OpenSSL

Voyage en ligne de commande

MSSIS 2324

nov 2023

OpenSSL

Le couteau suisse de la crypto ...

Fonctionnalites

- **Implementation du protocole SSL/TLS** ;
- **Generation de bi-cles (RSA, DH, DSA, ...)** ;
- Creation de certificats, CA ;
- Gestion des listes de revocation ;
- Signatures, empreintes ;
- Chiffrement symetrique/asymetrique ;
- ...

Comment ?

- **en ligne de commande** ;
- en tant que bibliotheque.

Telechargement

```
$ ftp_proxy="your_ftp_proxy:port"; export ftp_proxy
$ VERSION="3.2.0"
$ wget -c ftp://ftp.openssl.org/source/openssl-"$VERSION".tar.gz
```

Installation

```
$ tar -xvzf openssl-"$VERSION".tar.gz"
$ cd openssl-$VERSION"
$ make
$ cd apps && ./openssl version
⇝ OpenSSL 3.2.0 23 November 2023
```

Allons-y !

```
$ openssl help
```

Chiffrement symetrique

Rappels de cryptographie ;-)

Chiffrement symetrique

Rappels de cryptographie ;-)

Concept de crypto symetrique

Une meme cle pour le chiffrement et pour le dechiffrement

Concept de crypto symetrique

Une meme cle pour le chiffrement et pour le dechiffrement

Definition d'un systeme cryptographique *(Douglas Stinson)*

Un systeme cryptographique est un quintuplet (P, C, K, E, D) satisfaisant :

1. P est un ensemble fini de blocs de *textes clairs* possibles ;
2. C est un ensemble fini de blocs de *textes chiffres* possibles ;
3. K, que l'on appelle *l'espace des cles*, est un ensemble fini de *cles possibles* ;
4. Pour tout $k \in K$, il existe une *regle de chiffrement* $e_k \in E$ et une *regle de dechiffrement* correspondante $d_k \in D$. Chaque $e_k : P \to C$ et $d_k : C \to P$ sont des fonctions telles que $d_k(e_k(x)) = x$ pour tout test clair $x \in P$.

Chiffrement symetrique

Rappels de cryptographie ;-)

Algorithmes

Nom	Taille des cles (bits)	Taille des blocs (bits)
DES	56	64
3DES	56, 112, 168	64
AES	128, 192, 256,	128
Serpent	128, 192, 256,	128
Twofish	128, 192, 256,	128

Modes d'operation

- ECB
- CBC, CTS
- OFB, CFB
- CTR
- XTS (XEX-TCB-CTS)

Un algorithme de chiffrement par bloc est défini par :

- la taille de la clé secrète
- la taille du bloc de données qu'il peut traiter

Algorithme	Taille Cle (bits)	Taille Bloc (bits)	A Utiliser
DES	56	64	KO
3DES	56 / 112 / 168	64	OK
Blowfish	32 a 448	64	KO
AES (Rijndael)	128 / 192 / 256	128	OK
Twofish	128 / 192 / 256	128	OK
RC6	128 / 192 / 256	128	OK
Serpent	128 / 192 / 256	128	OK
Mars	128 / 192 / 256	128	OK

Chiffrement (*rtfm*)

```
$ openssl enc -help
⇝ -algo-taille_cle-mode / -e / -d / -in / -out
```

Chiffrement classique

```
$ echo -n "mssis_2324" > 01_plain
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher
⇝ enter aes-128-ecb encryption password:
⇝ Verifying - enter aes-128-ecb encryption password:
$ ls -l 01_{plain,cipher} | awk '{print $9" "$5" bytes"}'
⇝ 01_cipher 32 bytes
⇝ 01_plain 10 bytes
$ hexdump -C 01_cipher
```

Chiffrement (*rtfm*)

```
$ openssl enc -help
⇝ -algo-taille_cle-mode / -e / -d / -in / -out
```

Chiffrement classique

```
$ echo -n "mssis_2324" > 01_plain
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher
⇝ enter aes-128-ecb encryption password:
⇝ Verifying - enter aes-128-ecb encryption password:
$ ls -l 01_{plain,cipher} | awk '{print $9" "$5" bytes"}'
⇝ 01_cipher 32 bytes
⇝ 01_plain 10 bytes
$ hexdump -C 01_cipher
```

Chiffrement (*rtfm*)

```
$ openssl enc -help
⇝ -algo-taille_cle-mode / -e / -d / -in / -out
```

Chiffrement classique

```
$ echo -n "mssis_2324" > 01_plain
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher
⇝ enter aes-128-ecb encryption password:
⇝ Verifying - enter aes-128-ecb encryption password:
$ ls -l 01_{plain,cipher} | awk '{print $9" "$5" bytes"}'
⇝ 01_cipher 32 bytes
⇝ 01_plain 10 bytes
$ hexdump -C 01_cipher
```

Chiffrement classique *bis* (meme mdp)

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher-bis
$ diff -q 01_{cipher,cipher-bis}
⇝ Les fichiers 01_cipher et 01_cipher-bis sont differents.
$ hexdump -C 01_cipher-bis
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
```

La presence d'un sel permet pour un meme mot de passe d'obtenir une cle et un iv de chiffrement different a chaque utilisation.

$$f_{deriv}(mdp, salt) = (key, iv)$$

sans sel !

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 02_cipher -p -nosalt
$ hexdump -C 02_cipher
```

Chiffrement classique *bis* (meme mdp)

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher-bis
$ diff -q 01_{cipher,cipher-bis}
⇝ Les fichiers 01_cipher et 01_cipher-bis sont differents.
$ hexdump -C 01_cipher-bis
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
```

La presence d'un sel permet pour un meme mot de passe d'obtenir une cle et un iv de chiffrement different a chaque utilisation.

$$f_{deriv}(mdp, salt) = (key, iv)$$

sans sel !

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 02_cipher -p -nosalt
$ hexdump -C 02_cipher
```

Chiffrement classique *bis* (meme mdp)

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher-bis
$ diff -q 01_{cipher,cipher-bis}
⇝ Les fichiers 01_cipher et 01_cipher-bis sont differents.
$ hexdump -C 01_cipher-bis
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
```

La presence d'un sel permet pour un meme mot de passe d'obtenir une cle et un iv de chiffrement different a chaque utilisation.

$$f_{deriv}(mdp, salt) = (key, iv)$$

sans sel !

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 02_cipher -p -nosalt
$ hexdump -C 02_cipher
```

Chiffrement classique *bis* (meme mdp)

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher-bis
$ diff -q 01_{cipher,cipher-bis}
⇝ Les fichiers 01_cipher et 01_cipher-bis sont differents.
$ hexdump -C 01_cipher-bis
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
```

La presence d'un sel permet pour un meme mot de passe d'obtenir une cle et un iv de chiffrement different a chaque utilisation.

$$f_{deriv}(mdp, salt) = (key, iv)$$

sans sel !

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 02_cipher -p -nosalt
$ hexdump -C 02_cipher
```

Chiffrement classique *bis* (meme mdp)

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher-bis
$ diff -q 01_{cipher,cipher-bis}
⇝ Les fichiers 01_cipher et 01_cipher-bis sont differents.
$ hexdump -C 01_cipher-bis
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
```

La presence d'un sel permet pour un meme mot de passe d'obtenir une cle et un iv de chiffrement different a chaque utilisation.

$$f_{deriv}(mdp, salt) = (key, iv)$$

sans sel !

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 02_cipher -p -nosalt
$ hexdump -C 02_cipher
```

Chiffrement classique *bis* (meme mdp)

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher-bis
$ diff -q 01_{cipher,cipher-bis}
⇝ Les fichiers 01_cipher et 01_cipher-bis sont differents.
$ hexdump -C 01_cipher-bis
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
$ openssl enc -e -aes-128-ecb -in 01_plain -out 01_cipher -p
```

La presence d'un sel permet pour un meme mot de passe d'obtenir une cle et un iv de chiffrement different a chaque utilisation.

$$f_{deriv}(mdp, salt) = (key, iv)$$

sans sel !

```
$ openssl enc -e -aes-128-ecb -in 01_plain -out 02_cipher -p -nosalt
$ hexdump -C 02_cipher
```

Mode CBC (padding)

```
$ dd if=/dev/zero of=03_plain bs=15 count=1
$ openssl enc -e -aes-128-cbc -in 03_plain -out 03_cipher -nosalt
$ ls -l 03_{plain,cipher} | awk '{print $9" "$5" bytes"}'
```

Le padding intervient pour completer le dernier bloc si sa longueur n'est pas egale a 128 bits (taille de bloc de l'algorithme).

```
$ dd if=/dev/zero of=03_plain-bis bs=16 count=1
$ openssl enc -e -aes-128-cbc -in 03_plain-bis -out 03_cipher-bis -nosalt
$ ls -l 03_{plain-bis,cipher-bis} | awk '{print $9" "$5" bytes"}'
```

Le padding intervient meme si la taille du dernier bloc est egale a 128 bits (taille de bloc de l'algorithme) $\rightarrow$ on ajoute un bloc qui ne contient que du padding.

Mode CBC (padding)

```
$ dd if=/dev/zero of=03_plain bs=15 count=1
$ openssl enc -e -aes-128-cbc -in 03_plain -out 03_cipher -nosalt
$ ls -l 03_{plain,cipher} | awk '{print $9" "$5" bytes"}'
```

Le padding intervient pour completer le dernier bloc si sa longueur n'est pas egale a 128 bits (taille de bloc de l'algorithme).

```
$ dd if=/dev/zero of=03_plain-bis bs=16 count=1
$ openssl enc -e -aes-128-cbc -in 03_plain-bis -out 03_cipher-bis -nosalt
$ ls -l 03_{plain-bis,cipher-bis} | awk '{print $9" "$5" bytes"}'
```

Le padding intervient meme si la taille du dernier bloc est egale a 128 bits (taille de bloc de l'algorithme) $\rightarrow$ on ajoute un bloc qui ne contient que du padding.

Mode CBC (padding)

```
$ dd if=/dev/zero of=03_plain bs=15 count=1
$ openssl enc -e -aes-128-cbc -in 03_plain -out 03_cipher -nosalt
$ ls -l 03_{plain,cipher} | awk '{print $9" "$5" bytes"}'
```

Le padding intervient pour completer le dernier bloc si sa longueur n'est pas egale a 128 bits (taille de bloc de l'algorithme).

```
$ dd if=/dev/zero of=03_plain-bis bs=16 count=1
$ openssl enc -e -aes-128-cbc -in 03_plain-bis -out 03_cipher-bis -nosalt
$ ls -l 03_{plain-bis,cipher-bis} | awk '{print $9" "$5" bytes"}'
```

Le padding intervient meme si la taille du dernier bloc est egale a 128 bits (taille de bloc de l'algorithme) $\rightarrow$ on ajoute un bloc qui ne contient que du padding.

Mode CBC (padding)

```
$ dd if=/dev/zero of=03_plain bs=15 count=1
$ openssl enc -e -aes-128-cbc -in 03_plain -out 03_cipher -nosalt
$ ls -l 03_{plain,cipher} | awk '{print $9" "$5" bytes"}'
```

Le padding intervient pour completer le dernier bloc si sa longueur n'est pas egale a 128 bits (taille de bloc de l'algorithme).

```
$ dd if=/dev/zero of=03_plain-bis bs=16 count=1
$ openssl enc -e -aes-128-cbc -in 03_plain-bis -out 03_cipher-bis -nosalt
$ ls -l 03_{plain-bis,cipher-bis} | awk '{print $9" "$5" bytes"}'
```

Le padding intervient meme si la taille du dernier bloc est egale a 128 bits (taille de bloc de l'algorithme) $\rightarrow$ on ajoute un bloc qui ne contient que du padding.

Cryptographie Symétrique

Chiffrement par Bloc - **padding**

blocs initiaux	`... | DD DD DD DD DD DD DD DD | EE EE |`
X.923	`... | DD DD DD DD DD DD DD DD | EE EE 00 00 00 00 00 06 |`
ISO 10126	`... | DD DD DD DD DD DD DD DD | EE EE CF 5D 89 4A 71 06 |`
PKCS #7	`... | DD DD DD DD DD DD DD DD | EE EE 06 06 06 06 06 06 |`
ISO/IEC 7816-4	`... | DD DD DD DD DD DD DD DD | EE EE 80 00 00 00 00 00 |`

Autres options

-d : dechiffrement

-a : encodage/encodage en base64

-k : phrase de passe

-kfile : phrase de passe dans un fichier

-K : cle de chiffrement en hexadecimal

-iv : iv en hexadecimal

Fonction de hachage

Une fonction de hachage cryptographique peut fournir une assurance d'**integrite de donnees**.

- Une fonction de hachage H calcule une empreinte de n bits a partir d'un message arbitraire M ;
- Une fonction de hachage est **non inversible**.

Fonction de hachage : preimage

instance : une fonction de hachage $h : X \to Y$ et un element $y \in Y$

trouver : $x \in X$ tel que $h(x) = y$

Fonction de hachage : seconde preimage

instance : une fonction de hachage $h : X \to Y$ et un element $x \in X$

trouver : $x' \in X$ tel que $x' \neq x$ et $h(x') = h(x)$

Fonction de hachage : collision

instance : une fonction de hachage $h : X \to Y$

trouver : $x, x' \in X$ tel que $x' \neq x$ et $h(x) = h(x')$

Code d'authentification

Un code d'authentification assure l'authenticite du message : le message n'a pas ete modifie au cours d'une communication ou pendant le stockage.

Principes

- Fonction de hachage à clé (*HMAC*)
- Chiffrement par bloc (*CBC-MAC, GCM*)

Algorithmes

Type	Nom	Taille de la sortie (bits)
Hash	MD5	128
Hash	SHA1	160
Hash	SHA-256	256
Hash	SHA-512	512
Hash	ripemd-160	160
Hash	Blake	512
Hash	Skein	512
Hash	Keccak	512
HMAC	HMAC-SHA1	160
HMAC	HMAC-SHA256	256
HMAC	AES-CBC-MAC	128

Le CRC32 n'est pas une fonction de hachage cryptographique, le MD5 non plus, le SHA1 ...

Hachage (*rtfm*)

```
$ openssl dgst -help
⇝ -hex / -binary / -md5 / -sha1 / ...
```

Utilisation classique

```
$ echo -n "mssis_2324" > 01_message
$ openssl dgst -hex 01_message
⇝ SHA256(01_message)= 77620c3e065db1db2505245c054d7aa2a9c582a2877221c80e9090b9fe3e405d
$ openssl dgst -hex -sha1 01_message
$ openssl dgst -hex -sha1 < 01_message
$ openssl dgst -binary -sha1 < 01_message > 01_dgst
$ hexdump -C 01_dgst
$ openssl dgst -hex -sha1 -hmac DEADFACE 01_message
⇝ HMAC-SHA1(01_message)= f5e8f82b864656f6d19f3d55f21bfb6d264304b7
```

Hachage (*rtfm*)

```
$ openssl dgst -help
⇝ -hex / -binary / -md5 / -sha1 / ...
```

Utilisation classique

```
$ echo -n "mssis_2324" > 01_message
$ openssl dgst -hex 01_message
⇝ SHA256(01_message)= 77620c3e065db1db2505245c054d7aa2a9c582a2877221c80e9090b9fe3e405d
$ openssl dgst -hex -sha1 01_message
$ openssl dgst -hex -sha1 < 01_message
$ openssl dgst -binary -sha1 < 01_message > 01_dgst
$ hexdump -C 01_dgst
$ openssl dgst -hex -sha1 -hmac DEADFACE 01_message
⇝ HMAC-SHA1(01_message)= f5e8f82b864656f6d19f3d55f21bfb6d264304b7
```

Hachage (*rtfm*)

```
$ openssl dgst -help
⇝ -hex / -binary / -md5 / -sha1 / ...
```

Utilisation classique

```
$ echo -n "mssis_2324" > 01_message
$ openssl dgst -hex 01_message
⇝ SHA256(01_message)= 77620c3e065db1db2505245c054d7aa2a9c582a2877221c80e9090b9fe3e405d
$ openssl dgst -hex -sha1 01_message
$ openssl dgst -hex -sha1 < 01_message
$ openssl dgst -binary -sha1 < 01_message > 01_dgst
$ hexdump -C 01_dgst
$ openssl dgst -hex -sha1 -hmac DEADFACE 01_message
⇝ HMAC-SHA1(01_message)= f5e8f82b864656f6d19f3d55f21bfb6d264304b7
```

Alea cryptographique

Pratique

On utilise l'alea pour les cles et les iv. Un bon generateur produit des sorties imprevisbiles et qu'on ne peut pas reproduire $\rightsquigarrow$ alea physique.

Utilisation classique

```
$ openssl rand -h
$ openssl rand 32
$ openssl rand 32 > 01_random
$ openssl rand -hex 32
$ openssl rand -base64 32
```

Utilisation avancee

```
$ dd if=/dev/urandom of=seed.raw bs=512 count=1
$ openssl rand -rand seed.raw -hex 32
```

`seed.raw` ne doit etre utilise qu'une fois $\rightsquigarrow$ mise a jour necessaire

Alea cryptographique

Pratique

On utilise l'alea pour les cles et les iv. Un bon generateur produit des sorties imprevisbiles et qu'on ne peut pas reproduire $\rightsquigarrow$ alea physique.

Utilisation classique

```
$ openssl rand -h
$ openssl rand 32
$ openssl rand 32 > 01_random
$ openssl rand -hex 32
$ openssl rand -base64 32
```

Utilisation avancee

```
$ dd if=/dev/urandom of=seed.raw bs=512 count=1
$ openssl rand -rand seed.raw -hex 32
```

`seed.raw` ne doit etre utilise qu'une fois $\rightsquigarrow$ mise a jour necessaire

Alea cryptographique

Pratique

On utilise l'alea pour les cles et les iv. Un bon generateur produit des sorties imprevisbiles et qu'on ne peut pas reproduire $\rightsquigarrow$ alea physique.

Utilisation classique

```
$ openssl rand -h
$ openssl rand 32
$ openssl rand 32 > 01_random
$ openssl rand -hex 32
$ openssl rand -base64 32
```

Utilisation avancee

```
$ dd if=/dev/urandom of=seed.raw bs=512 count=1
$ openssl rand -rand seed.raw -hex 32
```

`seed.raw` ne doit etre utilise qu'une fois $\rightsquigarrow$ mise a jour necessaire

Alea cryptographique

Pratique

On utilise l'alea pour les cles et les iv. Un bon generateur produit des sorties imprevisbiles et qu'on ne peut pas reproduire $\rightsquigarrow$ alea physique.

Utilisation classique

```
$ openssl rand -h
$ openssl rand 32
$ openssl rand 32 > 01_random
$ openssl rand -hex 32
$ openssl rand -base64 32
```

Utilisation avancee

```
$ dd if=/dev/urandom of=seed.raw bs=512 count=1
$ openssl rand -rand seed.raw -hex 32
```

`seed.raw` ne doit etre utilise qu'une fois $\rightsquigarrow$ mise a jour necessaire

Cryptographie asymetrique

Rappels de cryptographie ;-)

Cryptographie asymetrique

Rappels de cryptographie ;-)

Concept de crypto asymetrique

Un couple (cle publique, cle privee) : seule la cle privee doit rester secrete

Cryptographie asymetrique

Rappels de cryptographie ;-)

Concept de crypto asymetrique

Un couple (cle publique, cle privee) : seule la cle privee doit rester secrete

Les problemes mathematiques

factorisation : RSA ;

logarithme discret : DSA, Diffie-Hellman, ElGamal, Courbes Elliptiques.

Generation de bi-cles (*rtfm*)

```
$ openssl genrsa -h
```

Generation de bi-cles

protection cle privee

```
$ openssl genrsa -aes128 1024 > rsa_priv.pem
$ openssl rsa -aes128 -in rsa_priv.pem > rsa_priv.pem
~> Enter PEM pass phrase:     ~> Verifying - Enter PEM pass phrase:
```

cle privee / cle publique

```
$ openssl rsa -in rsa_priv.pem -pubout > rsa_pub.pem
$ openssl rsa -noout -text -pubin -in rsa_pub.pem
```

Cryptographie asymetrique

RSA - Generation

Generation de bi-cles (*rtfm*)

```
$ openssl genrsa -h
```

Generation de bi-cles

```
$ openssl genrsa 1024
```

protection cle privee

```
$ openssl genrsa -aes128 1024 > rsa_priv.pem
$ openssl rsa -aes128 -in rsa_priv.pem > rsa_priv.pem
⇝ Enter PEM pass phrase:     ⇝ Verifying - Enter PEM pass phrase:
```

cle privee / cle publique

```
$ openssl rsa -in rsa_priv.pem -pubout > rsa_pub.pem
$ openssl rsa -noout -text -pubin -in rsa_pub.pem
```

Cryptographie asymetrique

RSA - Generation

Generation de bi-cles (*rtfm*)

```
$ openssl genrsa -h
```

Generation de bi-cles

```
$ openssl genrsa 1024 > rsa_priv.pem
$ cat rsa_priv.pem
$ openssl rsa -noout -text -in rsa_priv.pem
```

protection cle privee

```
$ openssl genrsa -aes128 1024 > rsa_priv.pem
$ openssl rsa -aes128 -in rsa_priv.pem > rsa_priv.pem
⇝ Enter PEM pass phrase:    ⇝ Verifying - Enter PEM pass phrase:
```

cle privee / cle publique

```
$ openssl rsa -in rsa_priv.pem -pubout > rsa_pub.pem
$ openssl rsa -noout -text -pubin -in rsa_pub.pem
```

Cryptographie asymetrique

RSA - Generation

Generation de bi-cles (*rtfm*)

```
$ openssl genrsa -h
```

Generation de bi-cles

```
$ openssl genrsa 1024 > rsa_priv.pem
$ cat rsa_priv.pem
$ openssl rsa -noout -text -in rsa_priv.pem
```

protection cle privee

```
$ openssl genrsa -aes128 1024 > rsa_priv.pem
$ openssl rsa -aes128 -in rsa_priv.pem > rsa_priv.pem
⇝ Enter PEM pass phrase:     ⇝ Verifying - Enter PEM pass phrase:
```

cle privee / cle publique

```
$ openssl rsa -in rsa_priv.pem -pubout > rsa_pub.pem
$ openssl rsa -noout -text -pubin -in rsa_pub.pem
```

Generation de bi-cles (*rtfm*)

```
$ openssl genrsa -h
```

Generation de bi-cles

```
$ openssl genrsa 1024 > rsa_priv.pem
$ cat rsa_priv.pem
$ openssl rsa -noout -text -in rsa_priv.pem
```

protection cle privee

```
$ openssl genrsa -aes128 1024 > rsa_priv.pem
$ openssl rsa -aes128 -in rsa_priv.pem > rsa_priv.pem
⇝ Enter PEM pass phrase:     ⇝ Verifying - Enter PEM pass phrase:
```

cle privee / cle publique

```
$ openssl rsa -in rsa_priv.pem -pubout > rsa_pub.pem
$ openssl rsa -noout -text -pubin -in rsa_pub.pem
```

Rappel

- La signature identifie celui qui signe ;
- on signe avec notre cle privee ;
- le destinataire verifie avec notre cle publique.

Exemple 1

```
$ echo -n "mssis_2324" > message.txt
$ openssl dgst -sign rsa_priv.pem message.txt > message.sign
$ openssl dgst -verify rsa_pub.pem -signature message.sign message.txt
⇝ Verified OK     ⇝ Verification Failure
```

Exemple 2

```
$ dd if=/dev/zero of=message2.txt bs=1M count=1
$ openssl dgst -sign rsa_priv.pem message2.txt > message2.sign
$ openssl dgst -verify rsa_pub.pem -signature message2.sign message2.txt
```

Exemple 2

```
$ dd if=/dev/zero of=message2.txt bs=1M count=1
$ openssl dgst -sign rsa_priv.pem message2.txt > message2.sign
$ openssl dgst -verify rsa_pub.pem -signature message2.sign message2.txt
```

Expliquez pourquoi ca marche !

Exemple 2

```
$ dd if=/dev/zero of=message2.txt bs=1M count=1
$ openssl dgst -sha256 -sign rsa_priv.pem message2.txt > message2.sign
$ openssl dgst -sha256 verify rsa_pub.pem -signature message2.sign message2.txt
```

Rappel

- On signe l'empreinte du message, pas le message

Rappel

- On chiffre avec la cle publique du destinataire ;
- le destinataire dechiffre avec sa cle privee.

Exemple 1

```
$ echo -n "mssis_2324" > message.txt
$ openssl rsautl -encrypt -inkey rsa_pub.pem -pubin -in message.txt -out message.enc
$ openssl rsautl -decrypt -inkey rsa_priv.pem -in message.enc -out message.dec
⇝ $ diff message.txt message.dec
```

Exemple 2

```
$ dd if=/dev/zero of=message2.txt bs=1M count=1
$ openssl rsautl -encrypt -inkey rsa_pub.pem -pubin -in message2.txt -out message2.enc
```

Exemple 2

```
$ dd if=/dev/zero of=message2.txt bs=1M count=1
$ openssl rsautl -encrypt -inkey rsa_pub.pem -pubin -in message2.txt -out message2.enc
⇝ RSA operation error (...) data too large for key size
```

Exemple 2

```
$ dd if=/dev/zero of=message2.txt bs=1M count=1
$ openssl rsautl -encrypt -inkey rsa_pub.pem -pubin -in message2.txt -out message2.enc
⇝ RSA operation error (...) data too large for key size
```

⇝ Trouver une solution !

Exemple 2

```
$ dd if=/dev/zero of=message2.txt bs=1M count=1
```

Chiffrement hybride :

1. on genere une cle symetrique K ;
2. on chiffre le message avec un algorithme symetrique et la cle K ;
3. on chiffre en asymetrique la cle K ;
4. on transmet les deux messages chiffres.

Cryptographie asymetrique

Autres algorithmes

DH : `$ openssl gendh 1024 -out dh_key.pem`

DSA : `$ openssl dsaparam -genkey 1024`

ECC : `$ openssl ecparam ...`

Certificat

Un certificat est un lien reconnu par une autorite entre une cle publique et une identite.

Norme x509

- version ;
- serial number ;
- issuer (nom de la CA) ;
- validity ;
- subject ;
- subject public key information ;
- (v2) issuer unique identifier ;
- (v2) subject unique identifier ;
- (v3) extensions ;
- signature de la CA.

Etapes :

1. creation de l'environnement de la CA ;
2. generation de la cle privee et du certificat autosigne de la CA ;
3. generation de la cle privee et de la requete de certificat pour un utilisateur ;
4. generation du certificat par la CA a partir d'une requete ;
5. gestion de la liste de revocation.

Configuration des protagonistes

```
$ mkdir CA USERS
```

Configuration de la CA

```
$ cd CA
```

- Mise en place de l'environnement

```
$ mkdir certs newcerts private
```

```
$ echo 01 > serial
```

```
$ touch index.txt
```

- Creation du certificat autosigne de la CA

```
$ openssl genrsa -out cakey.pem
```

```
$ openssl req -new -x509 -key cakey.pem -out cacert.pem
    -config openssl.conf -extensions CA_ROOT
```

Au niveau des utilisateurs

```
$ cd ../USERS
```

- Creation des cles RSA

  ```
  $ openssl genrsa -out user1_key.pem 2048
  ```

- Creation de la requete de certificat

  ```
  $ openssl req -new -key user1_key.pem -out user1_req.pem
  ```

- Envoi de la requete a la CA

  ```
  $ cp user1_req.pem ../CA/certs
  ```

Au niveau de la CA

```
$ cd ../CA
```

- Generation du certificat signe

```
$ openssl ca -out ./certs/user1_crt.pem -config ./openssl.conf
    -extensions CLIENT_RSA_SSL -infiles ./certs/user1_req.pem
```

- Envoi du certificat a l'utilisateur

```
$ cp ./certs/user1_crt.pem ../USERS/
```