PKI 入門その 2

実践編: OpenSSL による CA の構築

いまい ひろえい

NISOC 勉強会 @ 2007/01/06

1 課題

認証局（CA:Certificate Authority）を作って，鍵の作成，証明書要求，証明書発行，証明書の検証を，手動でやってみる．使うツールは OpenSSL のみ．

- ルート CA となる CA0 と，CA0 が発行した証明書を持つ 2 つの CA（CA1，CA2）の，計 3 つの CA を作成する．
- CA1 と CA2 がクライアント証明書を発行する，互いの証明書の信頼性を検証する．
- 使うのは一つの PC のみ．CA やクライアントに必要なファイル群をおくディレクトリを分けることで，仮想的に複数の CA とクライアントを作成する．

2 ルート CA（CA0）をつくる

鍵を生成して，自己署名の証明書を作成する．

1. ルート CA 用のディレクトリ（以下`~CA0`）を作成し，`openssl.cnf` をコピーしてきて編集．
   - `[ CA_default ]` セクションの `dir` を，`~CA0` に変更．
   - オプション：`[ req_distinguished_name ]` セクションを予め編集しておくことで，後に CSR に含める組織名等の入力を省くことができる．
2. `openssl.cnf` で指定したディレクトリを作成する．

```
@ca0:~ $ mkdir ~CA0
@ca0:~ $ cd ~CA0
@ca0:~CA0 $ mkdir certs crl newcerts private
@ca0:~CA0 $ chmod 700 private
@ca0:~CA0 $ echo '01' > serial
@ca0:~CA0 $ touch index.txt
```

3. プライベートキーの作成/dev/urandomを乱数の種とし，1024ビットのプライベートキーを生成し，~CA0/private/cakey.pemというファイルに保存する．その後，パーミッションを600に変更する．

```
@ca0:~CA0 $ openssl genrsa -rand /dev/urandom -out \
    private/cakey.pem 1024
Generating RSA private key, 1024 bit long modulus
.....................++++++
.........++++++
e is 65537 (0x10001)
@ca0:~CA0 $ chmod 600 private/cakey.pem
```

- cakey.pemには，公開鍵-秘密鍵のペア等重要な情報が保存されている．絶対に盗まれてはいけない．
- /dev/urandomはOS依存．適当なデバイスがない場合は，頻繁に値が書きかわるようなファイル（ログファイル等）で代用する．
- cakey.pem自体を暗号化しておきたい場合は，-des3オプションを付けて生成する．その際，パスフレーズを聞かれるので入力する．

4. 自己署名のCA証明書を作成するプライベートキー~CA0/private/cakey.pemを自己署名し，x509形式の証明書をcacert.pemというファイルに作成する．

```
@ca0:~CA0 $ openssl req -x509 -config openssl.cnf -new -key \
    private/cakey.pem -out cacert.pem
You are about to be asked to enter information that will be \
    incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished \
    Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [JP]:
State or Province Name (full name) [Niigata]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Example0 Ltd]:
Organizational Unit Name (eg, section) [CA Section]:
Common Name (eg, YOUR name) [CA @ Example0 Ltd]:
Email Address [ca0@example0.jp]:
```

- openssl.cnf の [req_distinguished_name] セクションの値を予め書き換えておくと入力が楽.
- commonName には CA の名前を書く.

3 CA1 と CA2 をつくる

1. CA1 用のディレクトリ（以下~CA1）を作成し，openssl.cnf をコピーしてきて編集．編集箇所は CA0 と同じ.
2. プライベートキーの作成

```
@ca1:~CA1 $ openssl genrsa -rand /dev/urandom -out \
    private/cakey.pem 1024
```

3. CA 証明書発行要求 (CSR) の作成

```
hre@ca1:~CA1 $ openssl req -config openssl.cnf -new -key \
    cakey.pem -out CA1req.pem
```

4. CA 証明書の署名と発行 CA0 に CA1req.pem を持ってくる．CA0 に移動する．policy を policy_anything とし v3_ca の設定で，CSR を CA1req.pem というファイルから読み込み，自分の秘密鍵で署名した後，CA1cert.pem に保存する.

```
@ca0:~CA0 $ openssl ca -config openssl.cnf -policy \
    policy_anything -extensions v3_ca -out CA1cert.pem -in \
    CA1req.pem
Using configuration from openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Jan  4 17:44:07 2007 GMT
            Not After : Jan  4 17:44:07 2008 GMT
        Subject:
            countryName               = JP
            stateOrProvinceName       = Niigata
            organizationName          = Example1 Ltd
            organizationalUnitName    = CA Section
            commonName                = CA @ Example1 Ltd
```

```
            emailAddress              = ca1@example1.jp
        X509v3 extensions:
            X509v3 Subject Key Identifier:
96:DB:02:EE:96:52:BF:92:A4:25:FF:7E:0A:5F:FA:FE:76:F1:42:60
            X509v3 Authority Key Identifier:
keyid:F1:31:62:45:AA:BE:83:95:93:53:07:C7:C0:F9:0B:4C:87:BE:8B:BA
DirName:/C=JP/ST=Niigata/O=Example0 Ltd/OU=CA Section/CN=CA @ \
    Example0 Ltd/emailAddress=ca0@example0.jp
                serial:93:41:31:B9:5A:01:04:D3

            X509v3 Basic Constraints:
                CA:TRUE
Certificate is to be certified until Jan 4 17:44:07 2008 GMT (365 \
    days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

- 正常に発行できると，~CA0 以下のいくつかのファイルが更新される．

5. CA 証明書を CA1 にコピーする．

```
@ca0:~CA0 $ cp CA1cert.pem ~CA1/cacert.pem
```

CA2 についても同様に作成する．

4 クライアント証明書の作成

人やホスト等を証明するためのクライアント証明書を作成する．Client1 は CA1 に認証してもらい，Client2 は CA2 に認証してもらう．

- 鍵の作成

```
@client1:~Client1 $ openssl genrsa -rand /dev/random -out \
    client1_key.pem 1024
```

- 証明書発行要求の作成

```
@client1:~Client1 $ openssl req -config openssl.cnf -new -key \
    client1_key.pem -out client1_req.pem
```

- CA1 で証明書の発行

```
hre@ca1:~CA1 $ openssl ca -config openssl.cnf -policy \
    policy_match -out client1_cert.pem -in client1_req.pem
```

  - CA 証明書と違う所は，policy と`-extensions x509_ca` がついていない所.

5 CA1 が CA2 の証明書の確認する

1. 必要なものを集めるディレクトリを`~Client1` とする.
2. `~Client1` に以下のものをコピーしてくる.
   - Client2 の証明書（client2_cert.pem）
   - CA2 の証明書（ca2_cert.pem）
   - CA0 の証明書（ca0_cert.pem）
3. openssl が検証できるように準備.

```
@ca1:~ $ cd ~CA1/certs
@ca1:~CA1/certs $ ln -s ca0_cert.pem `openssl x509 -noout -hash < \
    ca0_cert.pem`.0
@ca1:~CA1/certs $ ln -s ca2_cert.pem `openssl x509 -noout -hash < \
    ca2_cert.pem`.0
@ca1:~CA1/certs $ ln -s client2_cert.pem `openssl x509 -noout \
    -hash < client2_cert.pem`.0
@ca1:~CA1/certs $ ls -l
total 6
lrwxr-xr-x 1 ca-adm ca-adm 12 Jan 5 15:57 265d4ceb.0@ -> \
    ca2_cert.pem
lrwxr-xr-x 1 ca-adm ca-adm 12 Jan 5 15:57 7bb5240c.0@ -> \
    ca0_cert.pem
lrwxr-xr-x 1 ca-adm ca-adm 16 Jan 5 15:57 c6d65fca.0@ -> \
    client2_cert.pem
-rw-r--r--   1 ca-adm  ca-adm  1273 Jan  5 15:55 ca0_cert.pem
```

```
-rw-r--r--   1 ca-adm  ca-adm  3516 Jan  5 15:55 ca2_cert.pem
-rw-r--r--   1 ca-adm  ca-adm  3572 Jan  5 03:52 client1_cert.pem
-rw-------   1 ca-adm  ca-adm   887 Jan  5 03:52 client1_key.pem
-rw-r--r--   1 ca-adm  ca-adm   664 Jan  5 03:52 client1_req.pem
-rw-r--r--   1 ca-adm  ca-adm  3563 Jan  5 15:55 client2_cert.pem
-rw-r--r--   1 ca-adm  ca-adm  9440 Jan  5 03:52 openssl.cnf
```

4. 検証.

```
@client1:~Client1 $ openssl verify -CApath ~Client1 \
    client2_cert.pem
client2_cert.pem: OK
```

- openssl は自動的に証明書をかき集めて検証してくれるわけではない...（たぶん）
- つまり，client2_cert.pem を検証する場合...

  1. cert_file = client2_cert.pem
  2. cert_file の中身をみて，有効期限等のチェックを行う.
  3. cert_file は信頼できる CA なら終了.
  4. cert_file に署名した CA を調べ，その CA 証明書を手に入れる.
  5. CA 証明書を~Client1 にコピーし，ハッシュ値を使ってリンクを作成する.
  6. cert_file =（今コピーしてきた CA 証明書）として (2) に戻る.

  ということを繰り返す．実際には CRL（失効リスト）もチェックする必要がある.

6 証明書をどうやって配布するか？

1. 個人配布
   - CD 等で直接手渡しする or 電子メール等で送る.
   - 規模に限界がある.
   - 失効情報の確実な伝達ができない.
2. イン・バンド・プロトコル
   - 証明書を利用するプロトコルを用いて送る．例えば，S/MIME で署名付きのメールを送ると，メールに証明書が組み込まれた形で送られる.

ちょっとやってみると....

```
@client1:~Client1 $ echo test message >| message.raw
@client1:~Client1 $ openssl smime -sign -signer client1_cert.pem \
    -inkey client1_key.pem -in message.raw -out message.signed \
    -from yamada@example1.jp

... message.signed を Client2 に送る ...

@client2:~Client2 $ ls -l *.pem *.0
lrwxr-xr-x 1 cli2 cli2 12 Jan 6 09:27 50681737.0@ -> ca1_cert.pem
lrwxr-xr-x 1 cli2 cli2 12 Jan 6 09:28 7bb5240c.0@ -> ca0_cert.pem
-rw-r--r--   1 cli2  cli2  1273 Jan  6 09:28 ca0_cert.pem
-rw-r--r--   1 cli2  cli2  3516 Jan  6 09:26 ca1_cert.pem
-rw-r--r--   1 cli2  cli2  3516 Jan  6 09:26 cacert.pem
-rw-r--r--   1 cli2  cli2  3563 Jan  5 03:53 client2_cert.pem
-rw-------   1 cli2  cli2   887 Jan  5 03:52 client2_key.pem
-rw-r--r--   1 cli2  cli2   655 Jan  5 03:53 client2_req.pem
@client2:~Client2 $ openssl smime -verify -CApath . -in \
    message.signed
test message
Verification successful
```

message.signed の中身はこんな感じ...

```
From: yamada@example1.jp
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/x-pkcs7-signature"; \
    micalg=sha1; boundary="----B24B814041DB10CF405DC326882DAF9C"

This is an S/MIME signed message

------B24B814041DB10CF405DC326882DAF9C
test message

------B24B814041DB10CF405DC326882DAF9C
Content-Type: application/x-pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIIFpwYJKoZIhvcNAQcCoIIFmDCCBZQCAQExCzAJBgUrDgMCGgUAMAsGCSqGSIb3
...
vX815MRaJCM7G3s=

------B24B814041DB10CF405DC326882DAF9C--
```

3. リポジトリによる配布
   - LDAP（RFC2559, RFC2587), FTP/HTTP(RFC2585), DNS(RFC2538) を使って送る.

7 まとめ

- 面倒です．鍵をつくるのはともかく，通信相手の鍵の検証が大変.（有名所の CA にから証明書を発行してもらえば良いだけ？）
- 独自の CA を作ると嬉しいことなんてあるの？
  - ユーザにクライアント証明書を発行し，それを使って Web アプリや VPN の認証に使うことで，よりセキュアな通信が構築できる，こともあります.
  - 自己署名のサーバ証明書を Web サーバで使う場合は,「信じるか？」「はい」で済みますが，独自の CA の署名付きの場合は，CA の証明書を予めブラウザに教えておく必要かあるようです.

8 Bibliography

1. PKI 関連技術解説[1]
2. 日替わり実験室 第 16 回 OpenSSL で PKI[2]
3. openssl:CA[3]
4. OpenSSL による CA の運営方法[4]

A Policy について

CA 自身の証明書と証明対象の証明書の属性がどれだけあっているものを対象とするか指定する.

リスト 1: 例 (openssl.cnf より抜粋)

```
[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ policy_anything ]
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

[1] http://www.ipa.go.jp/security/pki/
[2] http://www.daily-labo.com/ygg16.html
[3] http://e.tir.jp/wiliki?openssl%3ACA
[4] http://mars.elcom.nitech.ac.jp/Research/MM/security/openssl/ca.html

policy_anything は他の組織のための証明書を発行する．policy_matching は CA と同じ組織の証明書を発行する．

B　公開鍵証明書の種類

- CA 証明書
- サーバ/クライアント証明書

証明書の形式が違うわけではなく，CA 証明書のみ CA のためのものであるという属性がついており，他の証明書に署名するために使うことができる．サーバ証明書やクライアント証明書は，他の証明書にサインするのには使えない．

C　ファイルの中身

- プライベートキーの中身を解析して表示する．

```
@ca0: ~CA0 $ openssl rsa -in private/cakey.pem -text
Private-Key: (1024 bit)
modulus:
    00:be:1a:1f:13:2f:6f:b0:99:66:ac:67:b4:d5:1a:
    ...
publicExponent: 65537 (0x10001)
privateExponent:
    43:33:3a:ae:ee:82:dd:fb:ac:36:df:c6:25:da:c8:
    ...
prime1:
    00:f8:8d:13:c5:62:32:f3:47:e6:97:27:78:2a:53:
    ...
prime2:
    00:c3:cc:9d:2a:ce:12:57:ce:3e:dc:df:ea:25:50:
    ...
exponent1:
    03:14:5c:66:59:ad:1f:ee:88:20:df:50:51:d5:ee:
    ...
exponent2:
    72:73:d7:e7:5e:a0:20:4d:82:37:90:78:09:2a:1c:
    ...
coefficient:
    36:2d:6d:04:70:71:d3:2f:bf:e2:a4:f4:fc:80:fc:
    ...
-----BEGIN RSA PRIVATE KEY-----
```

```
MIICWwIBAAKBgQC+Gh8TL2+wmWasZ7TVGjQOAvUde+o8/TApJ++LlxVa6h6MiZ4T
...
SZrksVkbJuoga9wlc8oM+I9zuo8DwvBIcOAaZKpfHQ==
-----END RSA PRIVATE KEY-----
```

日本語では「秘密鍵」と訳されていることが多いが，公開鍵-秘密鍵のペアのうちの秘密鍵のみ保存されているわけではない．公開鍵-秘密鍵のペアと，2 つの素数なんかも保存されている．

- CSR の中身を解析して表示する．

```
@ca1:~CA1 $ openssl req -in CA1req.pem -text
Certificate Request:
    Data:
        Version: 0 (0x0)
Subject: C=JP, ST=Niigata, O=Example1 Ltd, OU=CA Section, CN=CA @ \
    Example1 Ltd/emailAddress=ca1@example1.jp
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b8:6c:9a:a5:00:25:54:a5:2b:39:d8:a3:10:86:
                    ...
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: md5WithRSAEncryption
        40:6d:16:d7:bd:72:3d:c0:90:e0:f1:a3:b9:6a:e2:a1:4e:54:
        ...
-----BEGIN CERTIFICATE REQUEST-----
MIIByDCCATECAQAwgYcxCzAJBgNVBAYTAkpQMRAwDgYDVQQIEwdOaWlnYXRhMRUw
...
qZxNDJX+FVPpHz/1dr2TL1Gu+hlFiqO8+ECLuQ==
-----END CERTIFICATE REQUEST-----
```

組織の情報と公開鍵，鍵のアルゴリズム等が保存されている．

- CA 証明書の中身を解析して表示する．

```
@ca1:~CA1 $ openssl x509 -in cacert.pem -text
Certificate:
    Data:
```

```
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
Issuer: C=JP, ST=Niigata, O=Example0 Ltd, OU=CA Section, CN=CA @ \
    Example0 Ltd/emailAddress=ca0@example0.jp
        Validity
            Not Before: Jan  4 17:44:07 2007 GMT
            Not After : Jan  4 17:44:07 2008 GMT
Subject: C=JP, ST=Niigata, O=Example1 Ltd, OU=CA Section, CN=CA @ \
    Example1 Ltd/emailAddress=ca1@example1.jp
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b8:6c:9a:a5:00:25:54:a5:2b:39:d8:a3:10:86:
                    ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
96:DB:02:EE:96:52:BF:92:A4:25:FF:7E:0A:5F:FA:FE:76:F1:42:60
            X509v3 Authority Key Identifier:
keyid:F1:31:62:45:AA:BE:83:95:93:53:07:C7:C0:F9:0B:4C:87:BE:8B:BA
DirName:/C=JP/ST=Niigata/O=Example0 Ltd/OU=CA Section/CN=CA @ \
    Example0 Ltd/emailAddress=ca0@example0.jp
                serial:93:41:31:B9:5A:01:04:D3

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: md5WithRSAEncryption
        3c:22:59:a8:8f:8b:0d:d5:7a:25:20:2a:0f:8d:c8:ec:40:60:
        ...
-----BEGIN CERTIFICATE-----
MIIDdjCCAt+gAwIBAgIBATANBgkqhkiG9w0BAQQFADCBhzELMAkGA1UEBhMCSlAx
...
7Di6OiHyifk7dNDDnnRh/NJIBSTfvqf1/Xc=
-----END CERTIFICATE-----
```

証明した CA，証明してもらった CA の公開鍵等の情報や，CA のための証明書であること等が分かる．

D 用語

- 認証局（CA:Certificate Authority）... 認証局．鍵の証明書を発行したり，失効した鍵の管理等をする．
- 証明書署名要求（CSR:Certificate Signing Request）... 証明書要求．
- X509 ... 証明書の形式．
- PEM ... ファイル形式．DER 形式（バイナリ）の証明書を BASE64 でエンコードして，それが何かを表す簡単なヘッダをつけたもの．