Semaine 2 - Intégrité et Cryptographie Asymétrique

Créé par: RACHID BOUSELAMA

Superviseur: MR LAHCEN AIT IBOUREK

Intégrité et Cryptographie Asymétrique

Semaine 2 - Fondamentaux de la Sécurité et Cryptographie

Hachage Clé Publique Signature Numérique

Module: Fondamentaux de la Sécurité et Cryptographie

L'Intégrité et les Fonctions de Hachage

Définition

Transforme un message de taille **arbitraire** en une **empreinte de taille fixe**

Propriétés Clés

Unidirectionnel

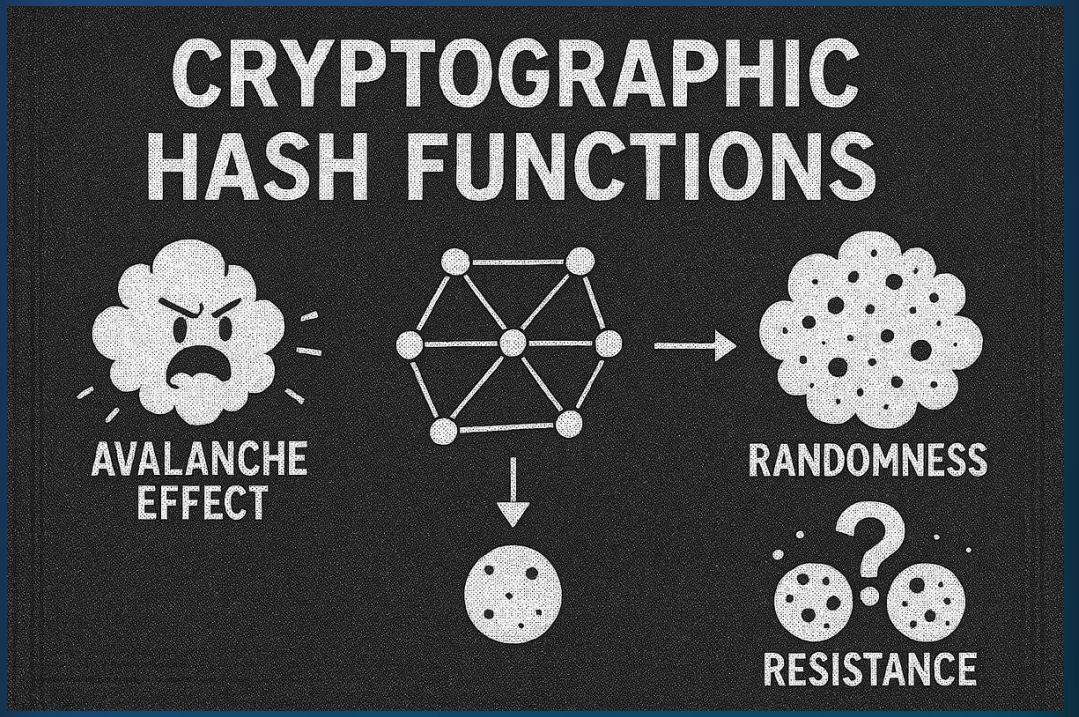
Impossible de retrouver le message original

Résistance aux collisions

Deux messages différents $\neq$ même hash

Effet Avalanche

1 bit modifié = 50% du hash changé



État de l'art des Algorithmes de Hachage

MD5 & SHA-1

Cassés • Collisions trouvables rapidement

SHA-256 (SHA-2)

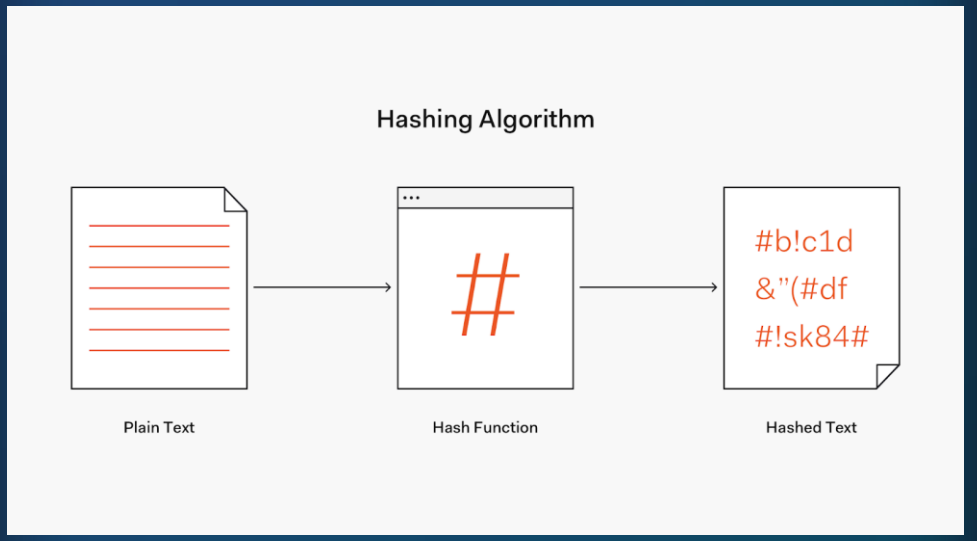
Standard actuel

Sortie sur 256 bits

SHA-3 (Keccak)

Nouvelle architecture

Résistant aux attaques théoriques



Stockage Sécurisé des Mots de Passe

Problème

Ne JAMAIS stocker un mot de passe en clair

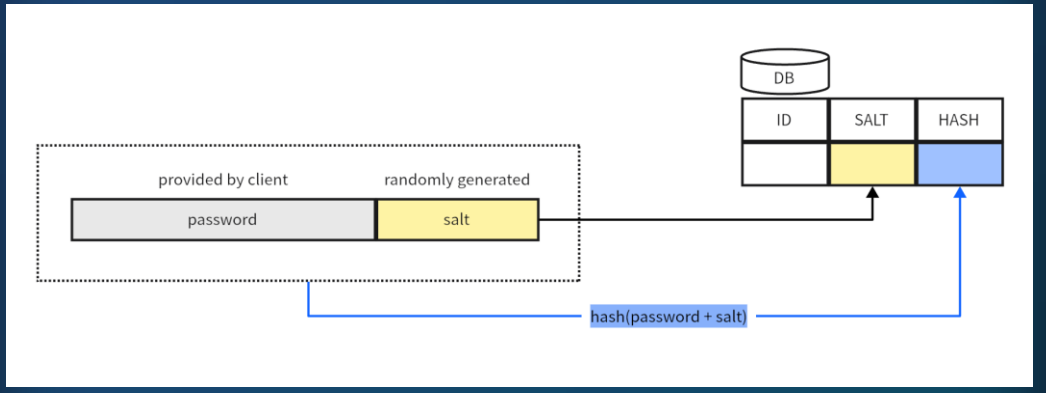
Risque

Stocker uniquement le hash vulnérable aux Rainbow Tables

Solution: Le Salage (Salting)

Ajouter une chaîne aléatoire unique avant le hachage

```
Hash(MotDePasse + Salt)
```



La Révolution Asymétrique: Clé Publique

Le Problème

Comment partager une clé sur un canal public?

Une Paire de Clés

- **Clé Publique:** Distribuée à tous (sert à chiffrer)
- **Clé Privée:** Gardée précieusement (sert à déchiffrer)

RSA

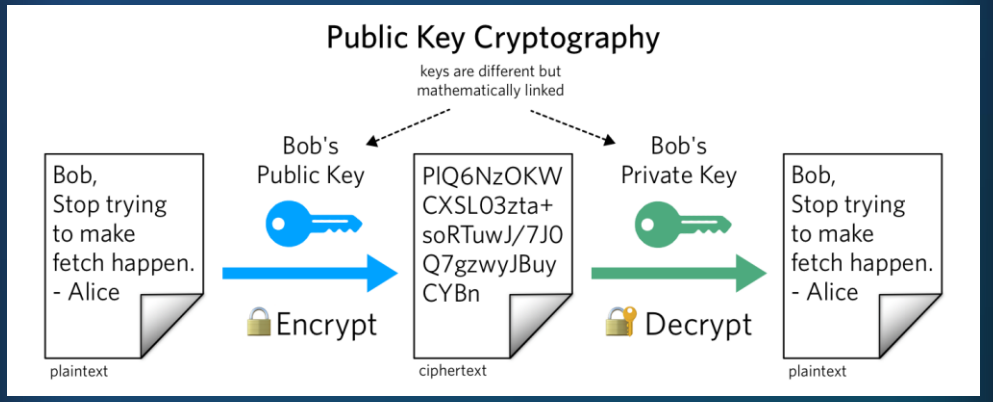
Basé sur la factorisation de grands nombres premiers

2048-4096 bits

ECC

Courbes Elliptiques, plus efficace

256 bits = 3072 bits RSA



Diffie-Hellman: Échange de Clés Sécurisé

Concept

Générer un secret commun **sans jamais l'échanger**

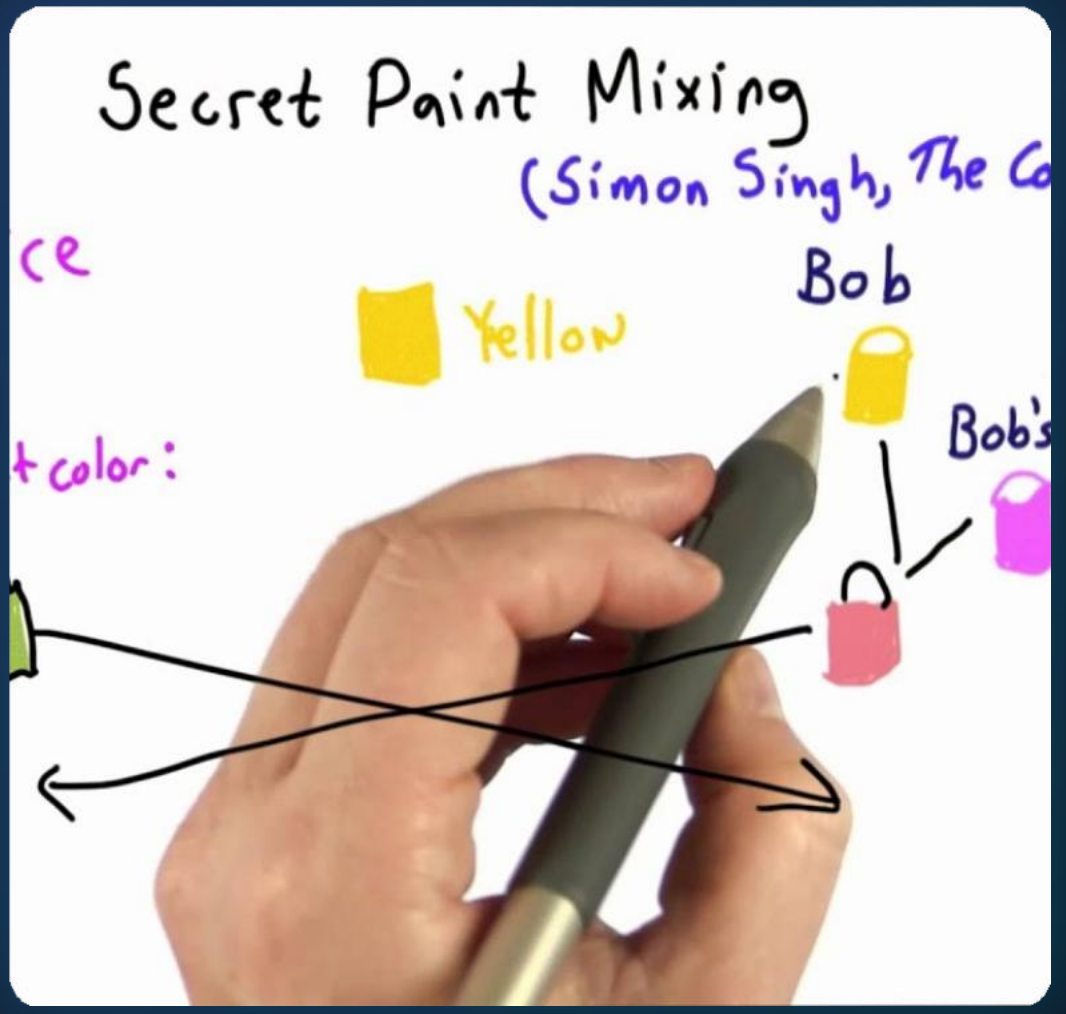
Date

1976 - **Révolution de la cryptographie**

Analogie de la Peinture

- Alice et Bob s'accordent sur une couleur publique
- Chacun ajoute sa couleur secrète et envoie le mélange
- Ils combinent leurs secrets pour obtenir la même couleur finale

Un espion ne peut pas séparer les couleurs



Cryptographie Hybride

Le Problème

RSA est **1000x plus lent** que AES

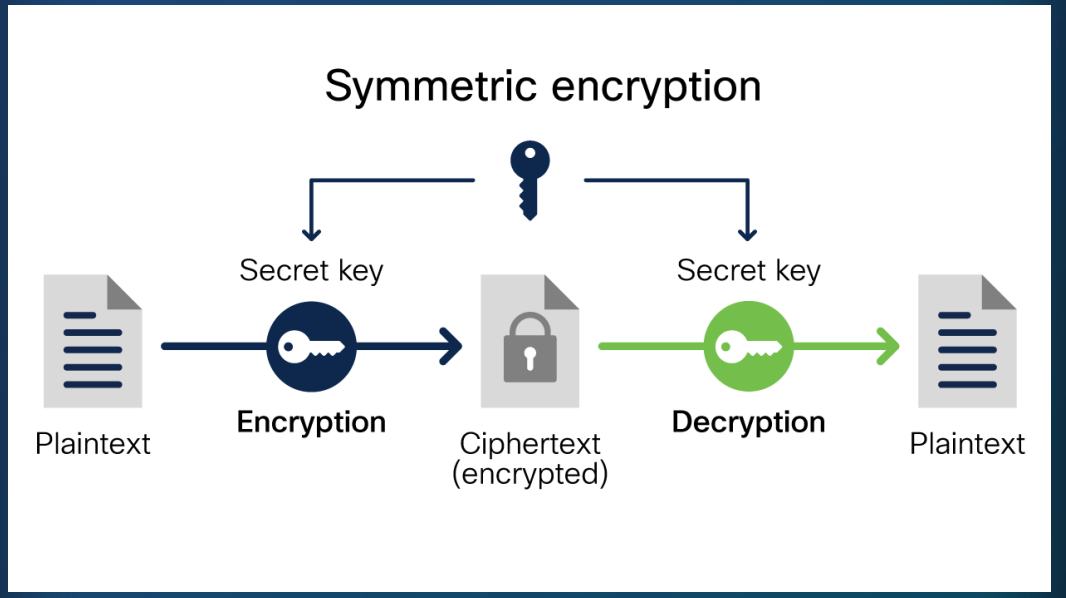
La Solution

Combiner **les deux approches**

Processus

1. Générer une clé AES aléatoire
2. Chiffrer les données avec AES (rapide)
3. Chiffrer la clé AES avec RSA
4. Envoyer [Données chiffrées AES] +[Clé AES chiffrée RSA]

Utilisé par HTTPS, SSH, PGP



La Signature Électronique

Objectif

Authenticité · **Intégrité** · **Non-répudiation**

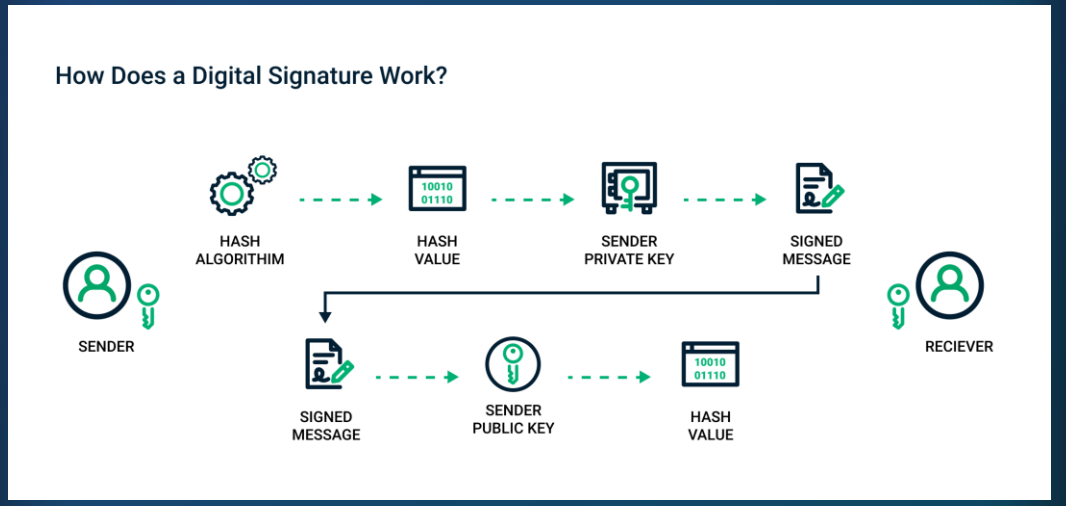
Processus de Signature

- Calculer le hash du document
- Chiffrer le hash avec la clé privée

Processus de Vérification

- Calculer le hash du document reçu
- Déchiffrer la signature avec la clé publique
- Comparer les deux hashes

Si identiques = Document authentique et intact



Travaux Pratiques

Intégrité et Cryptographie Asymétrique - Semaine 2

Machine virtuelle Linux ou Terminal avec OpenSSL

Hachage | Chiffrement Hybride | Signature Numérique

Module: Fondamentaux de la Sécurité et Cryptographie

Atelier 1: Fonctions de Hachage et Effet Avalanche

Objectif

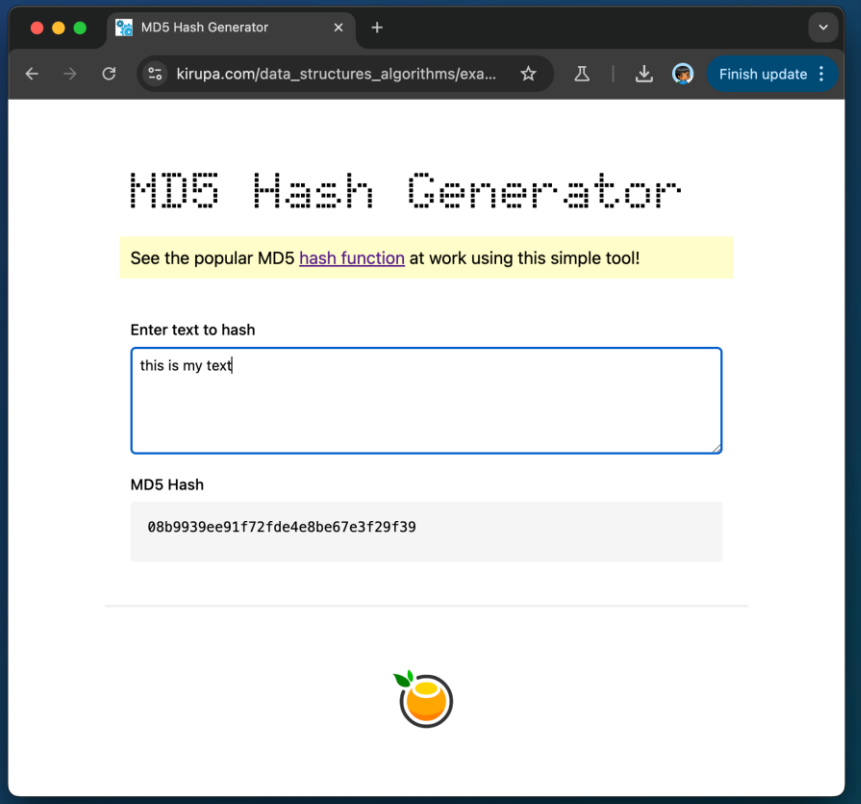
Prouver qu'une **modification mineure** change tout

1 Créer deux fichiers quasi-identiques

```
# Notez la différence 'h' vs 'e' à la fin
echo "Il fait beau a Marrakech" > msg1.txt
echo "Il fait beau a Marrakece" > msg2.txt
```

2 Calculer les empreintes

```
openssl dgst -md5 msg1.txt msg2.txt
openssl dgst -sha256 msg1.txt msg2.txt
```



Atelier 1: Observations et Interprétation

Observation Principale

Les sorties hexadécimales **n'ont rien en commun**

Effet Avalanche

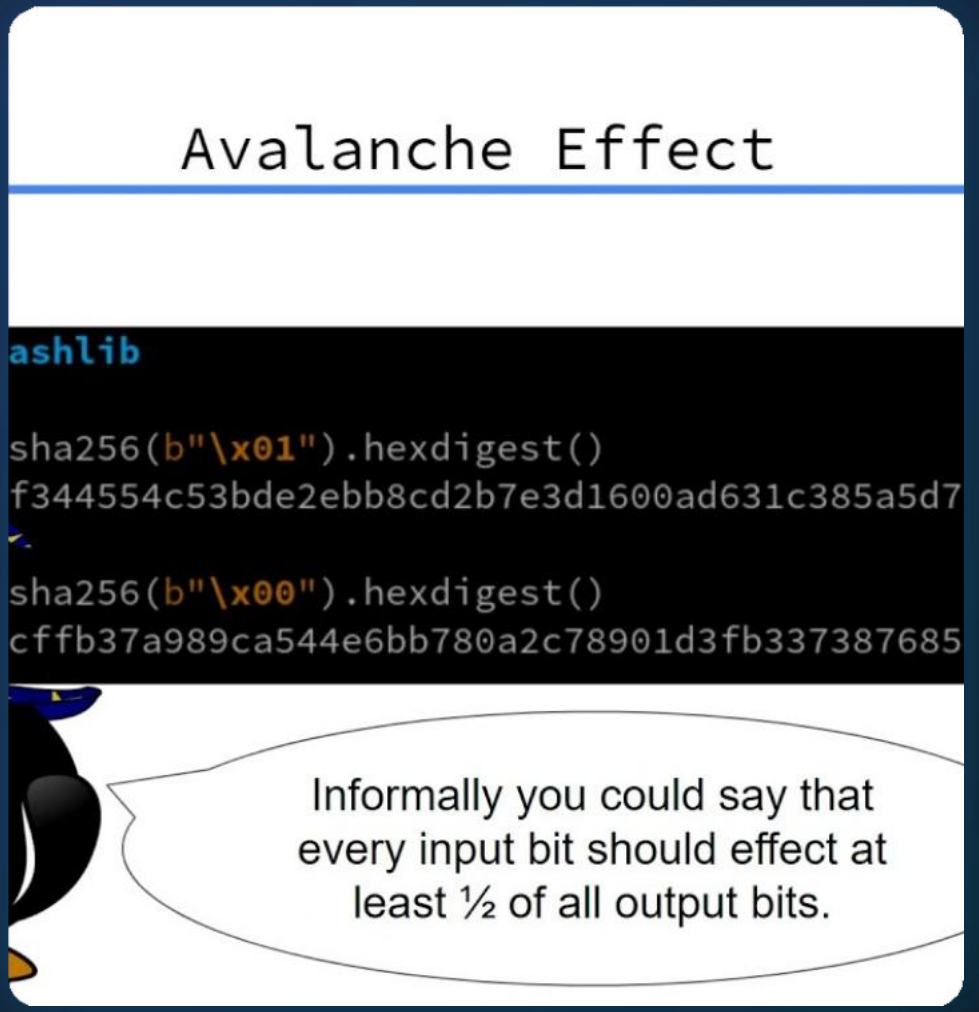
1 bit → 50%

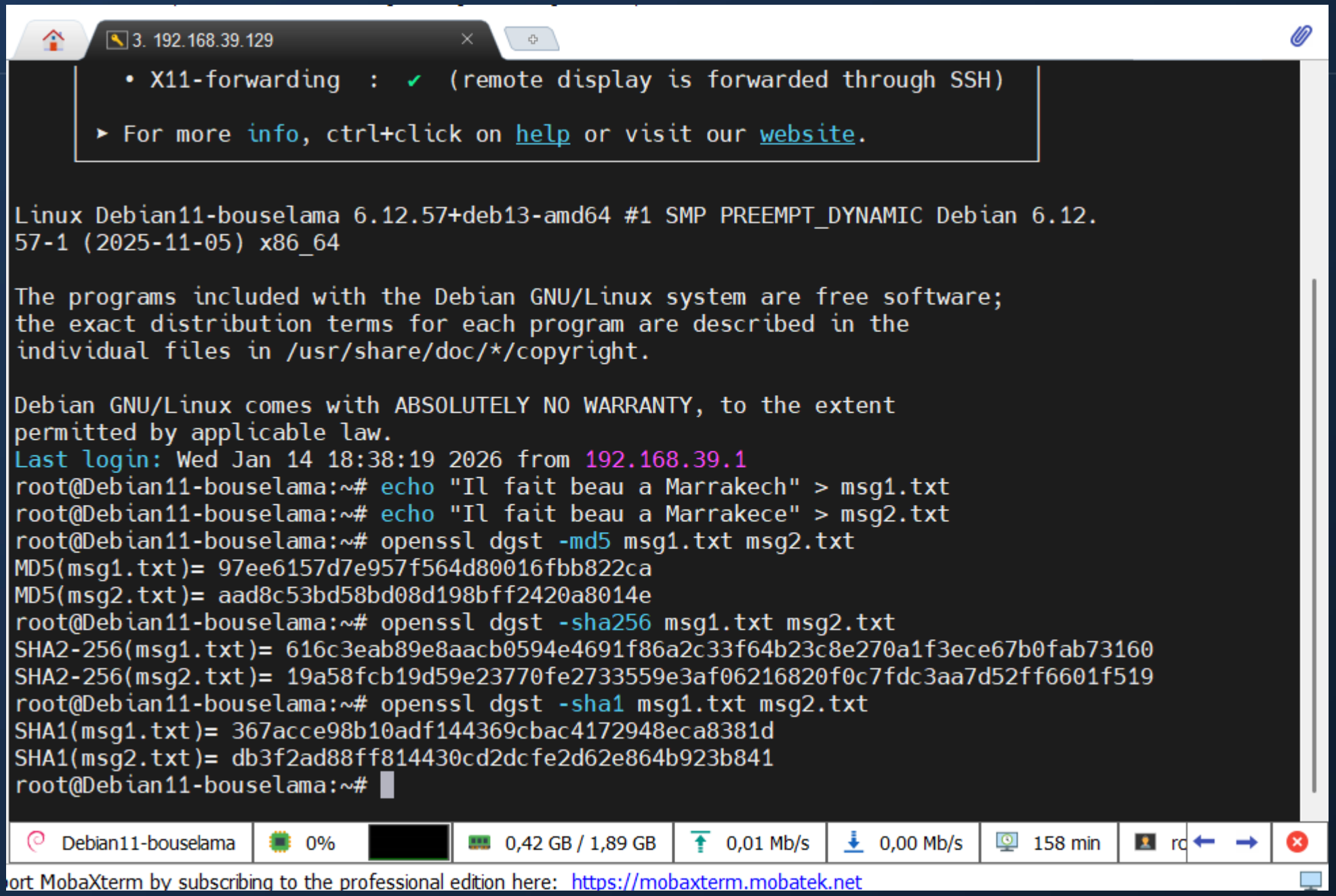
modifié = hash **totalement différent**

MD5 est Cassé

Collisions trouvables **rapidement**

```
# Création de collision (théorique)
fastcoll -o msg1.txt msg2.txt
```




3. 192.168.39.129
• X11-forwarding : ✔ (remote display is forwarded through SSH)
▸ For more info, ctrl+click on help or visit our website.
Linux Debian11-bouselama 6.12.57+deb13-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.12.
57-1 (2025-11-05) x86_64
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jan 14 18:38:19 2026 from 192.168.39.1
root@Debian11-bouselama:~# echo "Il fait beau a Marrakech" > msg1.txt
root@Debian11-bouselama:~# echo "Il fait beau a Marrakece" > msg2.txt
root@Debian11-bouselama:~# openssl dgst -md5 msg1.txt msg2.txt
MD5(msg1.txt)= 97ee6157d7e957f564d80016fbb822ca
MD5(msg2.txt)= aad8c53bd58bd08d198bff2420a8014e
root@Debian11-bouselama:~# openssl dgst -sha256 msg1.txt msg2.txt
SHA2-256(msg1.txt)= 616c3eab89e8aacb0594e4691f86a2c33f64b23c8e270a1f3ece67b0fab73160
SHA2-256(msg2.txt)= 19a58fcb19d59e23770fe2733559e3af06216820f0c7fdc3aa7d52ff6601f519
root@Debian11-bouselama:~# openssl dgst -sha1 msg1.txt msg2.txt
SHA1(msg1.txt)= 367acce98b10adf144369cbac4172948eca8381d
SHA1(msg2.txt)= db3f2ad88ff814430cd2dcfe2d62e864b923b841
root@Debian11-bouselama:~#
Debian11-bouselama
0%
0,42 GB / 1,89 GB
0,01 Mb/s
0,00 Mb/s
158 min
ort MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

Atelier 2: Chiffrement Hybride - Préparation (Bob)

Scénario

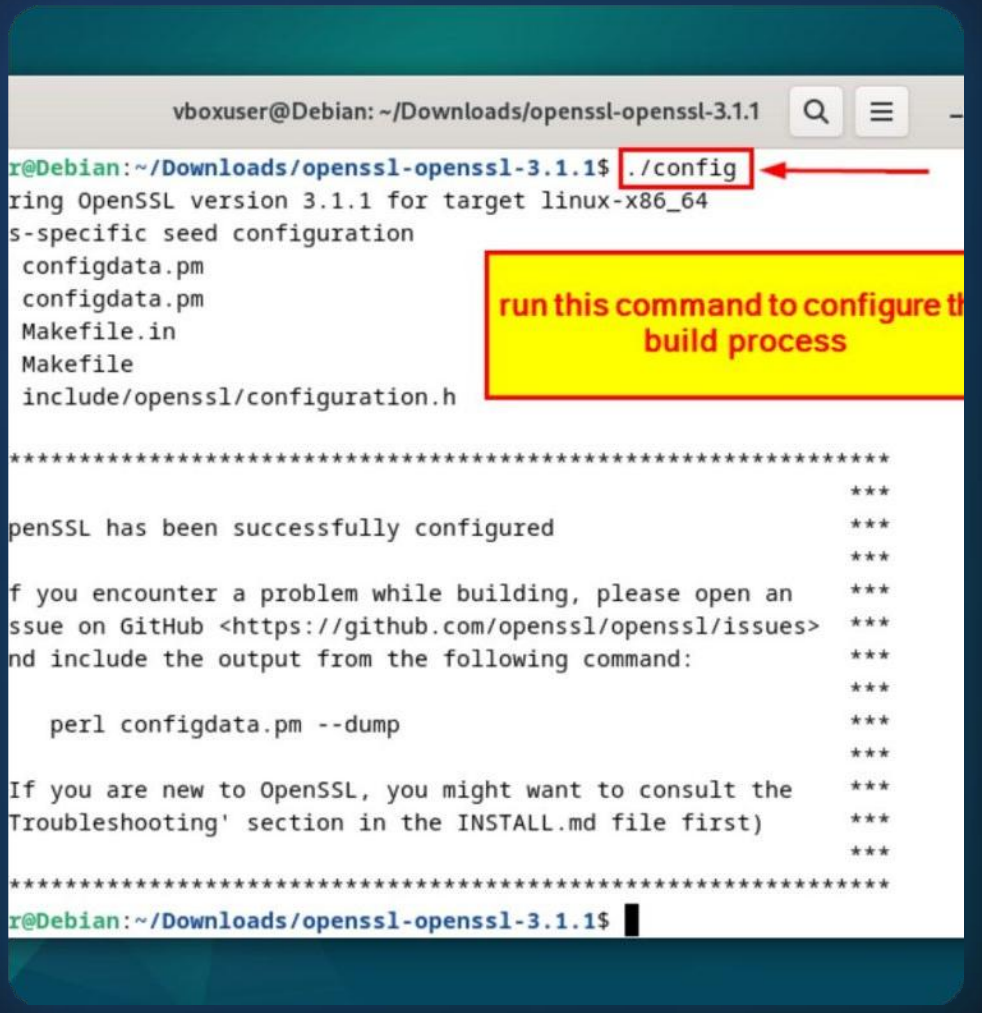
Alice envoie confidentiel.txt à Bob de manière sécurisée

1 Générer la paire de clés RSA de Bob

```
# Clé privée (4096 bits)
openssl genpkey -algorithm RSA -out bob_private.pem -
pkeyopt rsa_keygen_bits:4096

# Extraire la clé publique
openssl rsa -pubout -in bob_private.pem -out
bob_public.pem
```

Bob envoie bob_public.pem à Alice



Atelier 2: Chiffrement Hybride - Chiffrement (Alice)

1 **Créer le fichier de données**

```
echo "Donnees super secretes" > data.txt
```

2 **Générer une clé AES aléatoire**

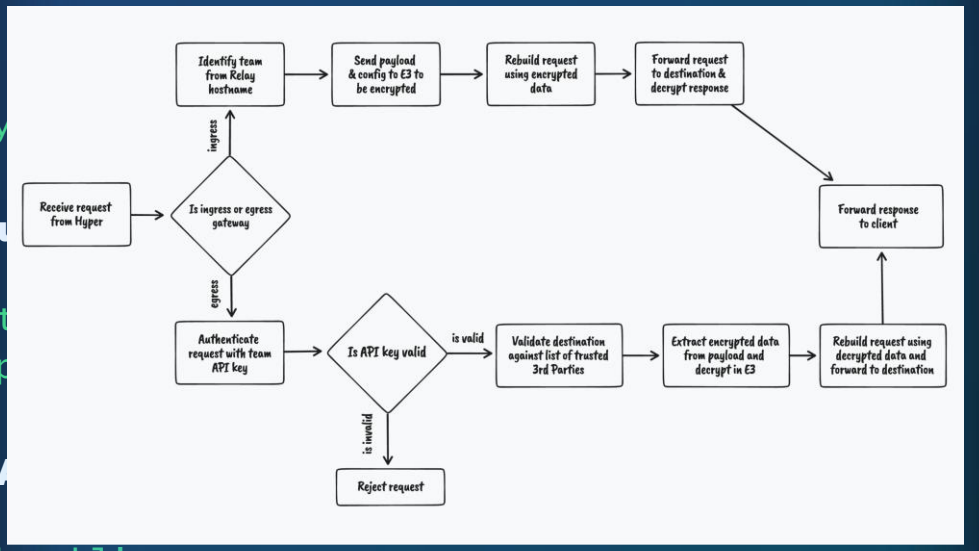
```
openssl rand -hex 32 > cle_aes.key
```

3 **Chiffrer les données avec AES (Symétriqu**

```
openssl enc -aes-256-cbc -in data.t
data.enc - pass file:cle_aes.key -p
```

4 **Chiffrer la clé AES avec la clé publique RS**

```
openssl pkeyutl -encrypt -inkey bob_public.pem -
pubin - in cle_aes.key -out cle_aes.enc
```



Alice envoie

Atelier 2: Chiffrement Hybride - Déchiffrement (Bob)

1 Déchiffrer la clé AES avec sa clé privée RSA

```
openssl pkeyutl -decrypt -inkey bob_private.pem -in
cle_aes.enc -out cle_recuperee.key
```

2 Déchiffrer les données avec la clé AES récupérée

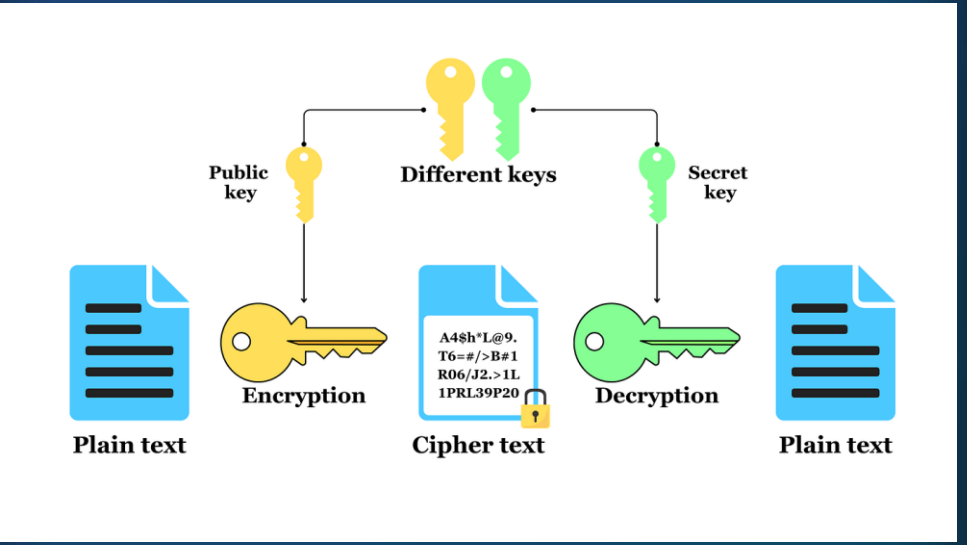
```
openssl enc -d -aes-256-cbc -in data.enc -out
data_final.txt -pass file:cle_recuperee.key -pbkdf2
```

Résultat: `data_final.txt` contient "Donnees super secretes"

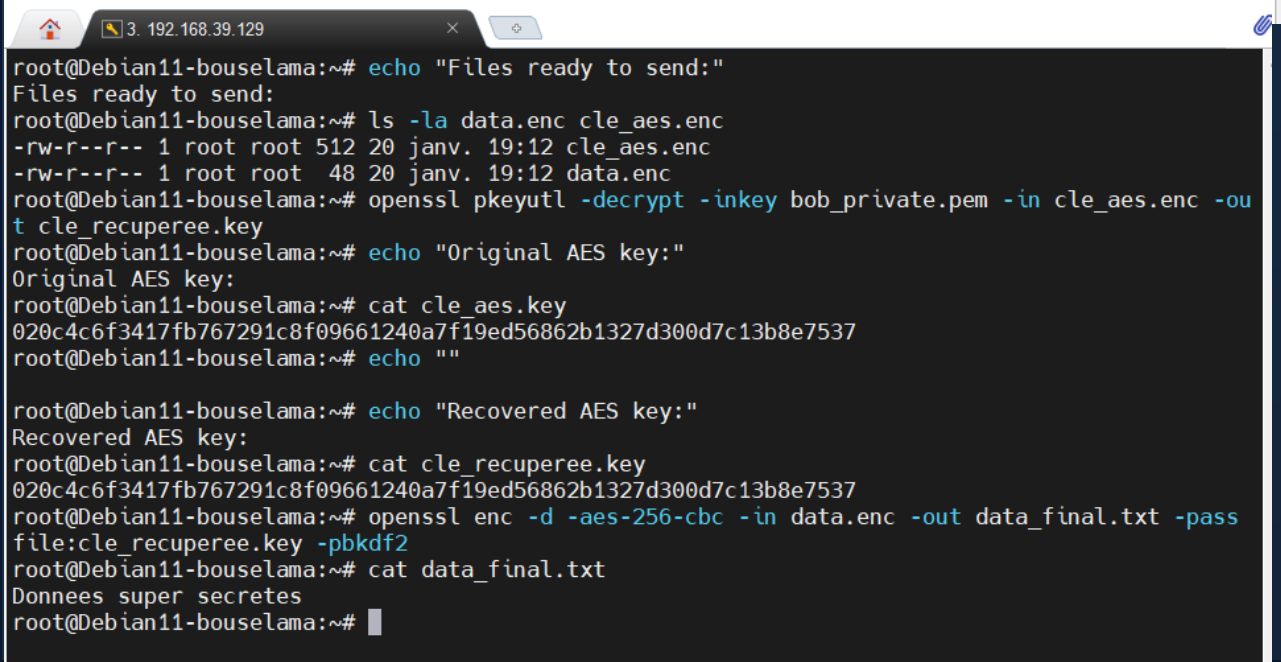
Concept Hybride

AES + Rapidité

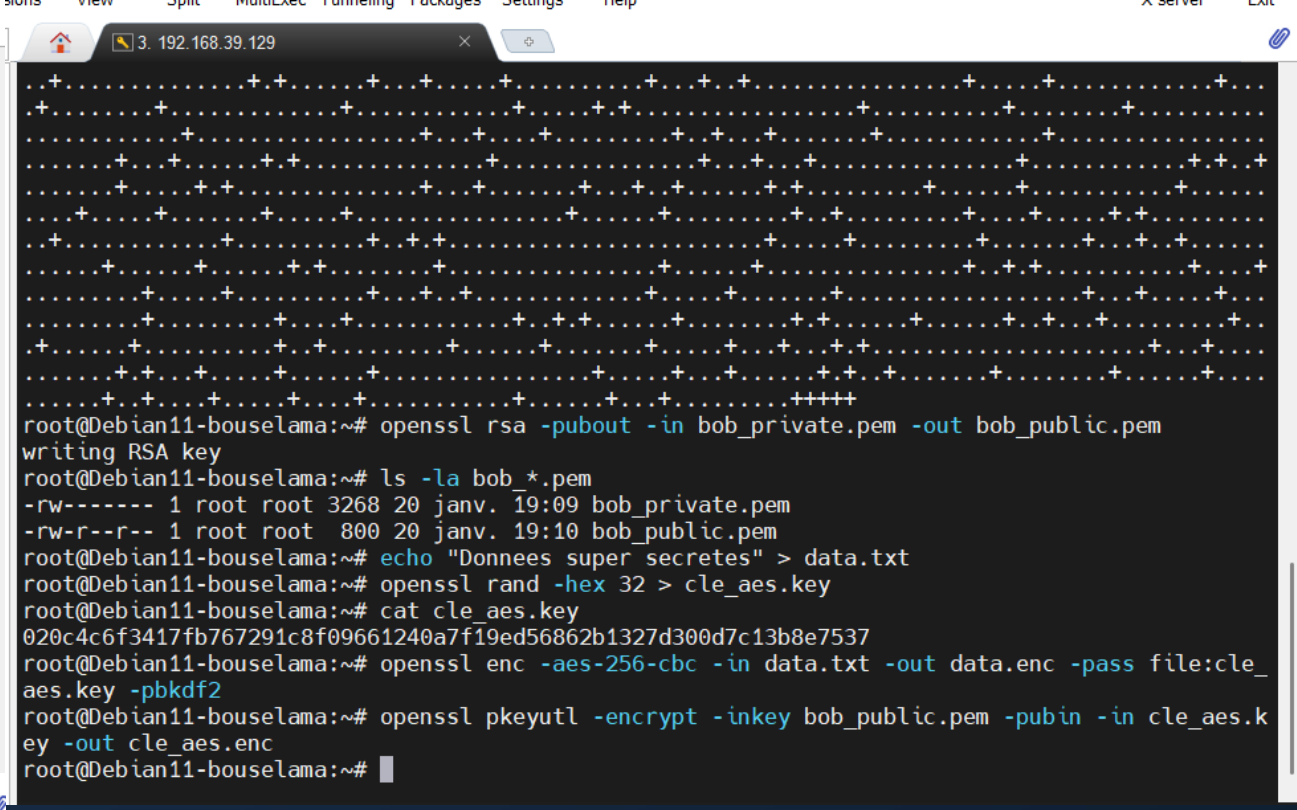
RSA + Sécurité




3. 192.168.39.129
SHA2-256(msg2.txt)= 19a58fcb19d59e23770fe2733559e3af06216820f0c7fdc3aa7d52ff6601f519
root@Debian11-bouselama:~# openssl dgst -sha1 msg1.txt msg2.txt
SHA1(msg1.txt)= 367acce98b10adf144369cbac4172948eca8381d
SHA1(msg2.txt)= db3f2ad88ff814430cd2dcfe2d62e864b923b841
root@Debian11-bouselama:~# openssl genpkey -algorithm RSA -out bob_private.pem -pkeyopt rsa_ke
ygen_bits:4096


3. 192.168.39.129
root@Debian11-bouselama:~# echo "Files ready to send:"
Files ready to send:
root@Debian11-bouselama:~# ls -la data.enc cle_aes.enc
-rw-r--r-- 1 root root 512 20 janv. 19:12 cle_aes.enc
-rw-r--r-- 1 root root 48 20 janv. 19:12 data.enc
root@Debian11-bouselama:~# openssl pkeyutl -decrypt -inkey bob_private.pem -in cle_aes.enc -ou
t cle_recuperee.key
root@Debian11-bouselama:~# echo "Original AES key:"
Original AES key:
root@Debian11-bouselama:~# cat cle_aes.key
020c4c6f3417fb767291c8f09661240a7f19ed56862b1327d300d7c13b8e7537
root@Debian11-bouselama:~# echo ""

root@Debian11-bouselama:~# echo "Recovered AES key:"
Recovered AES key:
root@Debian11-bouselama:~# cat cle_recuperee.key
020c4c6f3417fb767291c8f09661240a7f19ed56862b1327d300d7c13b8e7537
root@Debian11-bouselama:~# openssl enc -d -aes-256-cbc -in data.enc -out data_final.txt -pass
file:cle_recuperee.key -pbkdf2
root@Debian11-bouselama:~# cat data_final.txt
Donnees super secretes
root@Debian11-bouselama:~#


3. 192.168.39.129
root@Debian11-bouselama:~# openssl rsa -pubout -in bob_private.pem -out bob_public.pem
writing RSA key
root@Debian11-bouselama:~# ls -la bob_*.pem
-rw------- 1 root root 3268 20 janv. 19:09 bob_private.pem
-rw-r--r-- 1 root root 800 20 janv. 19:10 bob_public.pem
root@Debian11-bouselama:~# echo "Donnees super secretes" > data.txt
root@Debian11-bouselama:~# openssl rand -hex 32 > cle_aes.key
root@Debian11-bouselama:~# cat cle_aes.key
020c4c6f3417fb767291c8f09661240a7f19ed56862b1327d300d7c13b8e7537
root@Debian11-bouselama:~# openssl enc -aes-256-cbc -in data.txt -out data.enc -pass file:cle_
aes.key -pbkdf2
root@Debian11-bouselama:~# openssl pkeyutl -encrypt -inkey bob_public.pem -pubin -in cle_aes.k
ey -out cle_aes.enc
root@Debian11-bouselama:~#

Atelier 3: Signature Numérique - Signature et Vérification

1 Alice crée le contrat

```
echo "Je dois 100 dirhams a Bob" > contrat.txt
```

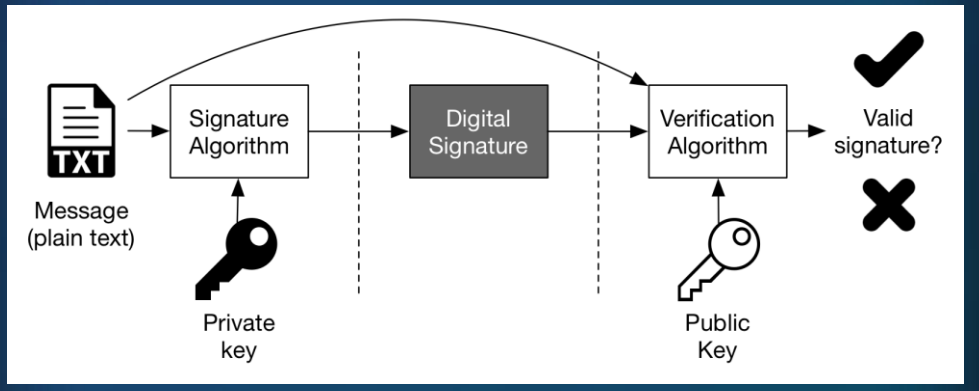
2 Alice signe le contrat

```
openssl dgst -sha256 -sign bob_private.pem -out
signature.bin contrat.txt
```

3 Bob vérifie la signature

```
openssl dgst -sha256 -verify bob_public.pem -signature
signature.bin contrat.txt
```

Résultat attendu: `Verified OK`



Atelier 3: Signature Numérique - L'Attaque (Tampering)

Scénario

Bob est malhonnête et modifie le contrat

1 Bob modifie le contrat

```
echo "Je dois 9000 dirhams a Bob" > contrat.txt
```

2 Bob essaie de vérifier la signature originale

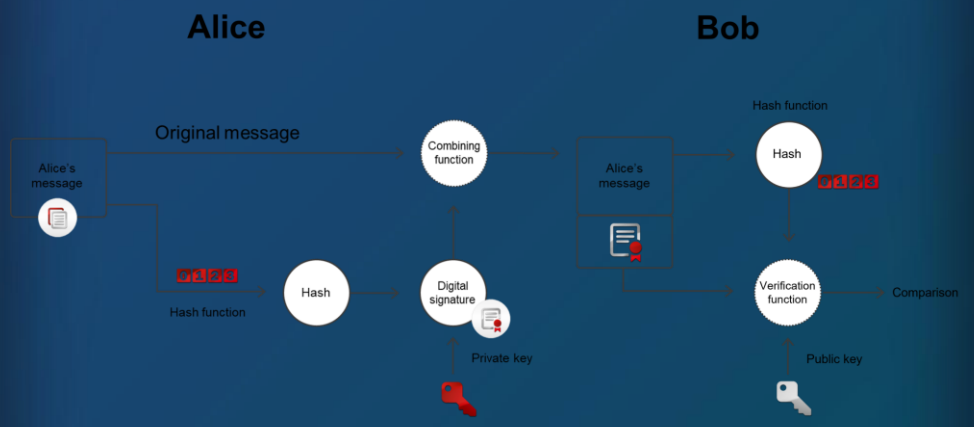
```
openssl dgst -sha256 -verify bob_public.pem -signature
signature.bin contrat.txt
```

Résultat: `Verification Failure`

Conclusion

La signature **protège contre la falsification**

Authenticité + Intégrité + Non-répudiation



3. 192.168.39.129

```
root@Debian11-bouselama:~# cat contrat.txt
Je dois 100 dirhams a Bob
root@Debian11-bouselama:~# openssl dgst -sha256 -sign bob_private.pem -out signature.bin contr
at.txt
root@Debian11-bouselama:~# xxd signature.bin | head
-bash: xxd : commande introuvable
root@Debian11-bouselama:~# ls -la signature.bin
-rw-r--r-- 1 root root 512 20 janv. 19:17 signature.bin
root@Debian11-bouselama:~# openssl dgst -sha256 -verify bob_public.pem -signature signature.bi
n contrat.txt
Verified OK
root@Debian11-bouselama:~# echo "Je dois 9000 dirhams a Bob" > contrat.txt
root@Debian11-bouselama:~# openssl dgst -sha256 -verify bob_public.pem -signature signature.bi
n contrat.txt
Verification failure
402726B8717F0000:error:02000068:rsa routines:ossl_rsa_verify:bad signature:../crypto/rsa/rsa_s
ign.c:442:
402726B8717F0000:error:1C880004:Provider routines:rsa_verify_directly:RSA lib:../providers/imp
lementations/signature/rsa_sig.c:1041:
root@Debian11-bouselama:~# echo "=== Compare the hashes ==="
=== Compare the hashes ===
root@Debian11-bouselama:~# openssl dgst -sha256 contrat.txt
SHA2-256(contrat.txt)= f7921764fddf2254b9478aeb33f27e571cc3a943ec14f91671bac8ad62ec9e1f
root@Debian11-bouselama:~# echo "Je dois 100 dirhams a Bob" > contrat.txt
root@Debian11-bouselama:~# openssl dgst -sha256 contrat.txt
SHA2-256(contrat.txt)= 627037f4e50a76c90928376067285064 8a06981ead17f6b53bb89351745ef2f9
root@Debian11-bouselama:~#
```