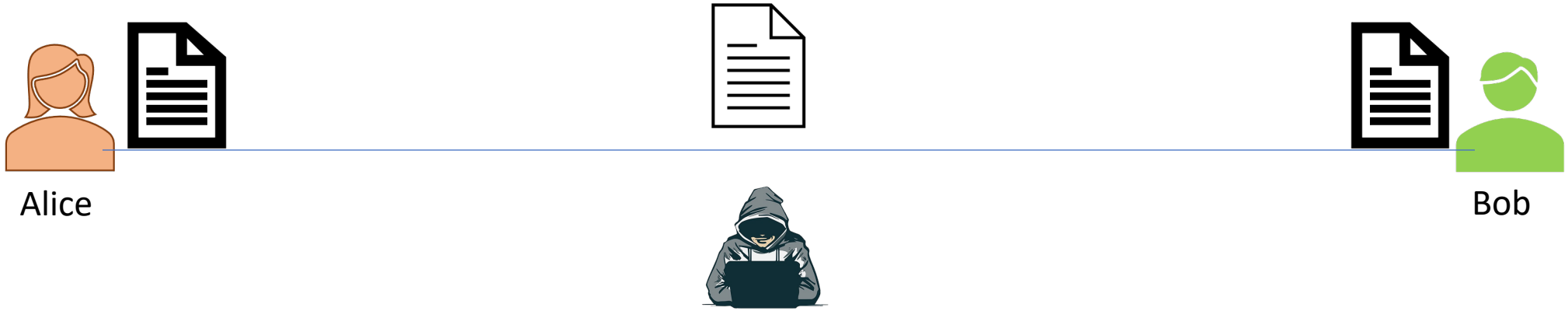
# COMMENT PROTÉGER LES DONNÉES TRANSMISES

- Les données transmises sur un support peuvent être intercepter par n'importe qui.

- On doit protéger les données de 3 façons :

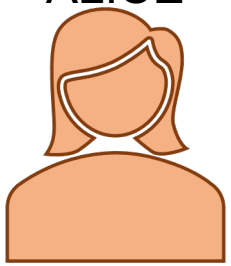
| | |
|---|---|
| Confidentialité | Les données sont accessibles seulement par Alice et Bob. |
| Intégrité | Les données ne sont pas modifiées entre Alice et Bob. |
| Authenticité | Alice et Bob sont réellement ceux qu'ils prétendent être. |



# RAPPEL SUR LE CHIFFRAGE ASYMÉTRIQUE

2. TRANSFERT D'UN EXEMPLAIRE DE LA CLE PUBLIQUE D'ALICE

ALICE



BOB

4. TRANSFERT DU FICHIER CHIFFRÉ

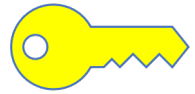
PUBLIQUE ALICE

5. DECHIFFRAGE DU FICHIER AVEC LA CLÉ PRIVÉE D'ALICE

3.CHIFFRAGE DU FICHIER AVEC LA CLÉ PUBLIQUE D'ALICE

PUBLIQUE BOB

PRIVÉE ALICE



PRIVÉE BOB

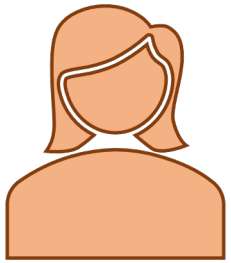
1. CRÉATION DE PAIRES DE CLÉS PRIVÉE ET PUBLIQUE

# MAN-IN-THE-MIDDLE

1. TRANSFERT DE LA CLE PUBLIQUE D'ALICE
(VOL DE LA CLÉ PUBLIQUE PAR UN PIRATE)

ALICE

BOB



4.RECHIFFRAGE D'UN AUTRE FICHIER AVEC LA CLÉ PUBLIQUE D'ALICE VOLÉE.

3.INTERCEPTION DU FICHIER

PUBLIQUE ALICE

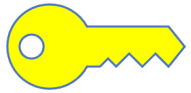
7.DÉCHIFFRAGE DU FICHIER AVEC LA CLÉ PRIVÉE D'ALICE

PUBLIQUE ALICE

2.CHIFFRAGE DU FICHIER AVEC CLÉ PUBLIQUE D'ALICE

PUBLIQUE BOB

PRIVÉE ALICE

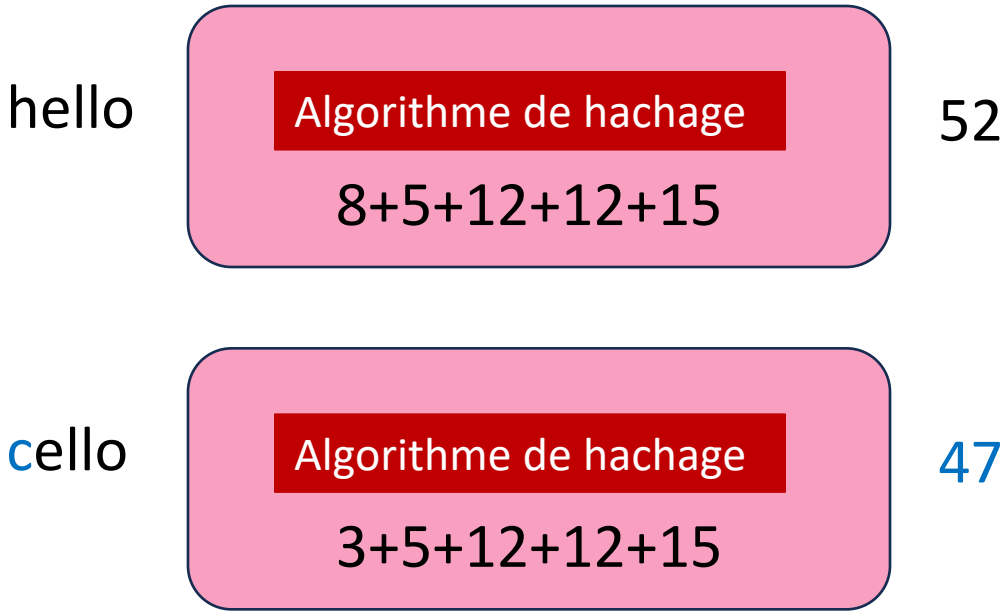


PRIVÉE BOB

Le message reçu par Alice n'est pas authentique (venant du pirate).

PUBLIQUE ALICE

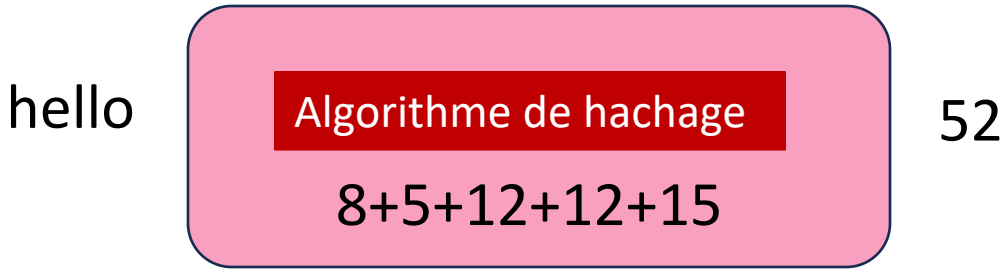
# INTÉGRITÉ : HACHAGE




Le résultat obtenu est appelé empreinte ou digest/fingerprint en anglais.

Si on change le message original, l'empreinte change.

# INTÉGRITÉ : HASHAGE




Cet exemple est extrêmement simplifié. Dans la réalité, l'algorithme de hashage doit satisfaire 4 points essentiels :

- Impossibilité de produire une empreinte donnée.
- Impossibilité d'extraire le message original.
- Un léger changement du message original modifie drastiquement l'empreinte.
- Le résultat de l'empreinte a une longueur fixe.

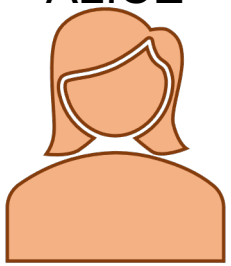
# INTÉGRITÉ : HACHAGE

- Impossibilité de produire une empreinte donnée.
- Impossibilité d'extraire le message original.
- Un léger changement du message original modifie drastiquement l'empreinte.
- Le résultat de l'empreinte a une longueur fixe.

```
root@user1:/# echo "hello"
hello
root@user1:/# echo "hello" | md5sum
b1946ac92492d2347c6235b4d2611184  -
root@user1:/# echo "Le BTS CIEL est surement la formation la plus complete proposee sur parcours sup" | md5sum
bfc6f80845fe31647e00962a76231761  -
root@user1:/# echo "Le BTS CIEL est surement la formation la plus complete proposee sur parcOurs sup" | md5sum
28c1703fec2a84337c7e8e7af043556f  -
root@user1:/#
```

# INTÉGRITÉ : HACHAGE

ALICE



45B..A4

2. TRANSFERT DU MESSAGE ET DE L'EMPREINTE

45B..A4

BOB

1. GENERATION
DE L'EMPREINTE

HASH

45B..A4

3. RE-CALCUL DE L'EMPREINTE ET
COMPARAISON AVEC LA VALEUR REÇUE.

HASH

45B..A4

# INTÉGRITÉ : HASHAGE

**Exemple d'algorithme de hashage :**

1.SHA-1 (Secure Hash Algorithm 1) : Maintenant considéré comme obsolète et vulnérable aux attaques. Il est recommandé de ne plus l'utiliser pour les applications nécessitant une sécurité solide.

2.SHA-256 (Secure Hash Algorithm 256 bits) : Il fait partie de la famille des fonctions de hachage SHA-2. SHA-256 produit une empreinte de 256 bits et est largement utilisé pour garantir l'intégrité des données et la sécurisation des communications.

3.SHA-3 (Secure Hash Algorithm 3) : SHA-3 est le dernier algorithme de hachage standardisé par le NIST (National Institute of Standards and Technology). Il offre une alternative aux fonctions SHA-2 et est conçu pour une sécurité et une performance accrues.

4.MD5 (Message Digest Algorithm 5) : Bien que vulnérable aux attaques, MD5 est toujours utilisé dans certains contextes non sécurisés, tels que la vérification de l'intégrité des fichiers ou la génération de checksums simples.

5.SHA-512 (Secure Hash Algorithm 512 bits) : Également faisant partie de la famille des fonctions de hachage SHA-2, SHA-512 produit une empreinte de 512 bits. Il offre une sécurité accrue par rapport à SHA-256, mais nécessite également plus de ressources computationnelles.

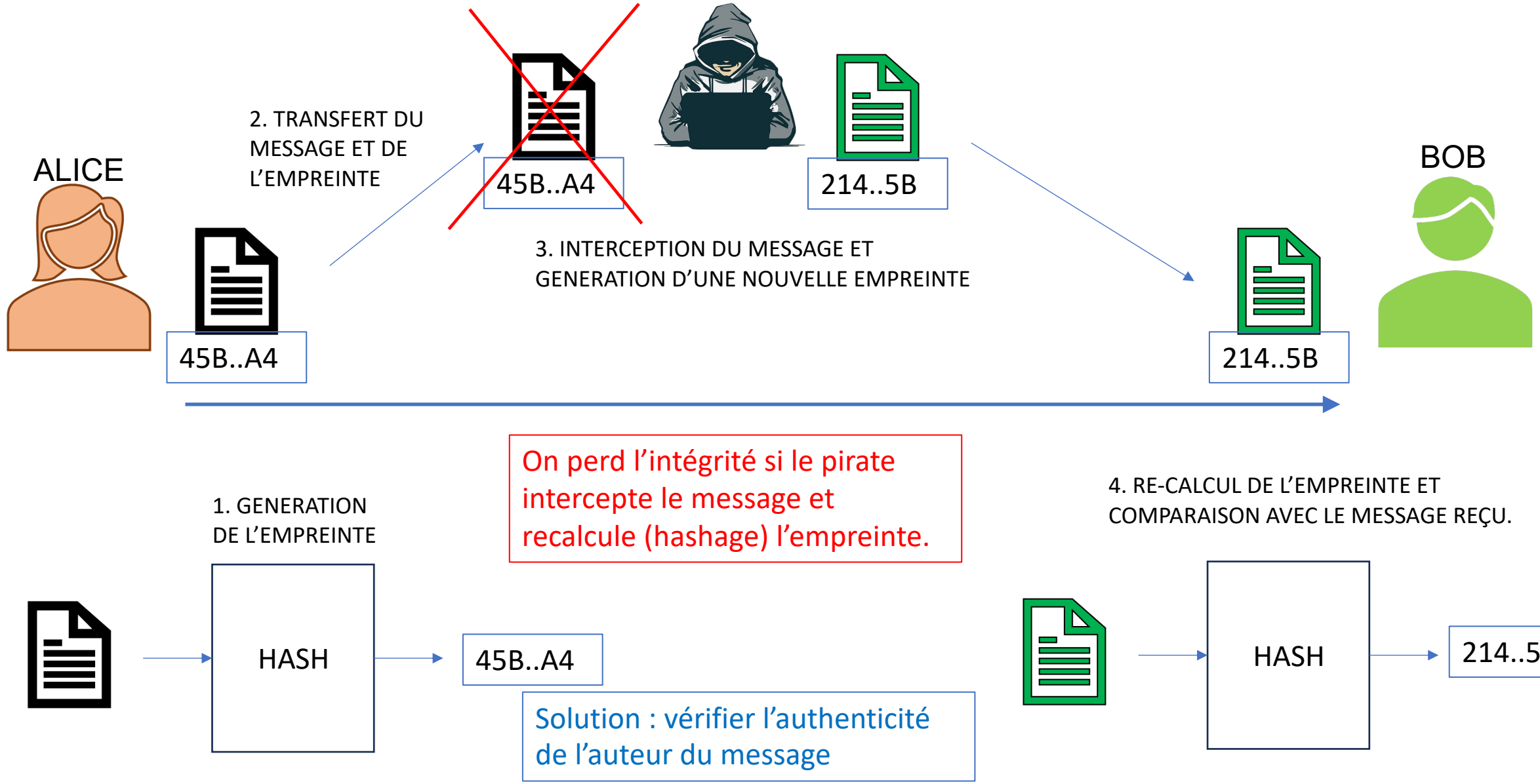
6. RIPEMD-160 : C'est un algorithme de hachage cryptographique développé en Europe, principalement utilisé dans les protocoles de blockchain tels que Bitcoin.

RainbowCrack est un outil de craquage de mot de passe qui utilise les tables arc-en-ciel pour des recherches rapides. Ces tables sont volumineuses.

## Rainbow Table Specification

| Algorithm | Table ID | Charset | Plaintext Length | Key Space | Success Rate | Table Size | Files |
|---|---|---|---|---|---|---|---|
| LM | lm_ascii-32-65-123-4#1-7 | ascii-32-65-123-4 | 1 to 7 | 7,555,858,447,479 ≈ $2^{42.8}$ | 99.9 % | 27 GB | Files |
| NTLM | ntlm_ascii-32-95#1-7 | ascii-32-95 | 1 to 7 | 70,576,641,626,495 ≈ $2^{46.0}$ | 99.9 % | 52 GB | Files |
| NTLM | ntlm_ascii-32-95#1-8 | ascii-32-95 | 1 to 8 | 6,704,780,954,517,120 ≈ $2^{52.6}$ | 96.8 % | 460 GB | Files |
| NTLM | ntlm_mixalpha-numeric#1-8 | mixalpha-numeric | 1 to 8 | 221,919,451,578,090 ≈ $2^{47.7}$ | 99.9 % | 127 GB | Files |
| NTLM | ntlm_mixalpha-numeric#1-9 | mixalpha-numeric | 1 to 9 | 13,759,005,997,841,642 ≈ $2^{53.6}$ | 96.8 % | 690 GB | Files |
| NTLM | ntlm_loweralpha-numeric#1-9 | loweralpha-numeric | 1 to 9 | 104,461,669,716,084 ≈ $2^{46.6}$ | 99.9 % | 65 GB | Files |
| NTLM | ntlm_loweralpha-numeric#1-10 | loweralpha-numeric | 1 to 10 | 3,760,620,109,779,060 ≈ $2^{51.7}$ | 96.8 % | 316 GB | Files |
| MD5 | md5_ascii-32-95#1-7 | ascii-32-95 | 1 to 7 | 70,576,641,626,495 ≈ $2^{46.0}$ | 99.9 % | 52 GB | Files |
| MD5 | md5_ascii-32-95#1-8 | ascii-32-95 | 1 to 8 | 6,704,780,954,517,120 ≈ $2^{52.6}$ | 96.8 % | 460 GB | Files |
| MD5 | md5_mixalpha-numeric#1-8 | mixalpha-numeric | 1 to 8 | 221,919,451,578,090 ≈ $2^{47.7}$ | 99.9 % | 127 GB | Files |
| MD5 | md5_mixalpha-numeric#1-9 | mixalpha-numeric | 1 to 9 | 13,759,005,997,841,642 ≈ $2^{53.6}$ | 96.8 % | 690 GB | Files |
| MD5 | md5_loweralpha-numeric#1-9 | loweralpha-numeric | 1 to 9 | 104,461,669,716,084 ≈ $2^{46.6}$ | 99.9 % | 65 GB | Files |
| MD5 | md5_loweralpha-numeric#1-10 | loweralpha-numeric | 1 to 10 | 3,760,620,109,779,060 ≈ $2^{51.7}$ | 96.8 % | 316 GB | Files |
| SHA1 | sha1_ascii-32-95#1-7 | ascii-32-95 | 1 to 7 | 70,576,641,626,495 ≈ $2^{46.0}$ | 99.9 % | 52 GB | Files |
| SHA1 | sha1_ascii-32-95#1-8 | ascii-32-95 | 1 to 8 | 6,704,780,954,517,120 ≈ $2^{52.6}$ | 96.8 % | 460 GB | Files |
| SHA1 | sha1_mixalpha-numeric#1-8 | mixalpha-numeric | 1 to 8 | 221,919,451,578,090 ≈ $2^{47.7}$ | 99.9 % | 127 GB | Files |
| SHA1 | sha1_mixalpha-numeric#1-9 | mixalpha-numeric | 1 to 9 | 13,759,005,997,841,642 ≈ $2^{53.6}$ | 96.8 % | 690 GB | Files |
| SHA1 | sha1_loweralpha-numeric#1-9 | loweralpha-numeric | 1 to 9 | 104,461,669,716,084 ≈ $2^{46.6}$ | 99.9 % | 65 GB | Files |
| SHA1 | sha1_loweralpha-numeric#1-10 | loweralpha-numeric | 1 to 10 | 3,760,620,109,779,060 ≈ $2^{51.7}$ | 96.8 % | 316 GB | Files |

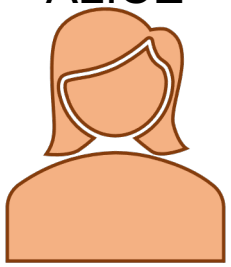
# INTÉGRITÉ : HACHAGE ET MAN-IN-THE-MIDDLE



# INTÉGRITÉ ET AUTHENTICITÉ :
# HMAC (Hash-based Message Authentication Code)

Intégrité — LE MESSAGE NE PEUT PAS ÊTRE MODIFIÉ

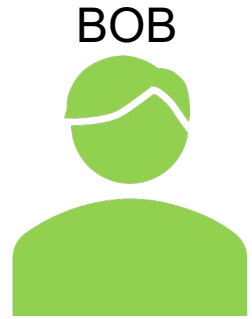
Authenticité — LA SOURCE DU MESSAGE EST IDENTIFIÉE À L'AIDE DE LA CLÉ



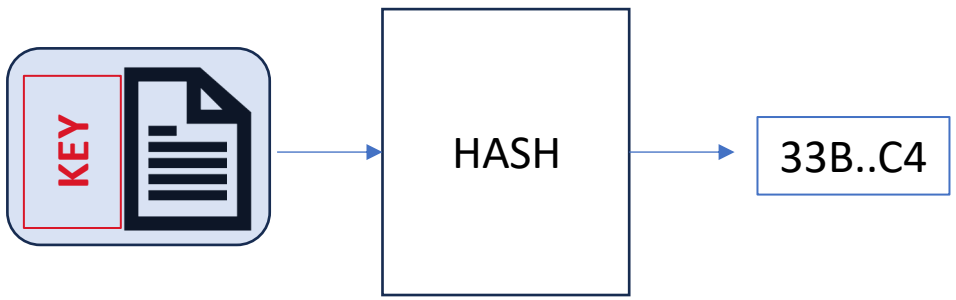

33B..C4

2. TRANSFERT DU MESSAGE ET DE L'EMPREINTE

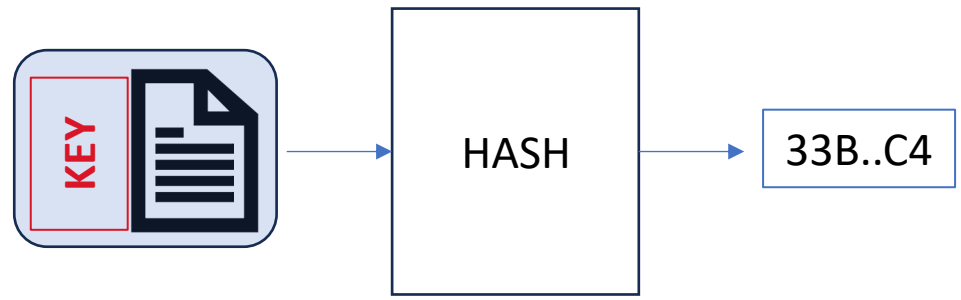
33B..C4




1. GENERATION DE L'EMPREINTE À PARTIR DU MESSAGE ET D'UNE CLÉ PARTAGÉE




3. RE-CALCUL DE L'EMPREINTE ET COMPARAISON AVEC LE MESSAGE REÇU ET LA CLÉ PARTAGÉE.



# INTÉGRITÉ ET AUTHENTICITÉ :
# HMAC (Hash-based Message Authentication Code)

Exemple de génération d'un hmac avec openssl :
Message : «Message important
Clé pré-partagée : **636c655f73656372657465**

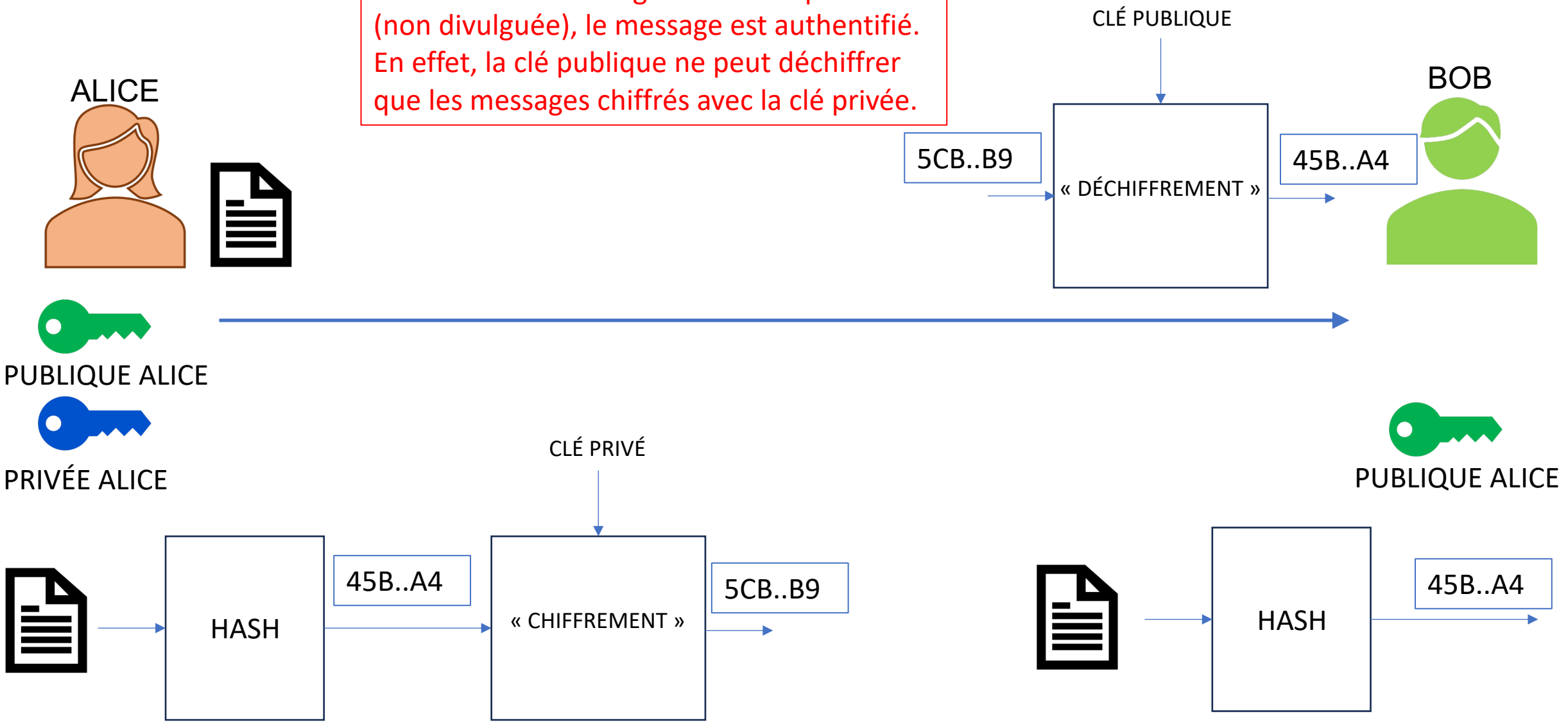
```
echo –n "Message important" | openssl dgst –sha256 –mac hmac –macopt
hexkey:636c655f73656372657465
```

Le standard HMAC permet l'utilisation de clés de n'importe quelle taille. En pratique, pour des raisons de sécurité, il est recommandé d'utiliser des clés d'une longueur qui n'est ni trop courte ni excessivement longue.

Une bonne pratique consiste à utiliser des clés d'une longueur comparable à la longueur de sortie de l'algorithme de hachage utilisé pour l'HMAC (32 octets pour SHA-256), car cela offre un bon équilibre entre facilité de gestion de la clé et sécurité.

# AUTHENTICITÉ : CLÉ PRIVÉE

En chiffrant le message avec la clé privée (non divulguée), le message est authentifié. En effet, la clé publique ne peut déchiffrer que les messages chiffrés avec la clé privée.



# AUTHENTICITÉ : CLÉ PRIVÉE

Création d’un fichier texte message.txt :

```
sudo nano message.txt
```

Création d’une clé privée RSA (private_key.pem) et extraction de la clé publique correspondante (public_key.pem) :

```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

Création d'une Signature Numérique :

```
openssl dgst –sha256 –sign private_key.pem –out signature.bin message.txt
```

Vérification de la Signature :

```
openssl dgst –sha256 –verify public_key.pem –signature signature.bin message.txt
```