INFO001 - TP : Cryptographie appliquée : TLS

✏️ Juliette Bois & Bertrand Passieux

Sommaire

1. Mise en place des machines virtuelles

1. 1. Création de la machine A

Cette partie est une partie manipulation où nous avons mis en place nos machines virtuelles.

---

2. Etude du chiffrement RSA

2. 1. Génération des clés RSA

On s'intéresse dans cette première partie à la création de clés asymétriques que l'on utilisera par la suite pour chiffrer et déchiffrer des messages.

Question 1 : En utilisant la page de manuel de la commande openssl, trouver la commande qui permet de générer une « bi-clé » RSA ?

La commande qui permet de générer une « bi-clé » RSA est :

```
openssl genrsa -out rsa_keys.pem 256
```

Cette commande va générer une clé privée sans chiffrement. Nous avons chiffré la clé dans la suite du TP avec 3DES.

Question 2 : Quelle est la longueur des 2 nombres premiers choisis ? Quelle est la longueur du « n » généré ?

Nous affichons le contenu de notre bi-clé à l'aide de la commande :

```
openssl rsa -in rsa_keys.pem -text -noout
```

Voici ce que nous avons :

```
prime1:
  00:fa:24:02:ec:7e:7a:80:92:9b:e1:4a:ac:75:11:23:03
prime2:
  00:e5:b6:4a:95:9f:c5:b0:a7:aa:2f:7f:be:78:3e:11:b1
```

La longueur des 2 nombres premiers choisis est de 17 octets. Sachant qu'on ignore le 00, on obtient 16 octets soit 128 bits. La longueur du "n" généré est de 128 * 128 = 16384 bits.

Question 3 : Que représente le public exponant et le private exponant ? Justifier l'utilisation du terme « exponant ».

On identifie 2 exponants :

—> le public exponant : représente l'exposant de chiffrement

—> le private exponant : représente l'exposant de déchiffrement

On utilise le terme "exponant" car on utilise respectivement ces 2 exposants dans les formules de chiffrement et de déchiffrement en tant qu'exposant.

Question 4 : Y-a-t-il un intérêt à chiffrer la clé publique ? À chiffrer la clé privée ?

La clé publique est la clé qui permet le chiffrement : elle peut être distribuée au public afin de crypter des informations mais seul le propriétaire de la clé privée (qui sert à *dé*chiffrer) peut prendre connaissance des messages au moyen de cette clé. Autrement dit, la clé publique seule ne permet pas de déchiffrer un message. Il n'y a donc pas d'intérêt à la chiffrer.

En revanche, il y a un intérêt à chiffrer la clé privée car elle permet de décrypter les données cryptées.

Question 5 : Pourquoi est-il intéressant de disposer d'un fichier à part contenant uniquement la clé publique ?

Un des rôles de la clé publique est de permettre le chiffrement ; c'est donc cette clé qu'utilisera Bob pour envoyer des messages chiffrés à Alice. C'est pourquoi il est intéressant de disposer d'un fichier à part contenant uniquement la clé publique car cela permet d'y avoir accès et de la diffuser facilement.

---

2. 2. Chiffrement asymétrique

On va utiliser le couple de clés générées précédemment pour chiffrer un document.

Question 6 : Quelle clé doit-on utiliser pour chiffrer un message RSA lorsqu'on veut l'envoyer de manière confidentielle ?

Pour chiffrer un message RSA lorsqu'on veut l'envoyer de manière confidentielle, on doit utiliser la clé publique. C'est donc cette clé qu'utilisera Bob pour envoyer des messages chiffrés à Alice. L'autre clé — la clé privée — sert à *dé*chiffrer. Ainsi, Alice, et elle seule, peut prendre connaissance des messages de Bob, à condition que la brèche ne soit pas trouvée.

Question 7 : Rechercher la sous-commande openssl que vous allez utiliser pour chiffrer en utilisant l'algorithme RSA, attention il ne s'agit pas de la sous-commande enc. Dans la suite cette sous-commande sera notée sc1.

La sous-commande est :

```
openssl rsault
```

Voici la commande entière :

```
openssl rsautl -encrypt -in <fichier_entree> -pubin -inkey <clé> -out <fichier_sortie>
```

En pratique, dans notre cas, on a exécuté cette commande :

```
openssl rsautl -encrypt -in clair.txt -pubin -inkey pub.pem -out essai1.txt
```

Question 8 : Rappeler le rôle d'une fonction de padding et plus précisément le rôle de PKCS.

Une fonction de padding, aussi appelée le remplissage en cryptographie, consiste à formater le message afin que la taille des données soit compatible avec les algorithmes utilisés.

Une fonction de padding est utilisée pour empêcher les attaques KPA (known-plaintext). Les échanges TLS sont formatés (au niveau des headers par exemple), il est donc nécessaire d'ajouter des caractères pour éviter de donner trop d'informations aux attaquants.

PKCS est un standard qui définit des remplissages qui évitent des attaques potentielles dans le cadre de la cryptographie asymétrique.

Question 9 : Quelle est la taille du fichier crypté ? Comparer avec celle du fichier original

La taille du fichier crypté est de 32 octets. Le fichier original fait 7 octets.

Le fichier crypté est plus grand (environ 4,5 x plus) car il fait la taille de la clé utilisée précédemment pour chiffrer le texte clair.

Question 10 : Quelle commande avez-vous saisie pour déchiffrer le fichier?

Pour déchiffrer le fichier, nous avons saisie cette commande :

```
openssl rsautl -decrypt -in <fichier_entree> -inkey <clé> -out <fichier_sortie>
```

En pratique, dans notre cas, on a exécuté cette commande :

```
openssl rsautl -decrypt -in essai1.txt -inkey rsa_keys_cyphered.pem -out decrypt1.txt
```

---

3. Examen du contenu d'un certificat

Avant de s'intéresser à la création d'une PKI, on se familiarise à nouveau avec les certificats et notamment le processus de validation de ces derniers.

Question 11 : On analyse le contenu du certificat du serveur.

Pour visualiser le contenu du certificat du serveur, nous avons saisie cette commande :

```
openssl x509 -in centos.pem -noout -text
```

Quelle partie d'une clé RSA, le certificat contient ?

Le certificat contient la clé publique.

```
Public-Key: (2048 bit)
Modulus:
    00:a3:d7:ee:93:bc:d9:8e:42:88:8a:1e:1e:74:51:
    ec:bc:ef:a6:63:80:b1:8c:89:04:4d:1e:c5:e6:b1:
    fb:e3:53:65:d3:9c:75:98:b2:b0:e5:96:a5:85:6b:
    fa:a9:90:54:09:dc:2a:5b:d6:fc:e9:45:12:06:be:
    d3:16:e3:f9:2c:35:dd:0e:69:e9:64:13:01:35:68:
    7b:54:b5:62:36:52:5d:b1:a7:8c:89:fb:7b:15:4e:
    e8:6f:48:95:99:b5:de:32:da:ea:50:15:74:43:be:
    0e:3e:66:df:8d:88:ff:e6:80:d6:39:ff:19:5b:20:
    f5:e6:f6:e8:9d:fd:83:69:9a:de:71:4d:09:43:f7:
    dc:62:7f:0d:ee:21:f1:c8:c3:42:b8:30:86:d2:c0:
    d8:b6:04:81:8c:3a:e9:0f:33:fe:2d:7b:14:9a:50:
    df:0a:3a:2f:9d:12:ca:db:a1:0d:8a:05:bb:87:65:
    07:9a:fd:65:84:88:50:b1:70:a8:35:bd:02:bc:01:
    5c:2a:31:bb:df:04:cc:c4:37:14:3f:61:b7:91:a1:
    a4:7a:03:33:8f:d6:f2:1a:88:a8:c6:5c:8c:95:18:
    0f:dc:a8:f8:21:94:00:71:fa:4e:40:21:82:1a:4c:
    cb:cb:cd:33:92:a3:52:b7:b2:92:27:a2:e0:cb:28:
    eb:2b
Exponent: 65537 (0x10001)
```

Quels algorithmes ont été utilisés pour signer le certificat ?

L'algorithme utilisé est le ***SHA-256 avec chiffrement RSA***

> *Signature Algorithm: sha256WithRSAEncryption*

Quel attribut contient le nom de la machine pour lequel le certificat a été signé ?

L'attribut qui contient le nom de la machine pour lequel le certificat a été signé est ***subject***.

> *Subject: CN=centos.org*

Quel attribut contient les autres noms de machine pour lesquels le certificat peut être utilisé ?

L'attribut qui contient les autres noms de machine pour lesquels le certificat peut être utilisé est ***Subject Alternative Name***.

> *X509v3 Subject Alternative Name:*

> *DNS:centos.org, DNS:www.centos.org*

Quel est l'utilité du lien pointant vers un fichier .crl ?

Un fichier .crl répertorie les certificats invalidés. L'intérêt est que cette liste est signée par l'autorité de certification pour en empêcher toute modification par une personne non autorisé.

Question 12 : Par qui le certificat du serveur a-t-il été signé ?

Le certificat du serveur a été signé par ***Let's Encrypt***.

Visualiser et/ou télécharger le certificat de la CA qui a signé le certificat du serveur. Par quelle CA ce certificat a-t-il été signé ?

Le certificat du serveur a été signé par **R3**, une CA intermédiaire.

> *issuer=/C=US/O=Let's Encrypt/CN=R3*

Visualiser et/ou télécharger le certificat de cette CA et visualiser son contenu. Quelle est la taille de la clé publique présente dans ce certificat ?

La taille de la clé publique présente dans ce certificat est de 2048 bits soit 256 octets.

Donner les dates de validité de chacun des certificats ? Ces dates sont-elles logiques ?

Certificat 1 (Centos) - Le certificat est valide du 19 octobre 2021 à 07:51:04 au 17 janvier 2022 à 06:51:03 (heure normale d'Europe centrale)

Certificat 2 (Let's Encrypt) - Le certificat est valide du 4 septembre 2020 à 00:00:00 au 15 septembre 2025 à 16:00:00 (heure normale d'Europe centrale)

Certificat 3 (Internet Security Research Group) - Le certificat est valide du 20 janvier 2021 à 19:14:03 au 30 septembre 2024 à 18:14:03 (heure normale d'Europe centrale)

Les dates d'expiration des certificats 2 et 3 ne semblent pas logiques car le certificat ISRG (3) signe le certificat Let's Encrypt (2) alors que le certificat 3 a une date d'expiration inférieure à la date d'expiration du certificat qu'elle signe, en l'occurence le certificat 2.

Question 13 : On s'intéresse au certificat de la CA ayant signé le certificat de la CA qui a signé le certificat du serveur. Comment s'appelle ce type de CA ? Le certificat de cette autorité de certification est-il présent dans le navigateur ? Dans le système d'exploitation CentOS ?

Ce type de CA est appelé une CA intermédiaire

> Subject: C=US, O=Let's Encrypt, CN=R3

Question 14 : Examinez le certificat de www.centos.org directement depuis un navigateur Internet. Vérifier que vous pouvez accéder aux mêmes informations.

Nous n'avons pas les mêmes dates de validité des certificats.

Certificat 1 (Centos) - Le certificat expire le 17 janvier 2022

Certificat 2 (Let's Encrypt) - Le certificat expire le 15 septembre 2025

Certificat 3 (Internet Security research Group) - Le certificat expire le 4 juin 2035

Les dates des certificats 1 et 2 sont cohérentes avec celles relevées à la question 12. En revanche, le certificat 3 n'a pas la même date d'expiration. Mais ici, la date semble plus cohérente car supérieure à la date d'expiration du certificat 2.

---

4. Mise en place d'une PKI

Dans cette partie, nous allons mettre en place une PKI qui va permettre de délivrer des certificats pour tous les serveurs de l'entreprise.

4. 1. Création d'une autorité de certification

Le nom de notre CA est `JbBp`

Noter le chemin du dossier de base de la CA appelé CA_Default : `/etc/pki/CA`

Noter également le sous-dossier qui contiendra les clés privée et publique de la CA : `$dir/private/`

Voici tous les paramètres saisis lors de notre demande de certificat :

```
[root@A-boisju private]# openssl req -new -key cakey.pem -out ca.csr
Country Name (2 letter code) [XX]:fr
State or Province Name (full name) []:Savoie
Locality Name (eg, city) [Default City]:Chambery
Organization Name (eg, company) [Default Company Ltd]:JbBp
Organizational Unit Name (eg, section) []:JbBp Unit
Common Name (eg, your name or your server's hostname) []:JbBp.fr
Email Address []:foo.foo@foo.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Question 15 : Visualiser le contenu de la demande de certificat. Qu'est-ce qu'il manque pour que ce soit un certificat ?

Pour visualiser le contenu de la demande de certificat, nous avons utilisé cette commande :

```
openssl req -text -noout -verify -in ca.csr
```

Pour que ce soit un certificat, il manque la signature d'une autorité de certification. C'est ce que nous faisons avec cette commande :

```
openssl x509 -req -days 365 -in ca.csr -signkey cakey.pem -out ca.crt
```

Cela permet de générer des dates de validité du certificat et le nom de l'autorité de certification (issuer)

Question 16 : Quel est l'algorithme de hachage utilisé ?

L'algorithme utilisé est le ***SHA-256 avec chiffrement RSA***

> *Signature Algorithm: sha256WithRSAEncryption*

Question 17 : Comparer la clé présente dans le certificat et les clés présentes dans cakey.

On a extrait la clé publique de `cakey.pem` au moins de cette commande :

```
openssl rsa -in cakey.pem -pubout -out pub.pem
```

Puis nous avons comparé la clé présente dans le certificat et les clés présentes dans `cakey`.

Pour visualiser la clé publique de `cakey.pem`, nous avons utilisé cette commande :

```
openssl rsa -pubin -in pub.pem -text -noout
```

```
Modulus:
    00:d6:39:f6:47:06:21:3e:92:98:64:44:b3:c9:61:
```

```
    16:f8:11:d9:5e:34:bd:14:4b:1f:2b:a0:e7:f8:11:
    fa:0d:ec:17:6b:cb:2a:f6:68:c4:9e:f8:a6:cf:74:
    d3:bc:a3:fc:2d:d3:33:59:12:af:92:2d:26:07:c3:
    7b:04:4f:1a:9f:0d:d3:8a:66:bd:69:d9:a1:b9:92:
    9e:65:2b:03:ca:2c:c9:1b:15:de:21:15:b3:a8:7f:
    28:e5:e4:0f:d9:70:35:dc:5c:cc:0f:0d:22:57:df:
    c6:47:32:c7:97:af:2b:74:e4:58:69:86:06:5d:1e:
    08:fc:64:b3:58:60:cd:08:f0:21:f0:f6:29:48:de:
    b8:6d:89:2c:63:5f:a4:9b:61:c3:5e:03:8d:5c:7f:
    8f:5d:21:cc:d5:09:81:45:30:e3:c4:76:77:53:21:
    55:83:86:cb:b9:3e:1e:91:c5:eb:36:d5:41:6f:01:
    d6:03:d2:57:1e:28:9b:fb:63:ad:72:a7:be:bd:29:
    80:86:11:cd:d7:45:79:2d:16:f2:46:e8:79:5f:63:
    dc:e1:e7:97:4c:e4:d3:72:d8:38:0a:d8:55:67:78:
    9e:e1:51:8e:00:ca:4b:93:e8:9e:cc:52:83:7d:4e:
    e6:f5:4a:82:a2:7a:ae:1a:ec:0d:c2:f2:21:a0:0a:
    2f:1b
```

Pour visualiser la clé publique de `ca.cry`, nous avons utilisé cette commande :

```
openssl x509 -in ca.crt -noout -text
```

```
Modulus:
    00:d6:39:f6:47:06:21:3e:92:98:64:44:b3:c9:61:
    16:f8:11:d9:5e:34:bd:14:4b:1f:2b:a0:e7:f8:11:
    fa:0d:ec:17:6b:cb:2a:f6:68:c4:9e:f8:a6:cf:74:
    d3:bc:a3:fc:2d:d3:33:59:12:af:92:2d:26:07:c3:
    7b:04:4f:1a:9f:0d:d3:8a:66:bd:69:d9:a1:b9:92:
    9e:65:2b:03:ca:2c:c9:1b:15:de:21:15:b3:a8:7f:
    28:e5:e4:0f:d9:70:35:dc:5c:cc:0f:0d:22:57:df:
    c6:47:32:c7:97:af:2b:74:e4:58:69:86:06:5d:1e:
    08:fc:64:b3:58:60:cd:08:f0:21:f0:f6:29:48:de:
    b8:6d:89:2c:63:5f:a4:9b:61:c3:5e:03:8d:5c:7f:
    8f:5d:21:cc:d5:09:81:45:30:e3:c4:76:77:53:21:
    55:83:86:cb:b9:3e:1e:91:c5:eb:36:d5:41:6f:01:
    d6:03:d2:57:1e:28:9b:fb:63:ad:72:a7:be:bd:29:
    80:86:11:cd:d7:45:79:2d:16:f2:46:e8:79:5f:63:
    dc:e1:e7:97:4c:e4:d3:72:d8:38:0a:d8:55:67:78:
    9e:e1:51:8e:00:ca:4b:93:e8:9e:cc:52:83:7d:4e:
    e6:f5:4a:82:a2:7a:ae:1a:ec:0d:c2:f2:21:a0:0a:
    2f:1b
```

Nous pouvons observer que les 2 clés publiques sont identiques.

---

4. 2. Création du certificat du serveur

On passe donc maintenant à la génération d'un certificat pour un serveur http.

Nous avons tout d'abord généré une paire de clé pour le serveur, au moyen de cette commande :

```
openssl genrsa -des3 -out rsa_server_key.pem 2048
```

Sans mot de passe : `openssl rsa -in [original.key] -out [new.key]`

Puis nous avons fait une demande de certificat :

```
openssl req -new -key rsa_server_key.pem -out ca_server.csr
```

Enfin nous avons créer le certificat signé par la CA en utilisant la commande :

```
openssl ca -in ca_server.csr -out ca_server.crt
```

4.3 Mise en place d’un serveur Web sécurisé

On va maintenant installer et configurer le serveur Apache. On veillera à ce qu'il utilise le certificat créé précédemment.

Tout d'abord, nous avons installer le serveur Apache ainsi que le module ssl pour Apache.

Nous avons ensuite démarré le service : `systemctl start httpd`

Après avoir vérifié que le service est démarré : `systemctl status httpd`, nous avons pu tester en local : `wget https://127.0.0.1`

Question 18 : Quels avertissements sont affichés par le client ? Le client doit en principe signaler 2 problèmes ?

Le client signale 2 problèmes :

1. Erreur de vérification du certificat serveur :

> AVERTISSEMENT : impossible de vérifier l'attribut 127.0.0.1 du certificat, émis par «/C=FR/ST=Savoie/L=Chambery/O=JbBp/OU=JbBp Unit/CN=jbbp.fr/emailAddress=juliette.bois@etu.univ-savoie.fr» : Impossible de vérifier localement le certificat autorité de l'émetteur.

2. Erreur de nom lié au certificat :

> AVERTISSEMENT : le nom commun du certificat «www.binome37.fr» ne concorde pas avec le nom de l'hôte demandé «127.0.0.1».

Nous souhaitons tout d'abord régler le problème du nom de la machine. Il faut que le CN présent dans le certificat corresponde au nom DNS de la machine.

Question 19 : Quelles manipulations avez-vous réalisées ?

Nous avons modifié le fichier `hosts` de notre machine pour ajouter le CN aux DNS.

Nous souhaitons maintenant régler le deuxième message d’erreur : « Le certificat n’est pas délivré par une autorité de de confiance » en important le certificat de la CA dans le magasin de certificat de la machine ou dans le magasin de certificat du navigateur.

Question 20 : Quelles manipulations avez-vous réalisées ?

Nous avons ajouté le certificat de notre CA au certificat trusted

```
[root@A-boisju private]# cp ca.crt ../../ca-trust/source/anchors/ca.crt
```

Puis nous avons mis à jour les certificats trusted

```
[root@A-boisju private]# update-ca-trust
```

---

5. Développement de logiciels client et serveur sécurisé par TLS

Pour cette partie, nous avons fait le choix du langage Python. L'objectif, ici, n'étant pas de s'intéresser de manière approfondie aux mécanismes pour une communication réseau entre deux applications, nous sommes partie sur un fonctionnement très simple : l'application cliente se connecte au serveur et on peut envoyer des messages textuels entre le serveur et le client. L'objectif est de sécuriser la communication entre ces deux applications en utilisant TLS. Le serveur devra fournir un certificat que le client devra valider complètement. La communication entre le client et le serveur sera chiffrée par TLS.

Le code source ainsi que les commentaires sont disponibles sur ce repository GitHub : https://github.com/juliette-bois/tp-securite-tls

Pour lancer le projet, à la racine du projet `tp-securite-tls`, lancer le serveur

```
python3 server.py
```

En parallèle, lancez 2 clients :

```
python3 client.py
```

Et voilà, les 2 clients communiquent !

> 💡 Assurez-vous également que la CA est bien trusted par votre machine et que le CN est ajouté à votre fichier hosts de votre machine.

Nous avons sauvegardé les clés et les certificats du serveur et de la CA.

```
scp [autres options] [nom d'utilisateur source@IP]:/[dossier et nom de fichier] [nom d'utilisateur de destination@IP]:/[dossier de des
```

Dans la pratique,

```
scp root@192.168.141.250:/../etc/pki/CA/ca.crt /Users/juliette/tp-securite-tls
scp root@192.168.141.250:/../etc/pki/tls/private/ca_server.crt /Users/juliette/tp-securite-tls
scp root@192.168.141.250:/../etc/pki/tls/private/rsa_server_key.pem /Users/juliette/tp-securite-tls
```