

Travaux pratiques Benchmark cryptographique par openssl

Objectif du TP :

- Evaluer les performances des algorithmes cryptographiques
- Evaluer le niveau de sécurité d'un serveur sécurisé selon les recommandations de l'ANSSI

Exercice1 : Evaluation des algorithmes cryptographiques

L'objectif de cet exercice est de répondre aux questions suivantes :

- Quel algorithme de signature est plus rapide en signature et vérification ECDSA ou RSA ?
- Quels algorithmes de chiffrement est plus rapide parmi les algorithmes symétriques suivants : RC4, DES ou AES.
- Quelles fonctions de hachage est plus rapide ?

Dans cet exercice, on va se baser sur la commande `openssl speed` pour l'évaluation des algorithmes de chiffrement symétriques, asymétriques et hachage.

Pour les algorithmes de signature, remplissez le tableau suivant :

Algorithme	Signature/sec	Vérification/sec
RSA 512		
RSA 1024		
RSA 2048		
ECDSA 160		
ECDSA 224		
ECDSA 384		
ECDSA 512		

Pour remplir le tableau ci-dessous, lancez les commandes suivantes : `openssl speed rsa` et `openssl speed ecdsa` (attendre la fin de l'exécution de chaque commande). Comparez les performances des deux algorithmes RSA 2048 et ECDSA 256 en termes de vérification et signature. Conclure.

Pour les algorithmes de chiffrement symétrique, remplissez le tableau suivant :

Algorithme	Temps de chiffrement	Taille du bloc
AES-CBC-128		64 bytes
AES-CBC-192		64 bytes
AES-CBC-256		64 bytes

AES-CBC-128		16384 bytes
AES-CBC-192		16384 bytes
AES-CBC-256		16384 bytes
RC4		64 bytes
RC4		16384 bytes
DES CBC		64 bytes
DES CBC		16384 bytes

1. A quoi est due la différence de temps de chiffrement entre AES-CBC-128, AES-CBC-192 et AES-CBC-256.
2. Classez les algorithmes ci-dessus (64 bytes) selon le temps de chiffrement.

Pour les algorithmes de hachage, remplissez le tableau suivant :

Algorithme	Fonction de hachage	Taille de bloc
MD5		16 bytes
MD5		16384 bytes
SHA256		16 bytes
SHA256		16384 bytes

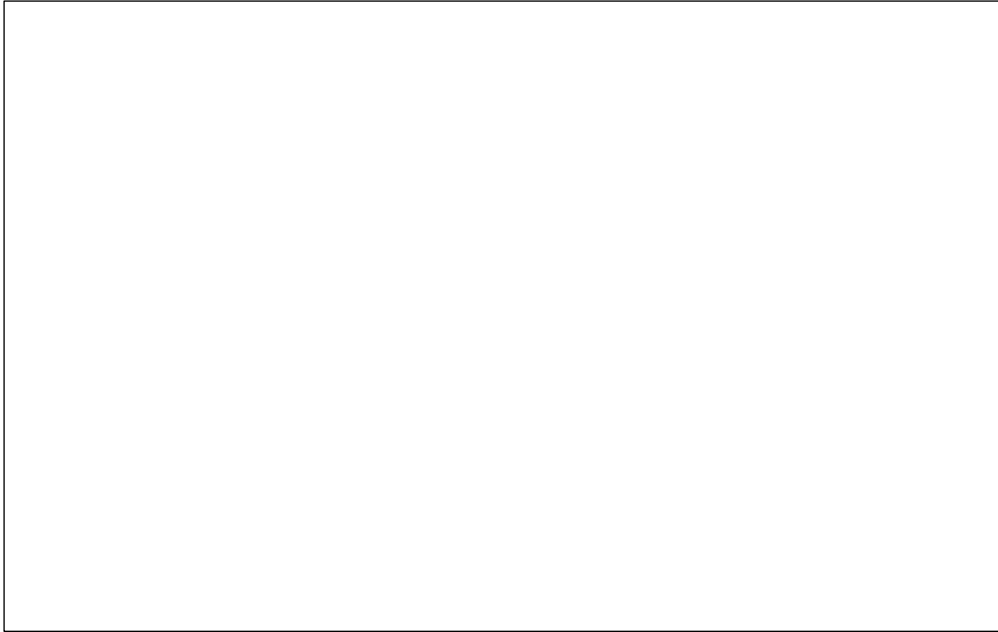
1. A quoi est due la différence de temps de hachage entre 16 bytes et 16384 pour la taille des blocs ?
2. Quelle fonction est plus rapide ? la plus sûre ?

Exercice2 : Recommandations de l'ANSSI

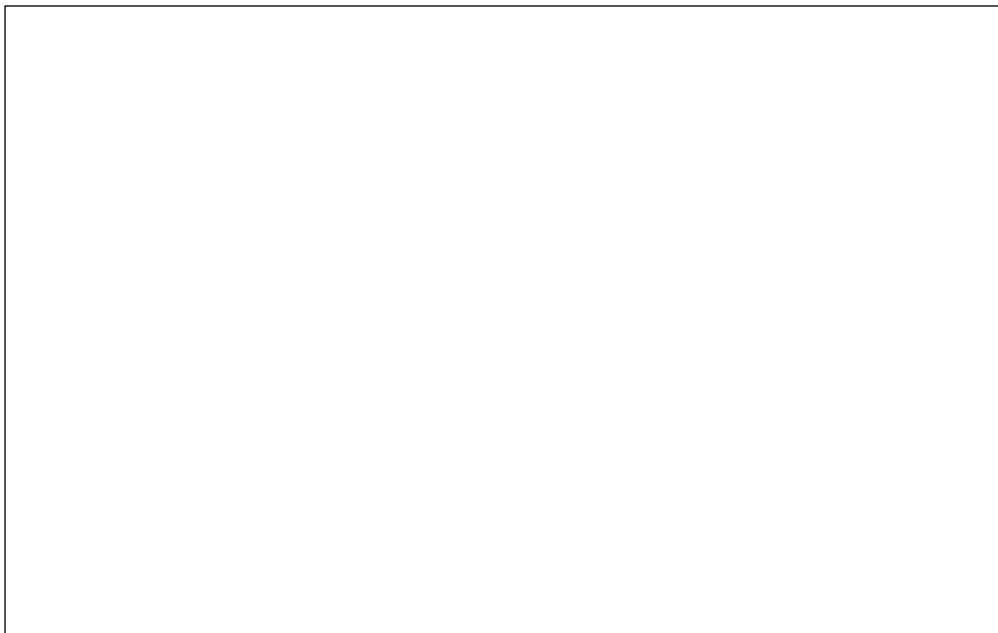
Dans cet exercice, vous allez chercher les recommandations de l'ANSSI par rapport aux chiffrements recommandés dans l'implémentation du protocole TLS. On se base dans cet exercice sur les recommandations de l'ANSSI publiées dans le document dont le lien est le suivant :

https://www.ssi.gouv.fr/uploads/2017/07/anssi-guide-recommandations_de_securite_relatives_a_tls-v1.2.pdf

1. Quels sont les suites TLS 1.2 recommandées avec un serveur disposant d'un certificat avec clé publique RSA ?



2. Quels sont les suites TLS 1.2 recommandées avec un serveur disposant d'un certificat avec clé publique ECDSA ?



3. Quelles sont les courbes elliptiques recommandées pour les échanges de clés ECDHE ?



4. L'échange de clés pourrait se faire en ECDHE ou RSA dans le cadre du TLS. Quelle est la recommandation de l'ANSSI à ce propos ? Justifiez votre réponse.

5. Quelles sont les recommandations de l'ANSSI par rapport aux modes opératoires des algorithmes de chiffrement symétriques ? citez les modes recommandés.

6. Dans le TLS1.3, qu'est-ce que signifie le paramètre *renegotiation_info* ? Quel est son avantage ?

7. Expliquez brièvement pourquoi est-il recommandé d'utiliser le PFS.

8. Expliquez brièvement la nouvelle fonctionnalité de TLS1.3 qui s'appelle 0-RTT.

Exercice 3 : Test de sécurité d'un serveur sécurisé

Dans cet exercice, vous allez tester la configuration d'un serveur web via le script ***TestSSL.sh*** qui est un script en ligne de commande. Il s'agit d'un script libre et open source et permet de tester les protocoles, les ciphers supportés, et remonte des infos sur la robustesse des PFS (Perfect Forward Secrecy) ainsi que certaines failles cryptographiques connues. Pour installer ***TestSSL.sh***, ouvrez un terminal et entrez la commande suivante :

```
root# git clone --depth 1 https://github.com/drwetter/testssl.sh.git
root# cd testssl.sh
root# ./testssl.sh https://myserver.com
```

Les principales options de la commande `./testssl.sh` sont :

-V, --local	pretty print all local ciphers
<URI>	host[host:port URL URL:port]
--mode <serial parallel>	
-e, --each-cipher	checks each local cipher remotely
-E, --cipher-per-protocol	checks those per protocol
-s, --std, --standard	tests certain lists of cipher suites by strength
-p, --protocols	checks TLS/SSL protocols (including SPDY/HTTP2)
-S, --server-defaults	displays the server's default picks and certificate info
-P, --server-preference	displays the server's picks: protocol+cipher
-c, --client-simulation	test client simulations, see which client negotiates with cipher and protocol
-U, --vulnerable	tests all (of the following) vulnerabilities (if applicable)
-H, --heartbleed	tests for Heartbleed vulnerability
-I, --ccs, --ccs-injection	tests for CCS injection vulnerability
-T, --ticketbleed	tests for Ticketbleed vulnerability in BigIP loadbalancers
-BB, --robot	tests for Return of Bleichenbacher's Oracle Threat (ROBOT) vulnerability
-R, --renegotiation	tests for renegotiation vulnerabilities
-C, --compression, --crime	tests for CRIME vulnerability (TLS compression issue)
-B, --breach	tests for BREACH vulnerability (HTTP compression issue)
-O, --poodle	tests for POODLE (SSL) vulnerability
-Z, --tls-fallback	checks TLS_FALLBACK_SCSV mitigation
-W, --sweet32	tests 64 bit block ciphers (3DES, RC2 and IDEA): SWEET32 vulnerability
-A, --beast	tests for BEAST vulnerability
-L, --lucky13	tests for LUCKY13
-F, --freak	tests for FREAK vulnerability
-J, --logjam	tests for LOGJAM vulnerability
-D, --drown	tests for DROWN vulnerability
-f, --pfs, --fs, --nsa	checks (perfect) forward secrecy settings
-4, --rc4, --appelbaum	which RC4 ciphers are being offered?

-g, --grease	tests several server implementation bugs like GREASE and size limitations
--fast cipher.	omits some checks: using openssl for all ciphers (-e), show only first preferred cipher.

Voici quelques exemples :

Test Everything on a Single Host and Output to console

```
./testssl.sh -e -E -f -p -y -Y -S -P -c -H -U TARGET-HOST
```

Test Everything on a Single Host and Output to HTML

```
./testssl.sh -e -E -f -p -y -Y -S -P -c -H -U TARGET-HOST | aha > OUTPUT-FILE.html
```

Test all hosts on a Subnet and Output to HTML

```
./testssl.sh -e -E -f -p -y -Y -S -P -c -H -U 192.168.1.0/24 | aha > OUTPUT-FILE.html
```

Same as above, but only enumerate each server supported ciphers:

```
./testssl.sh -E 192.168.1.0/24 | aha > OUTPUT-FILE.html
```

Test for RC4 list per protocol

```
testssl.sh -E TARGET | grep -iE "RC4|Hexcode|SSLv3|TLS 1"
```

Test for CRIME

```
testssl.sh -C TARGET
```

Test for BREACH

```
testssl.sh -T TARGET
```

Test for client initiated secure renegotiation

```
testssl.sh -R TARGET
```

Identify weak ciphers using testssl.sh:

```
./testssl.sh --ciphers TARGET
```

Testing for Heartbleed using testssl.sh:

```
./testssl.sh -B TARGET
```

Testing for POODLE using testssl.sh

```
./testssl.sh -O TARGET
```

Automated Testing for CCS Injection

```
./testssl.sh -I TARGET
```

Testing for BREACH using testssl.sh

```
./testssl.sh -T TARGET
```

Vous allez maintenant tester la sécurité d'un serveur. Allez sur le site : <https://www.ssllabs.com/ssltest/> et choisissez l'un des serveurs récemment testés dans la rubrique *recent worst*.

1. Expliquez brièvement la vulnérabilité *heartbleed* (CVE-2014-0160) et testez si le serveur est vulnérable à *Heartbleed*.

2. Expliquez brièvement la vulnérabilité *BREACH* (CVE-2013-3587) et testez si le serveur est vulnérable à *BREACH*

3. Expliquez brièvement la vulnérabilité *POODLE* (CVE-2014-3566) et testez si le serveur est vulnérable à *POODLE*.

4. Expliquez brièvement la vulnérabilité LOGJAM (CVE-201(-4000) et testez si le serveur est vulnérable à LOGJAM