# Résumé du cours – Application avec OpenSSL

**J. Briffaut** – Equipe SDS

ENSI de Bourges
LIFO Université d'Orléans
jeremy.briffaut@ensi-bourges.fr

# Outline

# Sommaire

# Chiffrement symétrique

- Algorithme symétrique ou à clef secrète
  - Plus rapides donc préférés pour le chiffrement de données
  - 1 seul clé pour le chiffrement/déchiffrement
    - Clé partagée
    - Elle doit resté secrète
  - Algorithmes basés sur une combinaison de transposition/substitution alphabétique
    - Diffusion de la confusion syntaxique et alphabétique sur l'ensemble du message
- Deux familles d'algorithmes :
  - chiffrement par flux
  - chiffrement par blocs :
    - mode CBC : Cipher Block Chaining
    - mode ECB : Electronic Code Book

# Chiffrement symétrique

- Algorithme symétrique ou à clef secrète
  - Plus rapides donc préférés pour le chiffrement de données
  - 1 seul clé pour le chiffrement/déchiffrement
    - Clé partagée
    - Elle doit resté secrète
  - Algorithmes basés sur une combinaison de transposition/substitution alphabétique
    - Diffusion de la confusion syntaxique et alphabétique sur l'ensemble du message
- Deux familles d'algorithmes :
  - chiffrement par flux
  - chiffrement par blocs :
    - mode CBC : Cipher Block Chaining
    - mode ECB : Electronic Code Book

# Chiffrement symétrique - exemple d'algorithme

- Chiffrement par blocs
  - **DES (Data Encryption Standard)** : 64 bits de clé donc 56 utiles (controle de parité)
  - **Triple DES** : 112 bits de clé, basé sur DES
  - **AES (Advance Encryption Standard)** : clé de 128,192 ou 256 bits
  -
- Chiffrement par flux
  - **RC4 (Rivest Cypher)** : clé entre 40 et 256 bits
  - **A5/1** : clé de 64 bits

# Chiffrement symétrique - OpenSSL

- Lister les algorithmes de chiffrements symètrique

```
openssl list -cipher-commands
```

- Chiffrement

```
# encrypt file.txt to file.enc using 256-bit AES in CBC mode
openssl enc -aes-256-cbc -salt -in file.txt -out file.enc

# the same, only the output is base64 encoded for, e.g., e-mail
openssl enc -aes-256-cbc -a -salt -in file.txt -out file.enc
```

- Déchiffrement

```
# decrypt binary file.enc
openssl enc -d -aes-256-cbc -in file.enc

# decrypt base64-encoded version
openssl enc -d -aes-256-cbc -a -in file.enc
```

# Chiffrement asymétrique

- Algorithme asymétrique ou à clé publique
  - Plus lents donc préférés pour l'authentification
  - 1 **paire de clés** :
    - Clé privé : à conserver préciseusement
    - Clé public : à rendre public (PKI)
  - Algorithmes basés sur des problèmes difficiles à résoudre : (logarithme discret, factorisation de grands nombres)

## Pour le chiffrement

- Clé privé : utilisée pour le déchiffrement
- Clé public : utilisée pour le chiffrement

# Chiffrement asymétrique

- Algorithme asymétrique ou à clé publique
  - Plus lents donc préférés pour l'authentification
  - 1 **paire de clés** :
    - Clé privé : à conserver préciseusement
    - Clé public : à rendre public (PKI)
  - Algorithmes basés sur des problèmes difficiles à résoudre : (logarithme discret, factorisation de grands nombres)

## Pour le chiffrement

- Clé privé : utilisée pour le déchiffrement
- Clé public : utilisée pour le chiffrement

# Chiffrement asymétrique - exemples d'algorithmes

- **RSA (Ron Rivest,Adi Shamir et Leonard Adleman)** :
  - Basé sur la factorisation des grands nombres
  - Longueur des clés : 1024, 2048 ou 4096 bits
  - Encoder le message en suivant l'Optimal Asymmetric Encryption Padding (OAEP)
- **ElGamal** :
  - Basé sur les logarithmes discrets
  - Longueur des clés : 1024, 2048 ou 4096 bits
  - Utilisé dans DSA (Digital Signature Algorithm)

# Chiffrement asymétrique - OpenSSL

- Génération de **clé privée** RSA

```
# default 512-bit key, sent to standard output
openssl genrsa

# 1024-bit key, saved to file named mykey.pem
openssl genrsa -out mykey.pem 1024

# same as above, but encrypted with a passphrase
openssl genrsa -des3 -out mykey.pem 1024
```

- Génération de **clé publique** RSA

```
openssl rsa -in mykey.pem -pubout
```

# Chiffrement asymétrique - OpenSSL

- Génération de **clé privée** DSA

```
#On génèrera des paramètres DSA dans un fichier paramdsa.pem par
    la commande (par exemple avec un clé de taille 1024) :
openssl dsaparam 1024 -out dsaparam.pem

#Pour générer une clé privée à partir de paramètres DSA dans un
    fichier dsa.priv :
openssl gendsa dsaparam.pem -out dsa.priv
```

- Génération de **clé publique** DSA

```
#Puis extraire la clé publique DSA dans un fichier dsa.pub à partir
    de la clé privée :
openssl dsa -in dsa.priv -pubout -out dsa.pub
```

# Chiffrement asymétrique - OpenSSL

- Chiffrement avec RSA

```
openssl rsautl -encrypt -in <fichier_entree> -pubin <cle public> -
    out <fichier_sortie>
```

- Dechiffrement avec RSA

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

# Sommaire

# Hachage

- Fonction de hachage :
  - Fonction à sens unique
    - Facile à calculer mais difficile à inverser
    - Il est difficile de trouver deux messages ayant la même empreinte
  - Convertit une chaîne de longueur quelconque en une chaîne de taille inférieure et généralement fixe
    - empreinte, condensé, somme ou hash

**Utilisation**

**Garantir l'intégrité d'un message Stockage de mots-de-passe**

# Hachage - exemple d'algorithme

- MD5 (Message Digest 5)
  - Empreinte de 128 bits
- SHA (Secure Hash Algorithm)
  - SHA-1 : empreinte de 160 bits
  - SHA-2 : SHA-224 SHA-256 SHA-384 SHA-512
  - SHA-3 : en cours de développement
- Whirlpool
  - Dérivé de AES
  - Empreinte de 512 bits

# Hachage - OpenSSL

- Calcul et vérification d'une empreinte

```
# MD5 digest
openssl dgst -md5 filename
#1f5d374b63c41ac3ad3e4a0c93511828

# SHA1 digest
openssl dgst -sha1 filename
#b2c30b8c870f2140042befe3808ed2ae5eac71e3

# SHA512 digest
openssl dgst -sha512 filename
#61f4937d0b7f78379d75402f5926bf170278ed5d9d8db95eee009be
#026fed419387840fb2bca6993cf540ed2f6998a457213
#3b2d89be2f36a66f764c58e97ae4
```

# Sommaire

# Signature symétrique

- Code d'authentification des messages
  - Aussi appelé **scellement**
  - Utilise la cryptographie à clés secrètes
  - Dernier bloc du cryptogramme obtenu avec un algorithme de chiffrement en mode CBC
  - Obtention d'un **Seau** ou d'un **MAC**

## Propriétés

- Authentification de l'origine des données
- Intégrité

# Signature asymétrique

- Signature en utilisant la cryptographie asymétrique
  - Approche 1 :
    - Chiffrer le message avec la clé privée
    - Vérifier la signature avec la clé publique

    => Problème : lenteur du chiffrement asymétrique
  - Approche 2 :
    - Calculer un résumé d'un message à l'aide d'un fonction de hachage
    - Chiffrement du résumé avec la clé privée
    - Vérification en déchiffrant la signature avec la clé publique et vérifiant le résumé

## Propriétés

- Authentification de l'origine des données
- Intégrité
- Non-répudiation de la source

# Signature asymétrique

- Signature en utilisant la cryptographie asymétrique
  - Approche 1 :
    - Chiffrer le message avec la clé privée
    - Vérifier la signature avec la clé publique

    => Problème : lenteur du chiffrement asymétrique
  - Approche 2 :
    - Calculer un résumé d'un message à l'aide d'un fonction de hachage
    - Chiffrement du résumé avec la clé privée
    - Vérification en déchiffrant la signature avec la clé publique et vérifiant le résumé

## Propriétés

- Authentification de l'origine des données
- Intégrité
- Non-répudiation de la source

# Signature - exemples d'algorithmes

- Scellement (Symétrique)
  - DES-MAC
- Asymétrique
  - MD5 + RSA
  - SHA-1 + RSA
  - SHA-1 + El-Gamal

# Signature - OpenSSL

- Calcul d'une signature asymétrique (RSA-MD5)

```
#Pour générer une signature fic.sig pour le fichier fic.txt :
openssl rsautl -sign -inkey rsa.priv -in fic.txt -out fic.sig
```

- Vérification d'une signature asymétrique (RSA-MD5)

```
#Que l'on déchiffrera par :
openssl rsautl -verify -pubin -inkey rsa.pub -in fic.sig
```

# Signature - OpenSSL

- Calcul d'une signature asymétrique (RSA-SHA1)

```
#signed digest will be foo-1.23.tar.gz.sha1
openssl dgst -sha1 \
  -sign mykey.pem
  -out foo-1.23.tar.gz.sha1 \
  foo-1.23.tar.gz
```

- Vérification d'une signature asymétrique (RSA-SHA1)

```
# to verify foo-1.23.tar.gz using foo-1.23.tar.gz.sha1
# and pubkey.pem
openssl dgst -sha1 \
  -verify pubkey.pem \
  -signature foo-1.23.tar.gz.sha1 \
  foo-1.23.tar.gz
```

# Sommaire

# Certificat et clé privée

- Génération d'une clé privée et d'une requête de certification (CSR)

=> CSR à envoyer à un CA commercial ou à auto-signer

```
openssl req -out MYCSR.csr -pubkey -new -keyout MYKEY.key
# ajouter -nodes pour générer un clé privée non chiffrée
# ajouter -config <openssl.conf> pour spécifier le fichier de configuration
```

- Déchiffrer une clé privée

```
openssl rsa -in MYKEY.key >> MYKEY-NOCRYPT.key
```

# Certificat auto-signé

- Certificat auto-signé : CA privé

```
openssl req -x509 -new -out MYCERT.crt -keyout MYKEY.key -days
    365
```

# Certificat : signature

- Générer une requête de certification (CSR)

```
openssl req -out MYCSR.csr -key MYKEY.key -new
```

- Signer la requête de certification (CSR)

```
openssl x509 -req -in MYCSR.csr -CA MY-CA-CERT.crt -CAkey
    MY-CA-KEY.key -CAcreateserial -out MYCERT.crt -days 365
```

# Certificat : signature

- Générer une requête de certification (CSR)

```
openssl req -out MYCSR.csr -key MYKEY.key -new
```

- Signer la requête de certification (CSR)

```
openssl x509 -req -in MYCSR.csr -CA MY-CA-CERT.crt -CAkey
    MY-CA-KEY.key -CAcreateserial -out MYCERT.crt -days 365
```

# Convertion de formats PER<->DEM

- Convertir DER (.crt .cer .der) en PEM

```
openssl x509 -inform der -in MYCERT.cer -out MYCERT.pem
```

- Convertir PEM en DER

```
openssl x509 -outform der -in MYCERT.pem -out MYCERT.der
```

# Convertion de formats PKCS#12

- Convertion de PKCS#12 (.pfx .p12) en PEM

```
openssl pkcs12 -in KEYSTORE.pfx -out KEYSTORE.pem -nodes
# -nocerts : uniquement la clé privée
# -nokeys : uniquement le certificat
```

- Convertion DER et clé en PKCS#12

```
openssl pkcs12 -export -in MYCERT.crt -inkey MYKEY.key -out
      KEYSTORE.p12 -name "tomcat"
```

# Vérifications

- Vérifier une clé privée

```
openssl rsa -in MYKEY.key -check
```

- Vérifier un CSR

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

- Vérifier un certificat

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

- Vérifier une chaine de certification

```
openssl pkcs12 -info -in KEYSTORE.p12
```

# Sommaire

# Benchmark OpenSSL

- Machines de test :
  - Macbook pro Core2 2.53Ghz / 4go RAM / MACOS X 10.6
  - Mac Mini Core2 2.26 / 4go RAM / Gentoo
  - Serveur Xeon 16 core 2.40Ghz / 18go RAM / CentOS 5.5
- Commande : openssl speed -multi $NB_{CORE}ALGO$

| **type** | 16 bytes | 64 bytes | 256 bytes | 1024 bytes | 8192 bytes |
|---|---|---|---|---|---|
| md5 | 50153.96k | 155053.64k | 403903.12k | 679954.05k | 797494.30k |
| md5 | 56283.03k | 179647.02k | 414182.23k | 620706.47k | 725224.11k |
| md5 | 268295.93k | 911316.69k | 2498006.54k | 4416352.31k | 5658389.92k |
| | | | | | |
| des cbc | 89593.76k | 94300.19k | 95934.86k | 96675.91k | 96709.42k |
| des cbc | 92497.28k | 95709.57k | 98055.59k | 96058.03k | 98388.65k |
| des cbc | 495923.47k | 512591.65k | 518225.21k | 519300.39k | 519704.74k |
| | | | | | |
| aes−256 cbc | 166734.96k | 174709.33k | 181725.36k | 183158.91k | 183129.60k |
| aes−256 cbc | 89091.87k | 95724.31k | 95885.82k | 163573.76k | 163919.19k |
| aes−256 cbc | 449255.73k | 465307.64k | 470419.49k | 471878.72k | 472747.46k |

| | sign | verify | sign/s | verify /s |
|---|---|---|---|---|
| rsa 4096 bits | 0.038482s | 0.000569s | 26.0 | 1758.3 |
| rsa 4096 bits | 0.038213s | 0.000603s | 26.2 | 1657.4 |
| rsa 4096 bits | 0.002783s | 0.000044s | 359.3 | 22482.6 |