TP3 - Hachage, salage et sécurité de mots de passes

1 Fonctions de hachage

Génération de Hachages avec OpenSSL :

MD5 :

```
echo -n "Hello" | openssl dgst -md5
echo -n "OpenSSL" | openssl dgst -md5
echo -n "Hash Functions" | openssl dgst -md5
```

SHA-1 :
```
echo -n "Hello" | openssl dgst -sha1
echo -n "OpenSSL" | openssl dgst -sha1
echo -n "Hash Functions" | openssl dgst -sha1
```

SHA-256 :
```
echo -n "Hello" | openssl dgst -sha256
echo -n "OpenSSL" | openssl dgst -sha256
echo -n "Hash Functions" | openssl dgst -sha256
```

Comparaison des Longueurs des Hachages

1. **MD5** :
    - Longueur du hachage : 32 caractères hexadécimaux (128 bits).
2. **SHA-1** :
    - Longueur du hachage : 40 caractères hexadécimaux (160 bits).
3. **SHA-256** :
    - Longueur du hachage : 64 caractères hexadécimaux (256 bits).

Calculer la somme de contrôle MD5 d'un document volumineux :

Utilisez la commande suivante pour générer le hachage MD5 :

```
openssl dgst -md5 large_document.txt
```

1. **Modifier le document** :

Ajoutez un espace ou changez un caractère dans le fichier `large_document.txt`. Par exemple, ajoutez un espace à la fin du document ou changez une lettre.

1. **Recalculer le hachage MD5 du document modifié** :

Utilisez à nouveau la commande suivante pour générer le nouveau hachage MD5 :

openssl dgst -md5 large_document.txt

Remarques

- **Changement significatif du hachage** : Même une modification mineure dans le document, comme l'ajout d'un espace ou le changement d'un seul caractère, entraîne un changement radical du hachage MD5. Cela montre la sensibilité des fonctions de hachage cryptographiques aux modifications du contenu.
- **Intégrité des données** : Cette propriété est cruciale pour la vérification de l'intégrité des données. Si un fichier est modifié de quelque manière que ce soit, son hachage MD5 changera, indiquant que le fichier n'est plus identique à l'original.

Cette expérience démontre que les fonctions de hachage cryptographiques comme MD5 sont conçues pour produire des hachages complètement différents même pour des modifications mineures, assurant ainsi la détection de toute altération dans les documents.

Calculer la somme de contrôle SHA-256 d'un document volumineux :

Utilisez la commande suivante pour générer le hachage SHA-256 :

openssl dgst -sha256 large_document.txt

Ajoutez un espace ou changez un caractère dans le fichier `large_document.txt`. Par exemple, ajoutez un espace à la fin du document ou changez une lettre.

Utilisez à nouveau la commande suivante pour générer le nouveau hachage SHA-256 :

openssl dgst -sha256 large_document.txt

Remarques

- **Changement significatif du hachage** : Comme avec MD5, une modification mineure dans le document entraîne un changement radical du hachage SHA-256. Cela montre la sensibilité des fonctions de hachage cryptographiques aux modifications du contenu.
- **Intégrité des données** : Cette propriété est cruciale pour la vérification de l'intégrité des données. Si un fichier est modifié de quelque manière que ce soit, son hachage SHA-256 changera, indiquant que le fichier n'est plus identique à l'original.

Question 3 Protocole perso LOL

Protocole pour Assurer l'Authenticité d'un Fichier

1. **Génération des Paires de Clés** :
    - Le destinataire génère une paire de clés publique/privée RSA.
    - La clé publique est partagée avec l'expéditeur, tandis que la clé privée est conservée secrète par le destinataire.
2. **Préparation du Fichier** :
    - L'expéditeur calcule le hachage SHA-256 du fichier.
    - L'expéditeur chiffre ce hachage avec la clé privée RSA pour créer une signature numérique.
    - Le fichier et la signature numérique sont envoyés au destinataire.
3. **Vérification par le Destinataire** :
    - Le destinataire reçoit le fichier et la signature numérique.
    - Il calcule le hachage SHA-256 du fichier reçu.
    - Il déchiffre la signature numérique avec la clé publique RSA pour obtenir le hachage original.
    - Il compare les deux hachages :
        - Si les hachages correspondent, l'authenticité et l'intégrité du fichier sont vérifiées.
        - Si les hachages ne correspondent pas, le fichier a été altéré ou n'est pas authentique.

Mise en Place du Protocole avec OpenSSL

1. Génération des Paires de Clés RSA

sh
Copier le code

```
# Générer la clé privée
openssl genpkey -algorithm RSA -out private_key.pem -aes256

# Générer la clé publique
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

2. Préparation du Fichier par l'Expéditeur

sh
Copier le code

```
# Calculer le hachage SHA-256 du fichier
openssl dgst -sha256 -out hash.txt large_document.txt

# Signer le hachage avec la clé privée
openssl rsautl -sign -inkey private_key.pem -in hash.txt -out
signature.bin
```

```
# Envoyer le fichier et la signature numérique au destinataire
# (utiliser un protocole de transfert comme SCP, SFTP, ou un service
de messagerie)
```

3. Vérification par le Destinataire

sh

Copier le code

```
# Calculer le hachage SHA-256 du fichier reçu
openssl dgst -sha256 -out hash_received.txt large_document.txt

# Déchiffrer la signature numérique avec la clé publique
openssl rsautl -verify -inkey public_key.pem -pubin -in
signature.bin -out hash_decrypted.txt

# Comparer les hachages
diff hash_received.txt hash_decrypted.txt

# Si la commande diff ne retourne aucune différence, le fichier est
authentique
# Sinon, le fichier a été altéré ou n'est pas authentique
```

2 Salage et sécurité de mots de passes :

Pour réaliser ce script Python, nous allons suivre ces étapes :

1. Créer un fichier texte contenant 10 mots de passe distincts.
2. Hacher chaque mot de passe et les enregistrer dans un autre fichier texte.
3. Saler et hacher chaque mot de passe, puis les enregistrer dans un troisième fichier texte.

Ensuite, nous allons démontrer l'importance du salage pour sécuriser les mots de passe contre les attaques par tables arc-en-ciel.

Voici le script Python complet :

import hashlib
import os

# Liste des mots de passe couramment utilisés
common_passwords = ["123456", "admin", "1234", "password", "123", "Aa123456", "111111", "000000", "admin123", "root"]

```
# Liste de 10 mots de passe distincts, incluant au moins un des mots de passe courants
passwords = ["mypassword1", "123456", "securePass", "admin", "helloWorld", "test123",
"pass1234", "randomPwd", "newAdmin", "root123"]

# Fichiers de sortie
password_file = "passwords.txt"
hashed_file = "hashed_passwords.txt"
salted_hashed_file = "salted_hashed_passwords.txt"

# Fonction pour hacher un mot de passe
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

# Fonction pour saler et hacher un mot de passe
def salt_and_hash_password(password):
    salt = os.urandom(16).hex()  # Générer un sel aléatoire de 16 octets
    salted_password = salt + password
    hashed = hashlib.sha256(salted_password.encode()).hexdigest()
    return f"{salt}:{hashed}"

# Écrire les mots de passe dans le fichier
with open(password_file, 'w') as f:
    for pwd in passwords:
        f.write(pwd + "\n")

# Écrire les mots de passe hachés dans le fichier
with open(hashed_file, 'w') as f:
    for pwd in passwords:
        hashed = hash_password(pwd)
        f.write(hashed + "\n")

# Écrire les mots de passe salés et hachés dans le fichier
with open(salted_hashed_file, 'w') as f:
    for pwd in passwords:
        salted_hashed = salt_and_hash_password(pwd)
        f.write(salted_hashed + "\n")

print("Fichiers générés :")
print(f"- {password_file}")
print(f"- {hashed_file}")
print(f"- {salted_hashed_file}")
```

Question 5

Voici un script Python qui permet à votre camarade de déterminer quels mots de passe courants sont présents dans votre liste de mots de passe hachés :

Étapes

1. **Lire les hachages des mots de passe depuis le fichier.**
2. **Calculer les hachages des mots de passe couramment utilisés.**
3. **Comparer les hachages et déterminer les correspondances.**

```
import hashlib

# Liste des mots de passe couramment utilisés
common_passwords = ["123456", "admin", "1234", "password", "123", "Aa123456", "111111",
"000000", "admin123", "root"]

# Fichier contenant les mots de passe hachés
hashed_file = "hashed_passwords.txt"

# Fonction pour hacher un mot de passe avec SHA-256
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

# Lire les hachages des mots de passe depuis le fichier
with open(hashed_file, 'r') as f:
    hashed_passwords = [line.strip() for line in f]

# Dictionnaire pour stocker les correspondances
found_passwords = {}

# Comparer les hachages
for common_pwd in common_passwords:
    hashed_common_pwd = hash_password(common_pwd)
    if hashed_common_pwd in hashed_passwords:
        found_passwords[common_pwd] = hashed_common_pwd

# Afficher les correspondances
if found_passwords:
    print("Les mots de passe couramment utilisés présents dans la liste sont :")
    for pwd, hash_val in found_passwords.items():
        print(f"Mot de passe : {pwd} - Hachage : {hash_val}")
else:
    print("Aucun des mots de passe couramment utilisés n'est présent dans la liste.")
```

Explications

1. **Liste des mots de passe couramment utilisés** : Nous avons une liste `common_passwords` contenant les 10 mots de passe les plus courants.
2. **Fonction de hachage** : `hash_password` utilise SHA-256 pour hacher un mot de passe.

3. **Lecture des mots de passe hachés** : Le script lit les hachages des mots de passe depuis `hashed_file`.
4. **Comparaison des hachages** : Pour chaque mot de passe courant, le script calcule son hachage et vérifie s'il est présent dans la liste des hachages lus.
5. **Affichage des résultats** : Le script affiche les mots de passe courants trouvés dans la liste hachée.

Utilisation

- Votre camarade doit exécuter ce script en s'assurant que le fichier `hashed_passwords.txt` est dans le même répertoire que le script.
- Le script comparera les hachages et affichera les correspondances trouvées, montrant quels mots de passe courants sont présents dans votre liste de mots de passe hachés.

Ce script permet d'illustrer comment une attaque de type table arc-en-ciel pourrait être effectuée, soulignant l'importance d'utiliser des mots de passe forts et de saler les mots de passe avant de les hacher.