# 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** rsaut1 -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

* Singer le haché :

**# openssl** rsaut1 -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** rsaut1 -in secret.txt.dgst.sign -out dgst1 -pubin -inkey src\_rsa\_pub.pem

* Déchiffrer le secret :

**# openssl** rsaut1 -decrypt -in secret.crypt -out secret\_decrypt.txt -inkey des\_rsa.pem

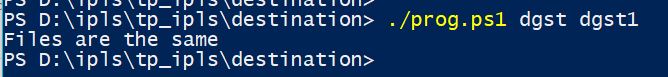
* Recalculer le haché de secret\_decypté :

**# 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 :



# 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** rsaut1 -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 décrypte 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’envoie de la clé partagée, pour vérifier la signature et l’intégrité de fichier reçu. Voici les commandes exécutées :

* Déchiffrer le signé :

**# openssl** rsaut1 -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\_decypté :

**# openssl** dgst -md5 -binary -out hache\_fichier Chapitre2\_PKI\_decrypte.pdf

* Comparer hache\_fichier avec hache1\_fichier :
  + Résultat de la comparaison :

