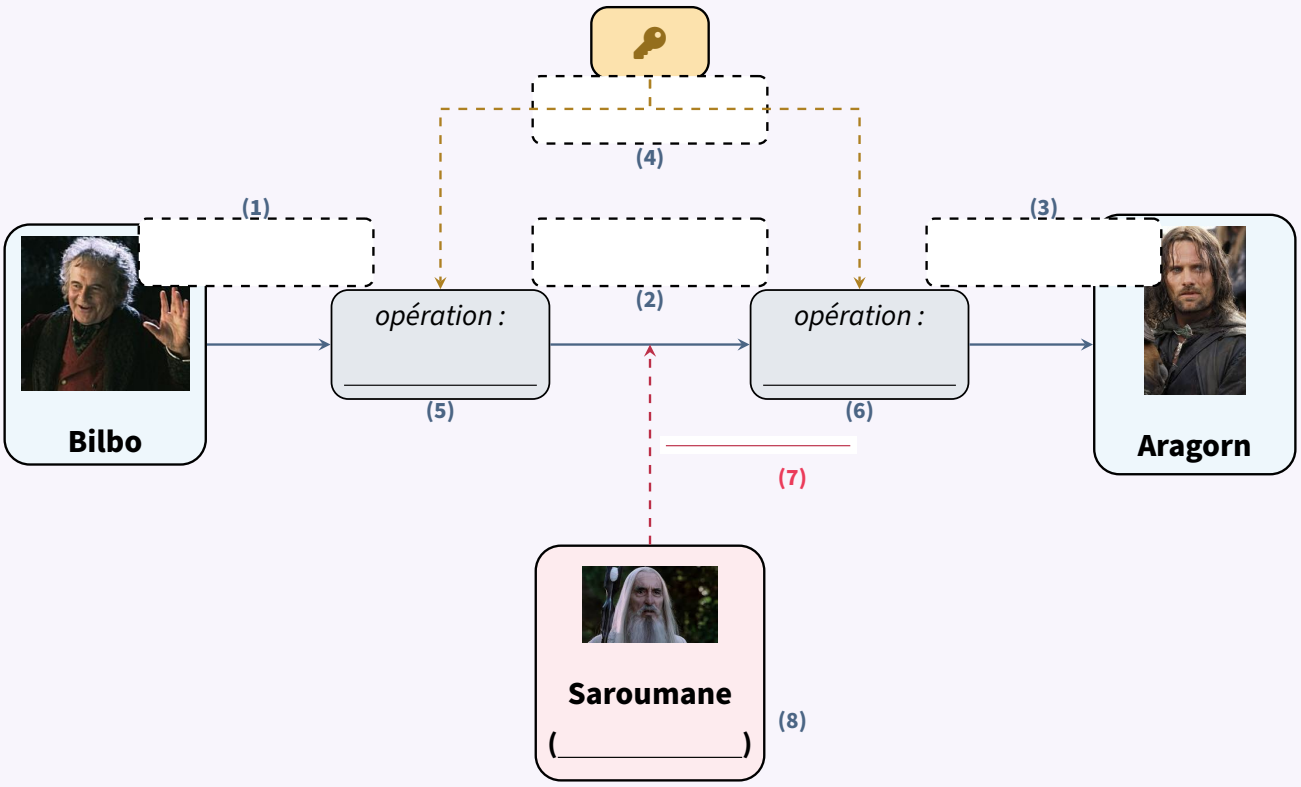
Définition 10 — Décrypter

Décrypter consiste à retrouver le message clair sans connaître la clé. C'est ce que tente de faire un attaquant.

Remarque 2. On confond souvent « déchiffrer » et « décrypter » dans le langage courant. En cryptographie, la distinction est importante : déchiffrer est une opération légitime (on possède la clé), décrypter est une attaque.

Exercice 2 — Récapitulatif vocabulaire

Compléter le schéma suivant, dans lequel Bilbo veut envoyer un message à Aragorn, avec les mots : *message clair* (×2), *message chiffré*, *clé*, *chiffrer*, *déchiffrer*, *décrypter* et *attaquant*

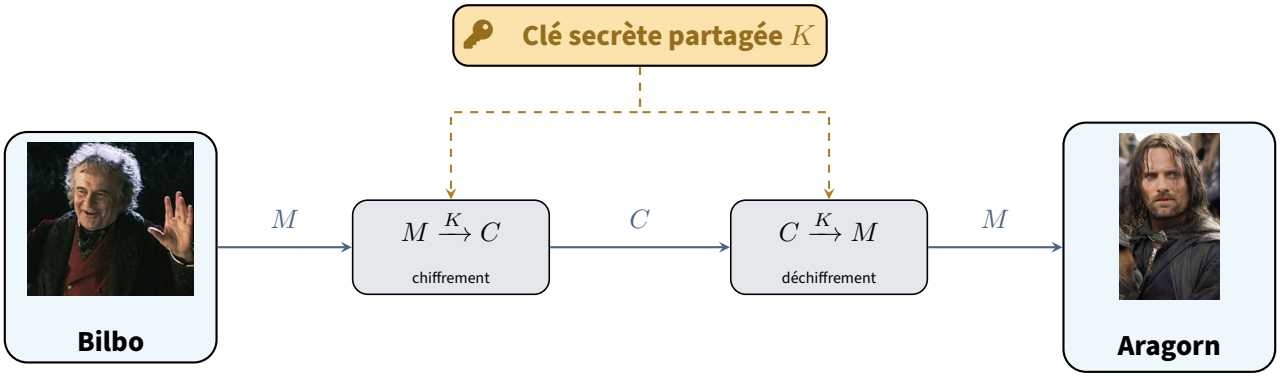


3.3 Le chiffrement symétrique

3.3.1 Principe et exemples

Définition 11 — Chiffrement symétrique

Dans un **chiffrement symétrique**, la même clé secrète est utilisée pour chiffrer et pour déchiffrer le message. L'émetteur et le destinataire doivent donc partager cette clé **avant** toute communication.



Le chiffrement de César

Le chiffrement de César est l'un des plus anciens exemples de chiffrement symétrique. Le principe est de décaler chaque lettre du message clair d'un nombre fixé dans l'alphabet, ce nombre constituant la **clé**.

Pour un décalage de 10 : A devient K, B devient L, S devient C, etc. Pour déchiffrer, il suffit de décaler dans l'autre sens du même nombre.

Exemple 2 — Chiffrement de César avec une clé de 10

On souhaite chiffrer le message `NSI` avec une clé de 10. Les positions sont numérotées de 0 (A) à 25 (Z), et le résultat est pris modulo 26 pour revenir au début de l'alphabet si nécessaire.

Lettre claire	Position	Lettre chiffrée
N	13	X $(13 + 10 = 23)$
S	18	C $(18 + 10 = 28 \equiv 2[26])$
I	8	S $(8 + 10 = 18)$

Le message chiffré est donc `XCS`.

Pour déchiffrer, on décale en sens inverse : X (23) $\to$ N $(23 - 10 = 13)$, C (2) $\to$ S $(2 - 10 + 26 = 18)$, S (18) $\to$ I $(18 - 10 = 8)$.

À coder 1 — Le chiffrement de César

On rappelle les fonctions utiles : `ord(c)` renvoie le code ASCII du caractère `c`, `chr(n)` renvoie le caractère de code ASCII `n`, et l'opérateur `%` donne le reste de la division euclidienne.

1. Écrire une fonction `cesar(message, cle)` qui prend en paramètres une chaîne de caractères en majuscules et un entier, et qui renvoie le message chiffré par la méthode de César. On ne chiffre ni les espaces ni la ponctuation.
2. Vérifier que `cesar("NSI", 10)` renvoie bien `"XCS"`.
3. Écrire une fonction `dechiffrer_cesar(message, cle)` qui déchiffre un message chiffré par César en réutilisant la fonction précédente. Appliquer votre fonction au message `"KVVOJ MYEBKQO NKXC MSXAEKXDO AEKDBO TYEBC MOCD VOC FKMKXMOC"` avec la clé `10`.

Chiffrer bit à bit avec XOR

L'opération **XOR** (ou exclusif, notée $\oplus$) s'applique sur deux bits. Deux bits identiques donnent 0, deux bits différents donnent 1.

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

On peut utiliser le XOR pour chiffrer un message M avec une clé K : on applique le XOR bit à bit entre chaque bit du message et la clé, ce qui donne le message chiffré $C = M \oplus K$.

Pour déchiffrer, il suffit d'appliquer **à nouveau** le XOR avec la même clé :

$$C \oplus K = (M \oplus K) \oplus K = M.$$

Chiffrer et déchiffrer sont donc **la même opération**, ce qui rend le XOR particulièrement élégant comme primitive de chiffrement.

Exemple 3 — Chiffrement XOR

On chiffre le message `M`, contenant la lettre "N" (code ASCII 78 = `01001110`) avec la clé $K = 42$ (`00101010`).

$$\begin{aligned} M(messageN) &= 01001110 \\ K &= 00101010 \\ \hline M \oplus K &= 01100100 \end{aligned}$$

Le message chiffré vaut `01100100` = 100. Pour déchiffrer, on applique à nouveau XOR avec la même clé : $100 \oplus 42 = 78 =$ `N`. On retrouve bien le message original.

</> À coder 2 — Le chiffrement XOR

On rappelle les fonctions utiles : `ord(c)` renvoie le code ASCII du caractère `c`, `chr(n)` renvoie le caractère de code ASCII `n`, et l'opérateur XOR entre deux entiers s'écrit `a ^ b` en Python.

1. Écrire une fonction `xor(message, cle)` qui chiffre un message caractère par caractère en appliquant un XOR entre le code ASCII de chaque caractère et la clé (un entier).
2. Vérifier que `xor(xor("NSI", 67), 67)` renvoie bien `"NSI"`.
3. On a reçu le message chiffré suivant, obtenu par XOR avec la clé 42 : les codes ASCII des caractères chiffrés sont `[102, 101, 102]`. Déchiffrer ce message en appliquant à nouveau le XOR avec la clé 42. Quel est le message original ?

Exercice 3 — Bonus. Prouver la propriété fondamentale du XOR

On veut démontrer rigoureusement que $(a \oplus b) \oplus b = a$, c'est-à-dire que déchiffrer avec la même clé redonne bien le message original.

1. **Par table de vérité.** Compléter la table de vérité suivante pour les valeurs $a \in \{0, 1\}$ et $b \in \{0, 1\}$. Vérifier que la colonne $(a \oplus b) \oplus b$ est identique à la colonne a.

a	b	$a \oplus b$	$(a \oplus b) \oplus b$
0	0		
0	1		
1	0		
1	1		

2. **Par simplification algébrique.** On rappelle que le XOR s'exprime avec les opérateurs ET ($\wedge$) et OU ($\vee$) de la façon suivante :

$$a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$$

En utilisant les règles usuelles de l'algèbre booléenne (distributivité, complémentation, idempotence…), simplifier l'expression $(a \oplus b) \oplus b$ pour montrer qu'elle est égale à a.

3.3.2 Le principe de Kerckhoffs

La sécurité ne repose pas sur le secret de l'algorithme

On pourrait penser qu'un algorithme de chiffrement est d'autant plus sûr qu'il est gardé secret. Auguste Kerckhoffs a formulé en 1883 un principe qui contredit cette intuition et qui reste aujourd'hui au fondement de la cryptographie moderne.

Un algorithme utilisé à grande échelle finit toujours par être connu : il peut être découvert par rétro-ingénierie, divulgué accidentellement, ou simplement publié pour être analysé par la communauté scientifique. En revanche, une clé est courte et peut être changée facilement.

C'est pourquoi les algorithmes modernes comme **AES** sont entièrement publics : leur robustesse a été vérifiée par des milliers de chercheurs, et leur sécurité repose uniquement sur la clé.

Propriété 1 (Principe de Kerckhoffs)**.** La sécurité d'un système de chiffrement ne doit pas reposer sur le secret de l'algorithme, mais **uniquement sur le secret de la clé**.

Autrement dit, même si un attaquant connaît l'algorithme utilisé, il ne doit pas être capable de déchiffrer le message sans connaître la clé.

Exercice 4 — César et le principe de Kerckhoffs

1. Le chiffrement de César respecte-t-il le principe de Kerckhoffs ? Justifier en vous appuyant sur le nombre de clés possibles.

2. Le chiffrement XOR avec une clé d'un seul octet offre combien de clés possibles offre-t-il ? Cela vous semble-t-il respecter le principe de Kerckoffs ?

3. Un algorithme de chiffrement moderne utilise des clés de 128 bits. Combien de clés différentes cela représente-t-il ? Que peut-on en conclure ?

3.3.3 Avantages et limites

Un chiffrement rapide mais avec un problème central

Le chiffrement symétrique est **rapide** et adapté au chiffrement de grands volumes de données. Les exemples vus jusqu'ici (César, XOR) sont sympathiques mais **fragiles**. En pratique, on utilise des algorithmes bien plus robustes, comme **AES** (*Advanced Encryption Standard*), standardisé en 2001 et utilisé aujourd'hui dans la quasi-totalité des communications sécurisées sur Internet. AES est entièrement public : sa sécurité repose uniquement sur la clé, conformément au principe de Kerckhoffs.

Cependant, le chiffrement symétrique souffre d'un problème fondamental que César illustre déjà : pour communiquer, Aragorn et Bilbo doivent se mettre d'accord sur la clé **avant** d'échanger quoi que ce soit. Sur Internet, deux machines qui ne se sont jamais vues doivent pourtant pouvoir établir une connexion sécurisée.

La question qui se pose est donc :

Comment échanger une clé secrète sur un réseau où n'importe qui peut intercepter les messages ?

C'est le **problème de l'échange de clé**, et c'est précisément ce que le chiffrement asymétrique va résoudre.

3.4 Le chiffrement asymétrique

3.4.1 Publique mais pas trop

Un cadenas ouvert

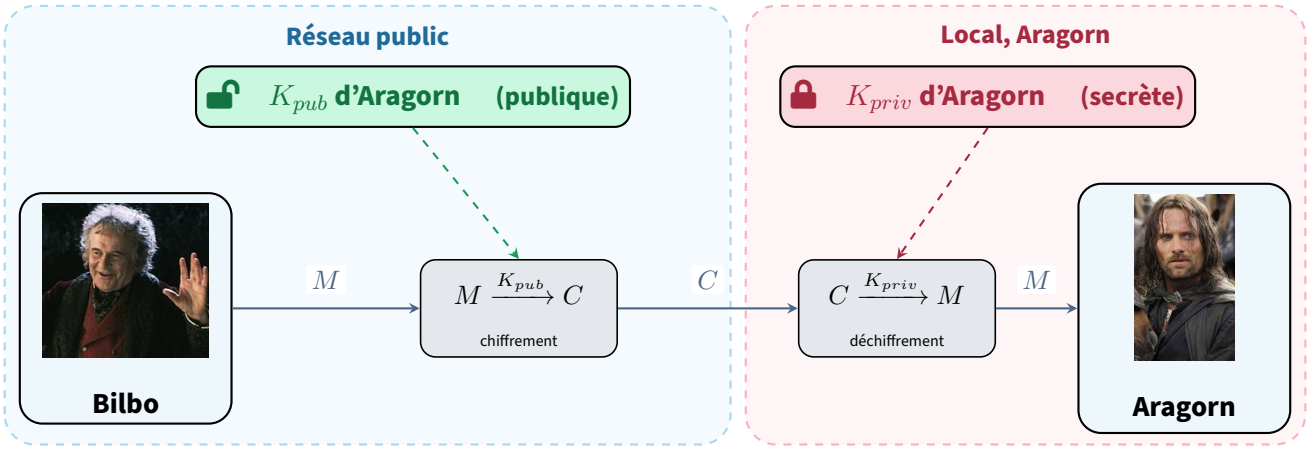
Imaginons qu'Aragorn veuille recevoir des messages secrets de n'importe qui, sans avoir à échanger de secret au préalable. Il fabrique un grand nombre de cadenas ouverts et les distribue librement : n'importe qui peut en prendre un. Mais lui seul possède la clé qui permet de les ouvrir.

Pour envoyer un message secret à Aragorn, Bilbo met son message dans une boîte, la ferme avec l'un de ces cadenas, et envoie la boîte. Personne, pas même Bilbo, ne peut plus l'ouvrir en chemin. Seul Aragorn, qui possède la clé, peut le faire.

Définition 12 — Chiffrement asymétrique

Dans un **chiffrement asymétrique**, chaque participant possède une paire de clés : une **clé publique**, librement distribuée, et une **clé privée**, gardée secrète.

La clé publique sert à **chiffrer** un message à destination de son propriétaire. **La clé privée** est la seule à pouvoir **déchiffrer** un message chiffré avec la clé publique correspondante.



Échange entre Aragorn et Bilbo

Aragorn souhaite recevoir des messages confidentiels de Bilbo, sans qu'ils aient jamais échangé de secret au préalable.

Étape 1 : Génération des clés. Aragorn génère une paire de clés : une clé publique K_{pub} et une clé privée K_{priv}. Il publie K_{pub} librement et garde K_{priv} secrète.

Étape 2 : Chiffrement. Bilbo récupère K_{pub} et chiffre son message M : il obtient le message chiffré $C = \text{chiffrer}(M,\ K_{pub})$.

Étape 3 : Transmission. Bilbo envoie C sur le réseau. Même si un attaquant intercepte C, il ne peut pas le déchiffrer sans K_{priv}.

Étape 4 : Déchiffrement. Aragorn déchiffre C avec sa clé privée : $M = \text{déchiffrer}(C,\ K_{priv})$.

À aucun moment une clé secrète n'a transité sur le réseau.

RSA : une idée de la construction

L'algorithme **RSA** est l'un des algorithmes asymétriques les plus connus. Sa sécurité repose sur la difficulté de **factoriser** de grands entiers : multiplier deux grands nombres premiers p et q est une opération rapide, mais l'opération inverse, retrouver p et q depuis leur produit $n = p \times q$, devient pratiquement impossible dès que p et q sont suffisamment grands. C'est cette asymétrie de difficulté qui protège le système : un attaquant qui ne connaît que n (la clé publique) ne peut pas retrouver p et q, et donc pas reconstruire la clé privée.

Aragorn génère ses clés. Il choisit deux nombres premiers $p = 5$ et $q = 11$, et calcule leur produit $n = p \times q = 55$. Il choisit ensuite un exposant public $e = 3$. Sa **clé publique** est la paire $(e,\, n) = (3,\, 55)$: il la publie librement. Sa **clé privée** est l'exposant d, calculé à partir de p, q et e selon une relation mathématique précise (voir ci-dessous) ; ici $d = 27$. Aragorn garde d secret.

Bilbo chiffre un message. Bilbo veut envoyer le message $M = 2$ à Aragorn. Il récupère la clé publique $(3,\, 55)$ et calcule le message chiffré C :

$$C \equiv M^e \,[n] \qquad \text{soit} \qquad C \equiv 2^3 \,[55] \qquad \text{soit} \qquad C = 8$$

Il envoie $C = 8$ sur le réseau. Même intercepté, ce message est inexploitable sans la clé privée d.

Aragorn déchiffre. Aragorn utilise sa clé privée $d = 27$ pour retrouver le message :

$$M \equiv C^d \,[n] \qquad \text{soit} \qquad M \equiv 8^{27} \,[55] \qquad \text{soit} \qquad M = 2$$

Il retrouve bien le message original.

En pratique, p et q font plusieurs centaines de chiffres. La clé publique n est alors un nombre colossal, et aucun algorithme connu ne permet de le factoriser en un temps raisonnable avec les technologies actuelles.

Pourquoi ça marche ?

RSA repose sur le **théorème d'Euler** : pour $n = p \times q$ avec p et q premiers, et pour tout message M premier avec n :

$$M^{\phi(n)} \equiv 1 \,[n]$$

où $\phi(n) = (p-1)(q-1)$ est l'**indicatrice d'Euler**.

La clé privée d n'est pas choisie au hasard : elle est construite comme l'**inverse de** e **modulo** $\phi(n)$, c'est-à-dire l'entier vérifiant :

$$e \times d \equiv 1 \,[\phi(n)]$$

On a alors :

$$C^d \equiv (M^e)^d \equiv M^{e \times d} \equiv M^{1 + k\,\phi(n)} \equiv M \cdot \left(M^{\phi(n)}\right)^k \equiv M \,[n]$$

On retrouve bien le message original.

Dans l'exemple ci-dessus : $\phi(55) = (5-1)(11-1) = 40$, et $3 \times 27 = 81 = 2 \times 40 + 1$, donc $3 \times 27 \equiv 1 \,[40]$.

3.4.2 HTTPS : les deux chiffrements réunis

Le meilleur des deux mondes

Lorsqu'on accède à un site en **HTTPS**, les requêtes HTTP ne disparaissent pas : elles sont simplement encapsulées dans un **tunnel chiffré** établi par le protocole **TLS** (*Transport Layer Security*).

On part du constat suivant :
Le chiffrement asymétrique est sûr mais lent.
Le chiffrement symétrique est rapide mais nécessite une clé partagée.
TLS combine les deux types de chiffrement :

— le **chiffrement asymétrique** (RSA) pour établir la connexion et s'assurer de l'identité du serveur ;

— le **chiffrement symétrique** (AES) pour chiffrer toutes les données échangées pendant la session.

Le protocole TLS en pratique

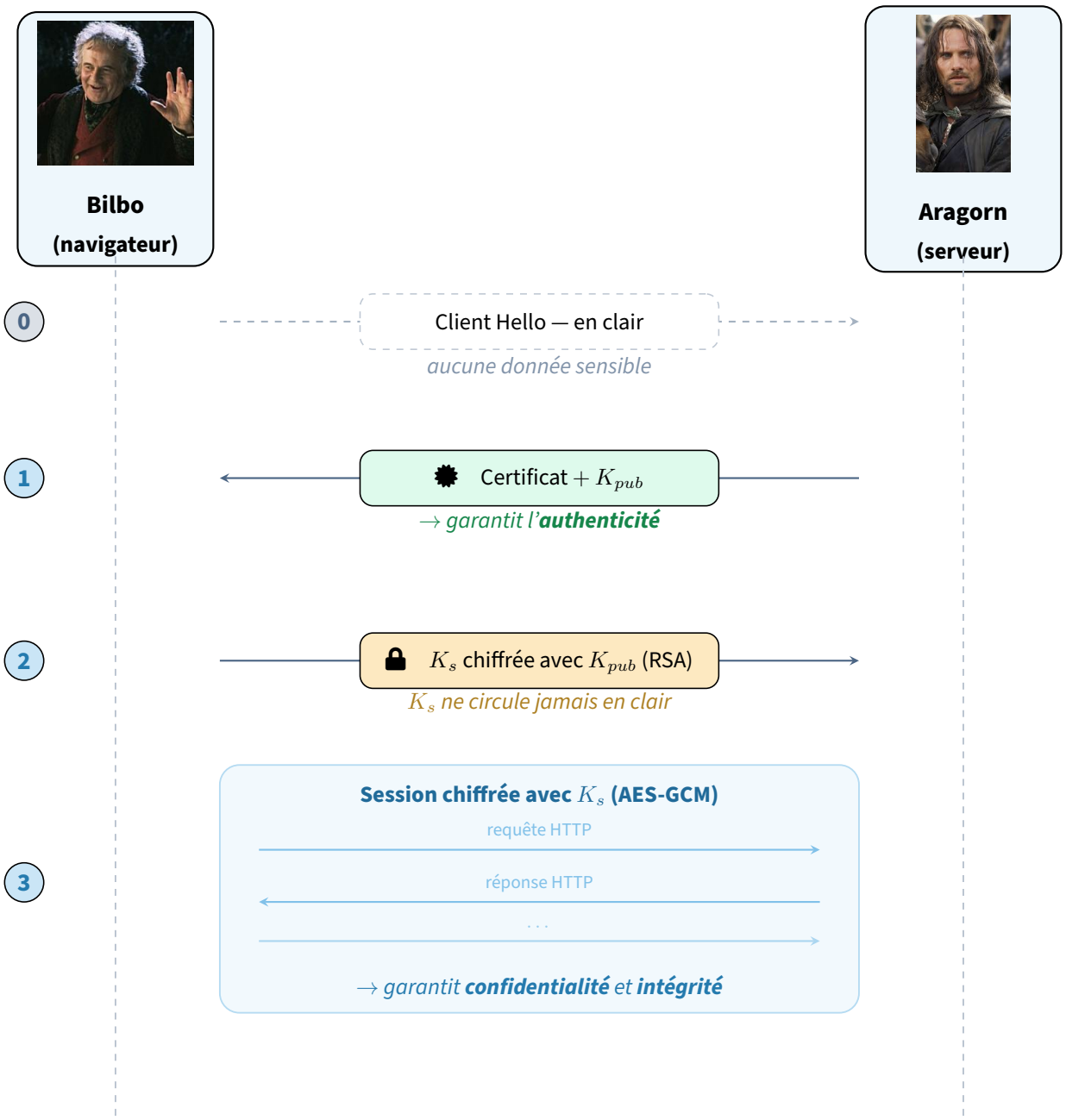
Lorsqu'un navigateur se connecte à un site en HTTPS, les trois étapes suivantes se déroulent automatiquement.

Étape 0. Initialisation de la connexion. Avant tout chiffrement, le navigateur envoie un message en clair au serveur pour initier la négociation TLS : c'est le **Client Hello**.

Étape 1. Authentification du serveur. Le serveur envoie un **certificat numérique** : un document électronique qui associe son identité à sa clé publique K_{pub}, et qui est **signé par une autorité de certification** (CA). La clé publique de cette autorité est préinstallée dans le navigateur, ce qui lui permet de vérifier la signature et de s'assurer qu'il parle bien au bon serveur et non à un imposteur. C'est ce mécanisme qui garantit l'**authenticité**.

Étape 2. Échange d'une clé symétrique (RSA). Le navigateur génère une clé symétrique K_s, la chiffre avec K_{pub} et l'envoie au serveur. Seul le serveur peut la déchiffrer avec sa clé privée K_{priv}. À l'issue de cette étape, les deux parties partagent K_s sans qu'elle n'ait jamais circulé en clair sur le réseau.

Étape 3. Communication chiffrée (AES). Tous les échanges suivants : requêtes HTTP, réponses, données, sont chiffrés avec K_s via AES. AES garantit la **confidentialité** des données. Il garantit également l'**intégrité** grâce à un mode spécial d'AES (AES-GCM), toute modification d'un paquet en transit est détectable, car elle invalide le chiffrement.



HTTPS dans le navigateur

La connexion HTTPS est signalée dans la barre d'adresse du navigateur par un cadenas. En cliquant dessus, on peut consulter les détails du certificat : le nom du propriétaire, l'autorité de certification qui l'a signé, et sa date de validité.

Les voyous aussi font du HTTPS

Le cadenas signifie que la **connexion** est chiffrée et authentifiée. Il ne garantit pas que le **contenu** du site est fiable : un site malveillant peut très bien posséder un certificat HTTPS valide.

3.5 Bilan, calmement..

Exercice 5 — Qui fait quoi ?

Compléter le tableau suivant avec des ✓.

Mécanisme	**Confidentialité**	**Intégrité**	**Authenticité**
Chiffrement symétrique			
Chiffrement asymétrique			
Certificat numérique			
HTTPS			

3.6 À l'attaque !!

3.6.1 L'attaque par force brute

Définition 13 — Attaque par force brute

Une **attaque par force brute** consiste à essayer **toutes les clés possibles** jusqu'à trouver celle qui permet de retrouver un message intelligible.

Sa complexité est directement liée à la taille de l'espace des clés : plus cet espace est grand, plus l'attaque est coûteuse en temps de calcul.

À coder 3 — Force brute sur César

On dispose du message chiffré `"KHOOR PRQGH"` obtenu par chiffrement de César.

1. Écrire une fonction `force_brute(message)` qui affiche les 26 déchiffrements possibles d'un message chiffré par César, en réutilisant la fonction `cesar` écrite précédemment.
2. Appliquer cette fonction au message `""ZIADI KZPO ZOMZ HJDIN NZGJI QJOMZ QDOZNNZ V AVDMZ XZ XJPMN""`. Quelle est la clé utilisée ? Quel est le message original ?
3. AES utilise des clés de 128 bits. Sachant qu'un ordinateur peut tester 10^{12} clés par seconde, estimer le temps nécessaire pour parcourir tout l'espace des clés. Le résultat vous surprend-il ?

3.6.2 Homme (de la Terre) du milieu !

S'intercaler dans la communication

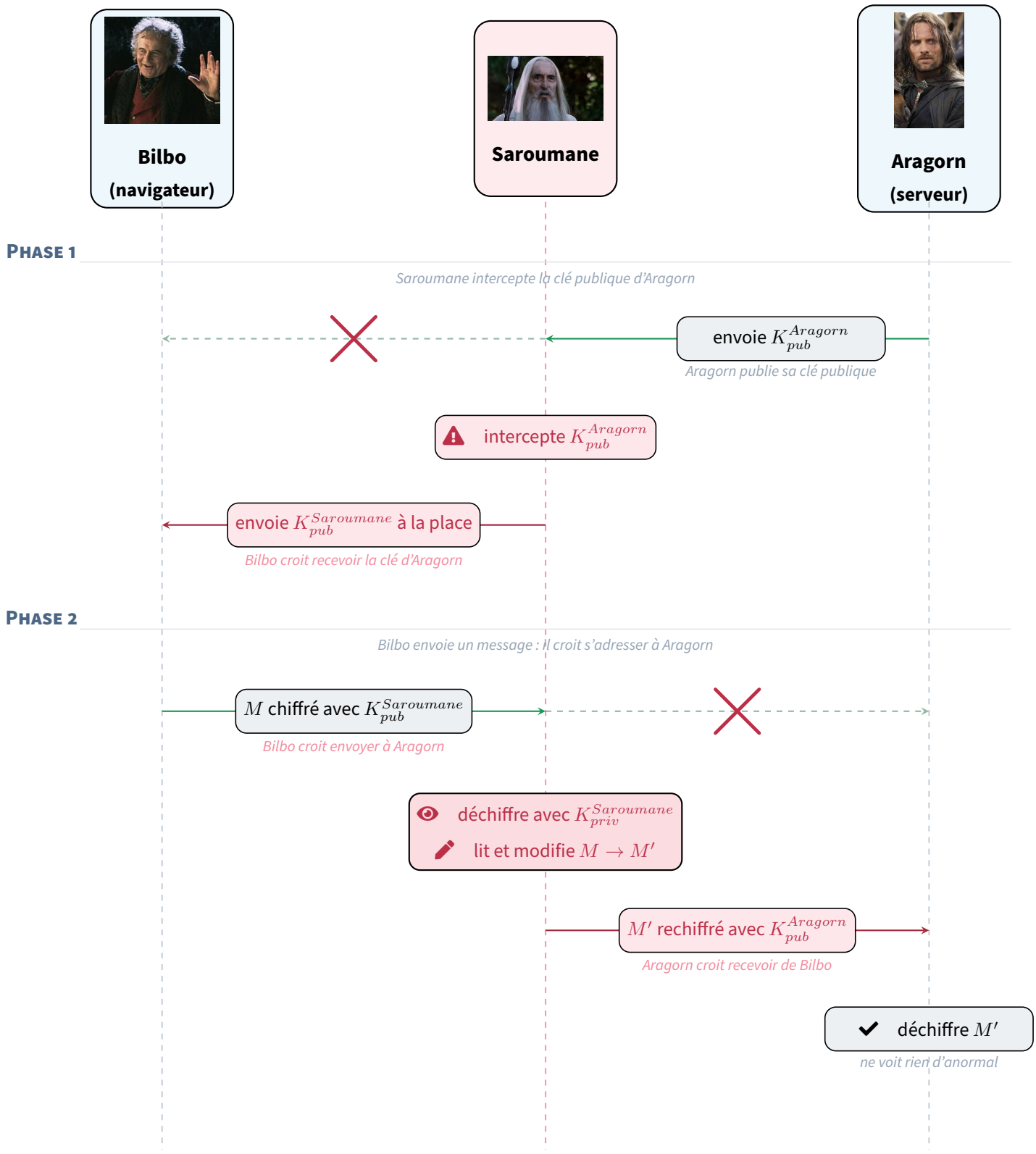
Le chiffrement asymétrique résout le problème de l'échange de clé, mais il introduit une nouvelle vulnérabilité : comment Bilbo peut-il être certain que la clé publique qu'il a récupérée appartient bien à Aragorn ?

Rien n'empêche un attaquant de s'intercaler entre les deux, d'intercepter la clé publique d'Aragorn et de la remplacer par la sienne. C'est l'**attaque par l'homme du milieu** (*man-in-the-middle*), illustrée ci-dessous.

Définition 14 — Attaque par l'homme du milieu

Dans une **attaque par l'homme du milieu** (man-in-the-middle), un attaquant s'intercale entre deux interlocuteurs. Il intercepte et relaie les messages en se faisant passer pour chacun d'eux auprès de l'autre.

Cette attaque compromet à la fois la **confidentialité** et l'**authenticité** de la communication, sans que les deux victimes ne s'en aperçoivent.



La parade : les certificats

C'est précisément pour contrer cette attaque que les certificats numériques ont été introduits dans HTTPS.

Un attaquant ne peut pas fabriquer un faux certificat valide sans accès à la clé privée de l'autorité de certification.

3.6.3 Le phishing

Contourner la cryptographie

Les attaques précédentes s'attaquent aux mécanismes techniques. Le **phishing** adopte une stratégie radicalement différente : il ne cherche pas à casser le chiffrement, il cherche à **tromper l'utilisateur** pour qu'il fournisse lui-même ses informations.

Un attaquant envoie un e-mail imitant l'apparence d'un message officiel (banque, administration, service en ligne). Le lien contenu dans l'e-mail pointe vers un faux site, dont l'adresse ressemble à celle du site légitime mais en diffère légèrement. Ce faux site peut même afficher un cadenas HTTPS valide, ce qui rassure à tort l'utilisateur.

Aragorn reçoit un e-mail de `secu@gondor-banque.net` lui demandant de confirmer ses identifiants sur `gondor-bânque.com` (notez l'accent sur le **â**). Le site est en HTTPS, le cadenas est présent. Mais le site appartient à Saroumane.

Définition 15 — Phishing

Le **phishing** (ou hameçonnage) est une attaque qui consiste à usurper l'identité d'un service de confiance pour inciter une victime à fournir volontairement des informations sensibles (identifiants, mots de passe, coordonnées bancaires).

Le phishing cible la **confiance humaine** plutôt que les algorithmes : aucun mécanisme cryptographique ne peut s'y opposer seul.

3.7 Bonus, cryptographie in the future

Et demain ? La menace quantique

Toute la sécurité de RSA repose sur une hypothèse : factoriser un grand entier est un problème **difficile**. Plus précisément, on ne connaît pas d'algorithme classique capable de le résoudre en temps polynomial, ce problème semble appartenir à la classe **NP** sans être dans **P**.

Les ordinateurs quantiques changent la donne. En 1994, Peter Shor a démontré qu'un ordinateur quantique suffisamment puissant pourrait factoriser n en temps polynomial, grâce à l'**algorithme de Shor**. RSA, et plus généralement toute la cryptographie asymétrique actuelle, serait alors cassé.

Les machines quantiques capables de casser RSA n'existent pas encore. Mais la menace est prise au sérieux, car un attaquant peut dès aujourd'hui **collecter des données chiffrées** pour les déchiffrer plus tard, une fois la technologie disponible.

La réponse s'appelle la **cryptographie post-quantique** : de nouveaux algorithmes, reposant sur des problèmes mathématiques qui résistent même aux ordinateurs quantiques, ont été standardisés par le NIST en 2024. La transition est déjà en cours.