Sicurezza dei Sistemi Informatici

Esercitazioni OpenSSL

Creazione e Gestione Certificati

Roberta Daidone

roberta.daidone@iet.unipi.it

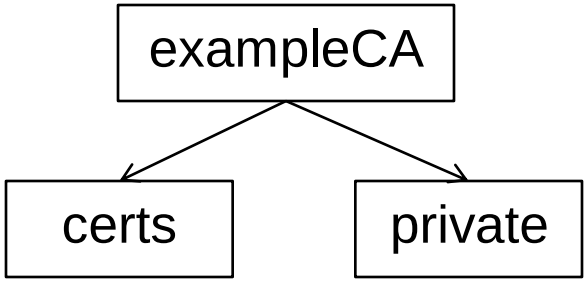
Obiettivi

- Setup di una Certification Authority
- Creazione di un *root certificate* autofirmato
- Generazione di una richiesta di certificato
- Generazione di un certificato X509
- Firma e verifica dell’ hash di un file

Setup di una CA a riga di comando

- Creazione di cartelle per la CA

```
$ mkdir exampleCA
$ cd exampleCA
$ mkdir certs,private
```



- Cambio di permessi su ***private***:

```
$ sudo chmod 700 private
$ ls -l
```
(per verificare)

Setup di una CA a riga di comando

- Creazione del file ***serial***. Serve a tracciare i numeri seriali dei certificati. Parte da 0x01

```
$ echo '01'> serial
```

- Creazione del file ***index.txt***. Funge da database dei certificati emessi

```
$ touch index.txt
```

- Creazione del file di configurazione ***openssl.cnf***

```
$ touch openssl.cnf
```

Configurazione OPENSSL_CONF

- Settare la **variabile d'ambiente** `OPENSSL_CONF` con il path del file ***openssl.cnf***

```
$ OPENSSL_CONF=./exampleCA/openssl.cnf
$ export OPENSSL_CONF
```

- Per verficare:

```
$ echo $OPENSSL_CONF
```

Come è fatto openssl.cnf?

- Consiste in una sequenza di sezioni individuate da un nome fra parentesi quadre [nome]

- Ogni sezione contiene una serie di coppie del tipo *keyword = valore*

- Le sezioni fondamentali sono:
  - `[ca]` Per settare il comando **ca**. Il comando ca serve per emettere e firmare certificati, oppure emettere CRL
  - `[req]` Per settare il comando **req**. Il comando req serve per emettere il root certificate autofirmato, oppure richiedere certificati

openssl.cnf

```
[ ca ]
default_ca   = exampleCA

[ exampleCA ]
dir              = ./
certificate      = $dir/cacert.pem
database         = $dir/index.txt
serial           = $dir/serial
new_certs_dir    = $dir/certs
private_key      = $dir/private/privkey.pem

default_days     = 365
default_crl_days = 7
default_md       = md5
```

openssl.cnf

```
 policy          = exampleca_policy
x509_extensions= certificate_extensions

[ exampleca_policy ]
commonName                = supplied
stateOrProvinceName    = supplied
countryName               = supplied
organizationName       = supplied
organizationalUnitName = optional

[ certificate_extensions ]
basicConstraints = CA:false
```

Generazione del root certificate

- L'utilizzo del comando **req** richiede che la sezione [req] venga inserita in openssl.cnf

```
[ req ]
default_bits        = 2048
default_keyfile     = ./private/privkey.pem
default_md          = md5
prompt              = no
distinguished_name=root_ca_distinguished_name
x509_extensions     = root_ca_extensions

[ root_ca_distinguished_name ]
commonName                = Daidone CA
stateOrProvinceName = Italy
```

Generazione del root certificate

```
countryName             = EU
emailAddress            = daidone@mycert.it
organizationName        = Root Cert Authority

[ root_ca_extensions ]
basicConstraints        = CA:true
```

- Adesso è possibile generare il **root certificate** (autofirmato) e le **chiavi pubblica e privata** della CA

Generazione del root certificate

- Utilizzare il comando **req**

```
$ cd exampleCA/
$ openssl req -x509 -newkey rsa:2048
  -out cacert.pem -outform PEM
```

  - `-x509` è il formato del certificato
  - `-newkey rsa:2048` specifica che verranno generate 2 chiavi RSA da 2048 bit ciascuna
  - `-out <file>` file di output per il certificato
  - `-outform` formato dell'output

Effetti

- Viene chiesta una password.

- Vengono generati due file:
  - ***privkey.pem*** in *exampleCA/private*.
  - ***cacert.pem*** in *exampleCA/*.

- Per visualizzare il certificato usare il comando **x509**:

```
$ openssl x509 -in cacert.pem -text -noout
```

  - `-text` per stampare in forma testuale
  - `-noout` per specificare che vogliamo la versione non codificata della richiesta

Richiesta di certificato

- L'utente richiede una coppia di chiavi contestualmente alla richiesta del certificato.

- La CA legge la richiesta e le chiavi da associare al certificato da generare

- Utilizzare una nuova shell, per non avere la variabile d'ambiente OPENSSL_CONF settata.

- Usare il comando **req**

```
$ openssl req -newkey rsa:1024 -keyout
  testkey.pem -keyform PEM -out
  testreq.pem
```

Effetti

- OPENSSL_CONF non è settata => il prompt chiede più informazioni

- Vengono richieste due password:
  - Una per cifrare la chiave privata
  - Una *challenge* salvata dentro la richiesta

- Si ottengono 2 file:
  - ***testkey.pem*** contiene la chiave privata
  - ***testreq.pem*** contiene la richiesta di certificato

- Per visualizzare la richiesta usare il comando **req**:

```
$ openssl req –in testreq.pem -text –noout
```

Emissione di un certificato

- Tornare alla shell con OPENSSL_CONF settata
- Grazie alla richiesta generata, basta fornire:

```
$ openssl ca –in testreq.pem
```

- Effetti:
  - Viene richiesta una password per accertare che si possa usare la chiave privata della CA
  - Viene chiesto se si vuole la firma della CA sul certificato
  - Viene visualizzato il certificato generato
  - in ***exampleCA/certs*** c'è il file ***<serial>.pem***

Emissione di una CRL

- Si usa il comando **ca** con l'opzione `-gencrl`

```
$ openssl ca -gencrl –out CRLfile.pem –keyform PEM
```

- Effetti:
  - Viene richiesta una password per accertare che si possa usare la chiave privata della CA
  - Se la verifica della firma della CA va a buon fine, viene generato il file ***CRLfile.pem***

- Per visualizzare la CRL usare il comando **crl**:

```
$ openssl crl –in CRLfile.pem -text –noout
```

Hash e firma

- Creare il file ***data.txt*** su cui fare hash e firma

```
$ echo 'stringa da firmare' > data.txt
```

- Creare il file ***hash*** contenente l'hash:

```
$ openssl dgst -sha1 < data.txt > hash
```

- Firmando il file ***hash*** si ottiene il file ***signature***:

```
$ openssl rsautl -sign -inkey
  testkey.pem -keyform PEM -in hash >
  signature
```

Recupero della chiave pubblica

- Si usa il comando **rsa** per ricavare la chiave pubblica dal file ***testkey.pem***

```
$ openssl rsa -in testkey.pem -out public.pem -outform PEM –pubout
```

  - `–pubout` impone che l'output sia la chiave pubblica. Di default è l'argomento di `–in`

- Per visualizzare la chiave pubblica:

```
$ openssl rsa -in public.pem -text -noout –pubin
```

  - `–pubin` specifica che leggo una chiave pubblica

Verifica

- Verificare il file ***signature*** per ottenere il file ***verified*** contenente l'hash

```
$ openssl rsautl -verify -inkey public.pem -keyform PEM -pubin -in signature > verified
```

- Verificare che i file ***verified*** e ***hash*** coincidano:

```
$ diff -s verified hash
```

`-s` fornisce un feedback quando i due file sono identici

Esercizio

- Creare una coppia di chiavi a riga di comando

- Sostituire le chiavi create con quelle sfruttate per l'eserczio su RSA fatto la volta scorsa

- Riadattare il codice di client e server per usare le nuove chiavi
  - Suggerimento: le chiavi usate la volta scorsa erano di tipo RSA, queste sono di tipo EVP_PKEY