Travaux pratiques
Confidentialité, Intégrité et Authenticité des Données

Rendu : Repository (e.g. GitHub) ou archive contenant les scripts/codes ainsi qu'un document texte de réponse aux questions. Ce document texte indiquera les résultats importants obtenus lors des manipulations (par exemple, les temps d'exécution de différents algorithmes) ainsi que les commandes/scripts/codes utilisés pour les obtenir (par copie brute ou par référence/chemin). Pour la section 3, ce document prendra la forme d'une notice (synthétique) pour mettre en évidence et utiliser les fonctionnalités de l'outil conçu.

Ne pas oublier d'inclure les noms et prénoms des membres du groupe dans le document texte et dans le nom de l'archive.

L'archive ou le lien vers le repo doivent être envoyés à l'adresse mail *matthieu.bettinger@insa-lyon.fr* avant le vendredi 11/11/2022 23h59.

1 Chiffrement

1.1 Chiffrement symétrique

- Testez les performances de différents algorithmes de chiffrement symétrique disponibles sur votre machine (par exemple avec la commande openssl, cf. A). Pour ce faire, chiffrez des données/fichiers de différentes tailles (1ko, 1Mo, ...) et répétez l'opération pour pallier les perturbations extérieures. Mesurez d'une part le chiffrement et d'autre part le déchiffrement ;
- Comparez ces performances à la robustesse de ces mêmes algorithmes à des attaques de cryptanalyse connues, afin d'en tirer des préférences performance/risque.

1.2 Chiffrement asymétrique

- Tentez les étapes de la section précédente en utilisant de la cryptographie asymétrique. Que constatez-vous ? Proposez une solution permettant d'obtenir une évaluation des performances similaire à celle en 1.1.
- Comparez ces résultats à ceux obtenus en 1.1. Justifiez les pratiques habituelles pour chiffrer des données de taille arbitraire.

2 Fonctions de hachage & Signature

2.1 Fonctions de hachage

- Hashez des fichiers de tailles diverses avec diverses fonctions de hachage à votre disposition. Evaluez les performances et le niveau de sécurité (en force brute) de ces algoritmes. Parmi les algorithmes que vous avez testés, rappelez ceux qui sont dépréciés à l'heure actuelle et pourquoi.
- Vérifiez l'intégrité d'un fichier grâce à un hash de celui-ci. Tentez cette même opération avec un fichier corrompu/modifié. On écrira un script/du code pour automatiser cette opération.

2.2 Signature

- Signez un fichier, puis son hash et évaluez ces deux opérations. Justifiez ainsi la pratique courante pour authentifier un fichier de taille arbitraire.
- Vérifiez l'authenticité d'un fichier grâce à un hash de celui-ci. Tentez cette même opération avec un fichier corrompu/modifié. On écrira un script/du code pour automatiser cette opération.

3 Outil de backup

3.1 Minimum Viable Product

Concevez un outil simple de backup de données. Les règles d'inclusion de données à sauvegarder sont à définir selon vos préférences, par exemple : un fichier spécifique, fichiers dans un répertoire en paramètre, fichier de configuration des chemins à capturer, regex, ... Selon vos affinités, cet outil peut être sous forme de script ou sous un langage de programmation (au choix).

Cet outil doit permettre dans un premier temps de copier des fichiers à sauvegarder vers la zone de backup ainsi que l'opération inverse, c'est-à-dire du backup vers l'espace de travail. Dans les sections suivantes, des propriétés cryptographiques vont être ajoutées de manière itérative.

Prévoyez la possibilité de mesurer et logger le temps d'exécution de l'outil, de bout en bout et par étape du protocole, en vue de la section 3.3.

3.2 Ajout de propriétés cryptographiques

En vous appuyant sur les manipulations et résultats des sections 1 et 2, ajoutez progressivement une protection de l'intégrité, puis de la confidentialité du backup, pour finir par l'authenticité de celui-ci.

Lorsque l'exécution du protocole dévie du comportement sans erreur, l'outil doit signaler le problème en cause (fichier corrompu, signature invalide, ...).

3.3 Performances

Evaluez la latence du backup due aux opérations cryptographiques (dans la mesure du possible, mesurez l'impact de chaque propriété séparément). Estimez ce surcoût de latence par rapport à un backup sans garanties de propriétés cryptographiques.

Mêmes questions concernant l'espace de stockage nécessaire et la facilité de gestion de celui-ci.

Pouvez-vous proposer un protocole dont les performances seraient meilleures ? Si oui, comparez les aspects de performance et de sécurité de la nouvelle proposition par rapport à la(/aux) précédente(s).

3.4 Nouvelles contraintes métier et d'environnement

- Les fichiers ciblés par le backup sont utilisés par une équipe de plusieurs personnes. Chacune d'entre elles peut être amenée à récupérer ce backup et doit être capable de lire les données qu'il contient. Proposez plusieurs solutions (les implémenter est facultatif) et discutez de leurs avantages et inconvénients.
- Vos solutions supportent-elles de nouveaux collaborateurs rejoignant l'équipe et nécessitant l'accès aux données ? Le départ de membres de l'équipe ? Si oui justifiez, sinon proposez de nouveaux mécanismes (pas besoin d'implémentation).
- Pour renforcer la résilience en terme d'intégrité du backup, on veut appliquer de la redondance par code de correction d'erreur aux données à sauvegarder. A quel moment du protocole ce CCE devrait-il être appliqué ? Justifier en quoi d'autres alternatives poseraient problème.

A Commandes OpenSSL

Ci-dessous des paramétrages utiles de la commande openssl (liste non exhaustive, cf. *man openssl* pour plus d'informations ou de variantes) :

- openssl genrsa -out ⟨fichier_rsa.priv⟩ ⟨size⟩ : génère la clé privée RSA de size bits. La valeur de size : 512, 1024, etc.
- openssl rsa -in ⟨fichier_srsa.priv⟩ -des3 -out ⟨fichier.pem⟩ : chiffre la clé privée avec l'algorithme DES3. Vous pouvez utiliser DES, 3DES, ou IDEA, etc.
- openssl rsa -in ⟨fichier_rsa.priv⟩ -pubout -out ⟨fichier_rsa.pub⟩ : stocke la partie publique dans un fichier à part (création de la clé publique associée à la clé privé dans fichier.pem)
- openssl enc ⟨-algo⟩ -in ⟨fichier.txt⟩ -out ⟨fichier.enc⟩ : chiffre fichier.txt avec l'algorithme spécifié (openssl enc –help pour avoir la liste des possibilités ou bien openssl list-cipher-commands) en un fichier.fic.
- openssl enc ⟨-algo⟩ -in ⟨chiffre⟩ -d -out ⟨resultat⟩ : pour le décryptage
- openssl dgst ⟨-algo⟩ -out ⟨sortie⟩ ⟨entree⟩ : pour hacher un fichier. algo est l'algorithme de hachage (sha, sha1, dss1, md2, md4, md5, ripemd160).
- openssl rand -out ⟨clé.key⟩ ⟨numbits⟩ : pour générer un nombre aléatoire sur numbits (utiliser l'option –base 64 pour la lisibilité).
- openssl aes-256-cbc -in ⟨fichier.txt⟩ -out ⟨fichier.enc⟩ -e -k clé.key : pour chiffrer un fichier avec l'algorithme AES utilisant une clé symétrique.
- openssl aes-256-cbc -in ⟨fichier.txt⟩ -out ⟨fichier.enc⟩ -d –k clé.key : pour déchiffrer un fichier avec l'algorithme AES utilisant une clé symétrique.
- openssl rsautl -encrypt -pubin -inkey rsa.pub -in fic.txt -out fic.enc : chiffrer un fichier fic.txt en un fichier fic.enc

- openssl rsautl -decrypt -inkey rsa.priv -in fic.enc -out fic.dec : déchiffrer dans un fichier fic.dec
- openssl rsautl -sign -inkey rsa.priv -in fic.txt -out fic.sig : pour générer une signature fic.sig pour le fichier fic.txt
- openssl rsautl -verify -pubin -inkey rsa.pub -in fic.sig : pour la vérification d'une signature