

Rapport du TP1 IPLS

Thème: Implémentation d'un protocole d'échange sécurisé avec OpenSSL

Réalisé par:

- BOUCHERIR Mohamed-Zineddine
- BOUIDAINE Diaeddin

L'objectif de ce TP est d'implémenter un protocole d'échange sécurisé, basé sur le système de chiffrement hybride, qui consiste à utiliser le chiffrement asymétrique pour transmettre une clé symétrique partagée entre les parties d'échange, et d'utiliser par la suite le chiffrement symétrique pour crypter un message en clair qui va être transmis. Ce protocole va nous garantir la confidentialité, l'intégrité et la non-répudiation.

Initialisation

Au début, on doit générer, pour chaque partie, une clé privée et une autre publique, pour cela, on exécute les commandes **openssl** suivantes qui vont générer pour chaque exécution un fichier contenant la clé correspondante :

- **Pour la source :**

- Génération de la clé privée :

```
# openssl genrsa -out src_rsa.pem -passout pass:Zmbiab123 -des 512
```

- Génération de la clé publique :

```
# openssl rsa -in src_rsa.pem -passin pass:Zmbiab123 -out  
src_rsa_pub.pem -pubout
```

- **Pour la destination :**

- Génération de la clé privée :

```
# openssl genrsa -out des_rsa.pem -passout pass:Diaa1998 -des 512
```

- Génération de la clé publique :

```
# openssl rsa -in des_rsa.pem -passin pass:Diaa1998 -out des_rsa_pub.pem  
-pubout
```

Partage de la clé symétrique :

Dans cette étape, la source commence par crypté la clé symétrique (soit un fichier **secret.txt**) avec la clé publique de la destination générant ce fichier crypté : **secret.crypt**. Le fichier **secret.txt** va être également haché (pour assurer l'intégrité des données), et puis signé avec la clé privé de la source (pour assurer la non-répudiation), ce qui génère le fichier **secret.txt.dgst.sign**, ce dernier va être finalement envoyé avec le fichier **secret.crypt** vers le destinataire. Pour réaliser les opérations précédentes, on exécute les commandes suivantes :

- Crypter la clé symétrique :

```
# openssl rsautl -in secret.txt -out secret.crypt -inkey des_rsa_pub.pem  
-pubin -encrypt
```

- Hacher le crypté :

```
# openssl dgst -md5 -binary -out secret.txt.dgst secret.txt
```

- Signer le haché :

```
# openssl rsautl -in secret.txt.dgst -out secret.txt.dgst.sign -sign -  
inkey src_rsa.pem
```

A la réception de ces 2 fichiers par le destinataire, ce dernier déchiffre le fichier **secret.txt.dgst.sign** avec la clé publique de la source (soit le fichier généré est **dgst1**), et le fichier **secret.crypt** avec sa clé privée, ensuite il recalcule le hachage de ce dernier (soit **dgst**), et enfin il compare les fichier **dgst** et **dgst1** pour vérifier l'égalité. On trouve ci-dessous les commandes exécutées :

- Déchiffrer le signé :

```
# openssl rsautl -in secret.txt.dgst.sign -out dgst1 -pubin -inkey  
src_rsa_pub.pem
```

- Déchiffrer le secret :

```
# openssl rsautl -decrypt -in secret.crypt -out secret_decrypt.txt -  
inkey des_rsa.pem
```

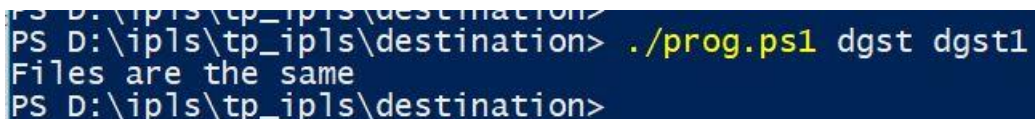
- Recalculer le haché de secret_decrypté :

```
# openssl dgst -md5 -binary -out dgst secret_decrypte.txt
```

- Comparer dgst avec dgst1 :
 - Ce petit program **powershell** (soit **prog.ps1**) compare le contenu de 2 fichiers :

```
$fileA="$args[0]"  
$fileB="$args[1]"  
if(Compare-Object -ReferenceObject $(Get-Content $fileA) -  
DifferenceObject $(Get-Content $fileB))  
{ "Files are different" }  
Else { "Files are the same" }
```

- Résultat de la comparaison :



```
PS D:\ip1s\tp_ip1s\destination> ./prog.ps1 dgst dgst1  
Files are the same  
PS D:\ip1s\tp_ip1s\destination>
```

Envoie de message :

Une fois la clé symétrique est partagée, la connexion donc entre la source et la destination est réalisée, ils peuvent maintenant faire des échanges sécurisés entre-eux.

Quand la source veut envoyer un message, il calcule d'abord son haché puis le signé de cet haché avec sa clé privée, ensuite, il chiffre ce message avec la clé symétrique partagée, et il envoie à la fin le fichier crypté et sa signature, vers le destinataire. Voici les commandes exécutées :

- Hacher le message :

```
# openssl dgst -md5 -binary -out Chapitre2_PKI.pdf.dgst Chapitre_PKI.pdf
```

- Signer le haché :

```
# openssl rsautl -in Chapitre2_PKI.pdf.dgst -out
Chapitre2_PKI.pdf.dgst.sign -sign -inkey src_rsa.pem
```

- Chiffrer le message :

```
# openssl enc -des-cbc -in Chapitre2_PKI.pdf -out Chapitre2_PKI.crypt -
pass file:secret.txt
```

A la réception de message et sa signature, le destinataire déchiffre ce dernier en utilisant la clé partagée, et à la fin, il fait les mêmes étapes que pour vérifier la signature de secret lors de l'envoi de la clé partagée, pour vérifier la signature et l'intégrité de fichier reçu. Voici les commandes exécutées :

- Déchiffrer le signé :

```
# openssl rsautl -in Chapitre2_PKI.pdf.dgst -out hache1_fichier -pubin -
inkey src_rsa_pub.pem
```

- Déchiffrer le fichier :

```
# openssl enc -in Chapitre2_PKI.crypt -out Chapitre2_PKI_decrypte.pdf -
pass file:secret_decrypte.txt -d -des-cbc
```

- Recalculer le haché de secret_decrypté :

```
# openssl dgst -md5 -binary -out hache_fichier Chapitre2_PKI_decrypte.pdf
```

- Comparer hache_fichier avec hache1_fichier :

- Résultat de la comparaison :

```
PS D:\ip1s\tp_ip1s\destination> ./prog.ps1 hache_fichier hache1_fichier
Files are the same
PS D:\ip1s\tp_ip1s\destination>
```