République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي

ESI ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الإعلام الآلي سابقا)
Ecole nationale Supérieure d'Informatique
ex. INI (Institut National de formation en Informatique)

TP Sécurité des systèmes et réseaux

Rapport

Conception et implémentation d'un protocole d'échange sécurisé avec OpenSSL

Réalisé par :

- MECHARBAT Lotfi Abdelkrim
- AZIZ Rezak

Groupe :

- SIQ3

Année Universitaire : 2020/2021

Partie Conception:

On désire concevoir un protocole d'échange sécurisé. Ce protocole doit garantir les services suivants:

- Confidentialité de l'échange
- Intégrité des données échangés
- Non répudiation de l'émetteur

Pour assurer ces contrainte nous avons opté pour les choix suivant:

- Utiliser le protocole AES256 pour garantir la confidentialité des données
- Utiliser le protocole RSA pour l'échange de la clé de session AES
- Pour assurer l'intégrité on utilise MD5 pour hasher notre message
- Pour la non répudiation on utilise le chiffrement RSA.

Pour utiliser ce protocole donc on suit les étapes suivantes:

- Générer les clés privées et publique pour le destinataires via rsa soit (SKA,PKA) et (SKB,PKB)
- Échanger les clés publiques:
  - A==> B: PKA
  - B==> A: PKB
- Générer la clé de session AES, soit KAB
- Échanger la clé de session:
  - A==> B: {KAB}PKB.{h(KAB)}PKA
  - B déchiffre la clé en utilisant sa clé privée
- Soit le message à envoyer: M.
  - On chiffre le message avec la clé KAB
  - On génère le haché de message crypté en utilisant MD5 et on le signe avec notre clé privée.
  - On stocke le haché dans un lieu sur (Mail par example)
  - A==>B : {M}KAB.{H(m)}SKA
- Réception:
  - Vérifier la signature de hashé avec la clé publique de A
  - Vérifier l'intégrité de message en calculant le hash d message crypté
  - Déchiffrer le message avec la clé symétrique

Implementation:

1. Génération des clés RSA:

```
[root@lxc testOpenSSL]# openssl genrsa -out aziz_rsa_prv.pem -passout pass:chfou
fell@5 -des 512
Generating RSA private key, 512 bit long modulus (2 primes)
.....++++++++++++++++++++++++++
..++++++++++++++++++++++++++
e is 65537 (0x010001)
[root@lxc testOpenSSL]# openssl rsa -in aziz_rsa_prv.pem -passin pass:chfoufell@
5 -out aziz_rsa_pub.pem -pubout
writing RSA key
```

```
meclotfi@meclotfi-Lenovo-V330-15IKB:~/Desktop/Git_Repos/Secure connexion$ openssl genrsa -o
ut lotfi_private.pem -passout pass:x+y=z -des 512
Generating RSA private key, 512 bit long modulus (2 primes)
..++++++++++++++++++++++++++++
...++++++++++++++++++++++++++++
e is 65537 (0x010001)
meclotfi@meclotfi-Lenovo-V330-15IKB:~/Desktop/Git_Repos/Secure connexion$ openssl rsa -in l
otfi_private.pem -passin pass:x+y=z -out lotfi_public.pem -pubout
writing RSA key
```

2. Échange des clés publiques par email
3. Échange de la clé de session:

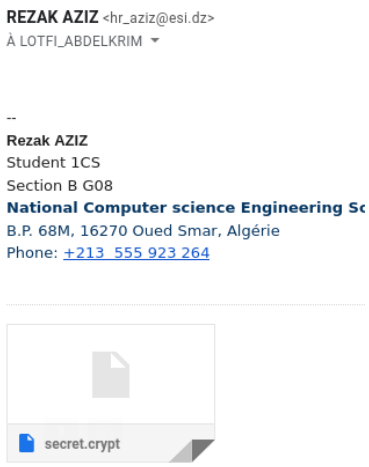
- Générer la clé de session:

```
[root@lxc testOpenSSL]# cat secret.txt
L0tfi_Rez@k_echange_c1e
```

- Crypter la clé de session en utilisant la clé publique de l'autre personne.

```
[root@lxc testOpenSSL]# openssl  rsautl -in secret.txt -out secret.crypt -inkey
lotfi_public.pem -pubin -encrypt
[root@lxc testOpenSSL]# ls
aziz_rsa_prv.pem  aziz_rsa_pub.pem  lotfi_public.pem  secret.crypt  secret.txt
```

- Envoyer la clé:



- Décrypter la clé de session:

```
meclotfi@meclotfi-Lenovo-V330-15IKB:~/Desktop/Git_Repos/Secure connexion$ openssl rsautl -decrypt -in secret.crypt -out secret.txt -inkey lotfi_private.pem
meclotfi@meclotfi-Lenovo-V330-15IKB:~/Desktop/Git_Repos/Secure connexion$ cat secret.txt
L0tfi_Rez@k_echange_c1e
```

A ce moment les deux partie ont la même clé de session

4. Crypter et envoyer un message secret:

On crypte le message avec la clé de session. on calcule le hash de message puis on le signe avec la clé privé

```
meclotfi@meclotfi-Lenovo-V330-15IKB:~/Desktop/Git_Repos/Secure connexion$ cat message.txt
lotfi -(private msg) -> Aziz
meclotfi@meclotfi-Lenovo-V330-15IKB:~/Desktop/Git_Repos/Secure connexion$ openssl enc -aes-
256-cbc -in message.txt -out message.crypt -pass file:secret.txt -pbkdf2
meclotfi@meclotfi-Lenovo-V330-15IKB:~/Desktop/Git_Repos/Secure connexion$ openssl dgst -md5
 -binary -out message.crypt.dgst message.crypt
meclotfi@meclotfi-Lenovo-V330-15IKB:~/Desktop/Git_Repos/Secure connexion$ openssl rsautl -i
n message.crypt.dgst -out message.crypt.dgst.sign -sign -inkey lotfi_private.pem
```

5. Décrypter le message

- Vérification de la signature en décryptant le message avec la clé publique de la source

```
aziz_rsa_prv.pem  lotfi_public.pem  message.crypt.dgst.sign  secret.txt
aziz_rsa_pub.pem  message.crypt     secret.crypt
[root@lxc testOpenSSL]# openssl rsautl -in message.crypt.dgst.sign -out dgst1 -p
ubin -inkey lotfi_public.pem
```

- Verification de l'integrite de message: pour cela on calcule le haché de message reçu (dgst 2) et puis le comparer à dgst 1

```
[root@lxc testOpenSSL]# openssl dgst -md5 -binary -out dgst2 message.crypt
[root@lxc testOpenSSL]# ls
aziz_rsa_prv.pem  dgst1  lotfi_public.pem  message.crypt.dgst.sign  secret.txt
aziz_rsa_pub.pem  dgst2  message.crypt     secret.crypt
[root@lxc testOpenSSL]# cmp dgst1 dgst2
```

- Décrypter le message en utilisant la clé de session

```
[root@lxc testOpenSSL]# openssl enc -in message.crypt -out message.txt -pass fil
e:secret.txt -d -aes-256-cbc -pbkdf2
[root@lxc testOpenSSL]# ls
aziz_rsa_prv.pem  dgst2            message.crypt.dgst.sign  secret.txt
aziz_rsa_pub.pem  lotfi_public.pem  message.txt
dgst1             message.crypt     secret.crypt
[root@lxc testOpenSSL]# cat message.txt
lotfi -(private msg) -> Aziz
```