ESIEE Engineering, année 2008-2009
IN3R21 Systèmes d'exploitation

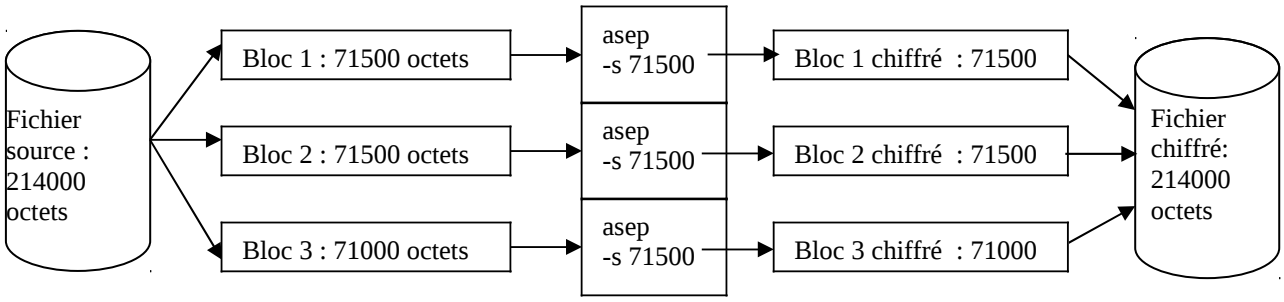
TP n°3
Exécution distribuée via SSH

RAPPEL IMPORTANT : les comptes-rendus (au format Word97 ou texte simple) sont à remettre par e-mail avec accusé de réception à x.hilaire@esiee.fr avant la diffusion par e-mail de la correction de ce TP (cf. échéancier du polycopié distribué en cours).

Le but du TP est d'écrire un script qui permette de paralléliser le chiffrage d'un fichier à l'aide d'un algorithme à clé privée, appelé ASEP – pour un aperçu, appeler `~hilairex/bin/asep` sans arguments.

L'algorithme opère en chiffrant un fichier par blocs, chaque bloc étant long de *s* octets, grandeur paramétrable. Le chiffrage est proprement dit est contrôlé par un nombre indéfini de clés (plus on en donne, plus la sécurité est grande). L'algorithme l'inconvénient d'être consommateur en temps CPU, mais son mode de traitement par blocs le rend « commutatif » par rapport la concaténation : si f1 est un fichier de taille multiple de *s*, et f2 un fichier de taille quelconque, chiffrer séparément f1 et f2, puis concaténer les fichiers chiffrés revient au même que concaténer f1 et f2 d'abord, puis chiffrer le fichier résultant – autrement dit, asep(concact(f1,f2))=concat(asep(f1),asep(f2)). La même remarque vaut pour le déchiffrage.

L'idée du TP consiste à découper un fichier source de N octets en blocs de taille multiple de *s*, sauf peut-être le dernier, et à faire chiffrer en parallèle chacun des blocs par une machine distante. Exemple :



Dans cet exemple, la concaténation des blocs fichiers chiffrés par asep est identique au fichier produit par un appel à `asep –r –s 71500 –i source –o cible`

1) Ecrivez un script `chiffrer` acceptant deux arguments, et effectuant les opérations suivantes :
   a. il vérifie qu'au moins deux arguments lui sont passés en paramètres
   b. il vérifie que le deuxième argument est un entier positif
   c. il vérifie que le premier argument est un nom de fichier valide, récupère sa taille dans une variable, et l'affiche
   d. si les trois conditions précédentes sont vérifiées, il termine avec le code 0. Sinon, il affiche un message d'erreur et termine avec le code 1.

Dans toute la suite, on supposera que les deux premiers arguments passés au script correspondent au nom du fichier à chiffrer, et à la taille des premiers blocs à utiliser.

2) Consultez la page de manuel de la commande `split` : celle-ci découpe un fichier source en plusieurs fichiers d'une taille maximale limitée. Examinez l'effet des options –b et –d, en particulier, sur un fichier de votre choix. Puis modifiez le script précédent pour qu'il découpe le fichier source en autant de fichiers que nécessaire, chaque fichier créé devant correspondre à un bloc à chiffrer.

3) Liez le fichier `~hilairex/bin/asep` vers votre répertoire `~/bin` personnel, ou un répertoire référencé par votre variable d'environnement PATH. Le chiffrage par Asep de l'un des fichiers blocs (fsource) du diagramme ci-dessus en un fichier cible (fcible) à l'aide des clés 1234 et 4321 est obtenu en faisant l'appel suivant :
```
asep –r –i fsource –o fcible –s 7150 1234 4321
```

   Modifiez votre script pour qu'il chiffre séquentiellement tous les fichiers produits par split. Les clés à utiliser pour le chiffrage seront passées en 3ème arguments et suivants du script (vous devrez donc modifier la condition 1a) pour que le test du nombre d'arguments passe de 2 à 3). Les fichiers cryptés devront porter l'extension « .asep », et les fichiers produits par split devront être écrasés. *Indications* : shift, et rm.

4) Ajoutez à votre script les lignes qui permettent : (i) de concaténer les fichiers .asep précédents en un fichier unique de même nom que le fichier source, mais complété par un suffixe .asep, et (ii) de supprimer les fichiers .asep précédents (*Indications* : `cat`, `rm`). Appelez asep directement sur le fichier que vous avez passé au script avec les mêmes options –s et –r pour produire un fichier crypté fictest.asep. Finalement, à l'aide de la commande `diff`, comparez fictest.asep avec le résultat produit par votre script : vous ne devez trouver aucune différence.

5) Modifiez votre script pour que les appels à asep sur les fichiers blocs soient à présent effectués **en parallèle** sur votre machine. Votre script devra prendre en compte l'ensemble des codes de retour des processus asep lancés : si l'un d'eux au moins est non nul, votre script doit afficher un message d'erreur et terminer avec le code 1. Pour tester votre solution, vous pourrez ajouter temporairement l'option (non documentée) –e à vos appels à asep, qui simule une erreur d'exécution en faisant sortir asep immédiatement avec un code d'erreur de 1 et une probabilité de ½. (Rappels : $! reflète la valeur de PID du dernier processus lancé en arrière-plan. De plus, `wait` *`pid`* attend la fin d'un processus fils d'identifiant *`pid`* ; lorsque celui-ci termine, $? reflète le code de sortie qu'il transmet).

6) Si ce n'est déjà fait, appelez `ssh-keygen –t rsa` sur votre machine et entrez des lignes vides à chaque invite. Ceci créera une paire de clés publique/privée `id_rsa.pub` et `id_rsa` dans votre répertoire `~/.ssh`. Quelle ligne de commande vous permet **d'ajouter** votre clé publique au fichier `authorized_keys` du même répertoire, puis d'interdire la lecture de ce dernier fichier aux autres et au groupe ? Lancez cette ligne. Un appel à `ssh` *`machine`* devrait désormais vous permettre d'atteindre par SSH n'importe quelle *`machine`* de la salle depuis votre machine de travail sans avoir à taper votre mot de passe.

7) Commentez temporairement la ligne « `exec bash` » de vos fichiers `.cshrc` ou `.tcshrc` si elle s'y trouve. La commande `ssh` peut également être suivie d'une commande à exécuter : examinez par exemple l'effet de la commande `w` sur votre machine, puis de `ssh` *`machine`* `w` , où *`machine`* est un nom de machine utilisée par un autre binôme. Examinez également la

valeur de $? au sortir d'une commande lancée à distance et qui a échoué (par exemple : `ssh gemini3 ls /berk`). Créez à présent un fichier texte `~/.machines` dans lequel chaque ligne contiendra un nom de machine accessible sur le site ESIEE (par exemple : gemini5, gemini3, pc5008b, pc5103c) et rien d'autre (pas de commentaires, ni d'espaces).

On vous demande de modifier le script précédent pour que le chiffrage du fichier source soit réalisé à l'aide des machines contenues dans votre fichier ~/.machines, la charge devant être répartie de la manière la plus homogène possible. *Indication* : si ssh ne trouve pas vos fichiers, c'est probablement parce que son exécution démarre dans votre HOME… la variable d'environnement PWD peut vous être utile.

8) Bien que parfaitement opérationnelle, la solution proposée ci-dessus est critiquable sur un point : elle requiert la création et le stockage sur disque des blocs de fichiers à la fois dans leurs versions claire et chiffrée (questions 4 et 5). Consultez la page de manuel de la commande `dd` : appelée avec les options `bs`, `count`, `skip`, et `if`, celle-ci lit (sous réserve qu'ils existent) les octets `bs*skip` à `bs*(skip+count)-1` du fichier `if`, et les écrit sur la sortie standard. En observant que `asep` est capable de lire sur l'entrée standard, proposez une solution n'utilisant plus `split`, et ne stockant plus que les fichiers intermédiaires sous forme cryptée. *Conseils* : (i) conservez une copie de votre script dans son état actuel avant de traiter cette question ! (ii) pour créer des fichiers de nom x??.asep , pensez à printf "x %2.2d.asep" valeur.