PKI unworks

おためし版

第 3 章

証明書を作成する

証明書とはどういったものか、PKI とは何かは大まかに分かってきたかと思います。ここでは、実際に証明書を自作してみて、PKI とはどういったものかを見ていきます。

3.1 makecert.exe

makecert.exe はマイクロソフトの開発環境に含まれている、テスト用の証明書の作成ツールです。Windows 上で動作し、1 コマンドで自己署名証明書や、既存の証明書を使って生成した証明書に電子署名までかけることができます。

入手方法として最も手軽なのは、以下ダウンロードページより **Windows SDK for Windows 8.1** を入手することです。

図 3.1 `https://msdn.microsoft.com/ja-jp/windows/desktop/bg162891`

Windows SDK for Windows 8.1 のインストーラを起動、**Windows Software Development Kit** をインストールすると、c:\Program Files (x86)\Windows Kits\8.1\bin\{x64,x86}\ 以下に配置されます。[*1]

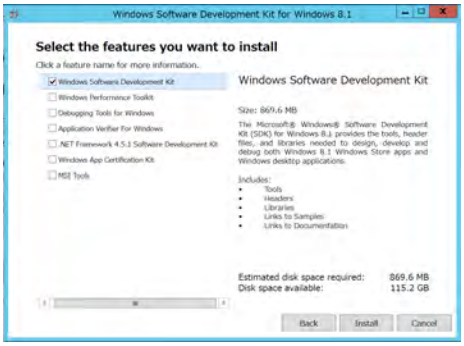

図 3.2　Windows Software Development Kit を選択

では、makecert.exe を用いて自己署名証明書を作ってみます。

```
c:\> makecert.exe -n "CN=www.example.jp"  -sky exchange  -ss My -pe -r
```

-n にて Subject に記載する内容を指定します。最低限、CommonName だけが入っていれば使用可能です。**-sky** は鍵の用途を指定するもので、**exchange** で SSL など暗号化通信のためを指定しています。それ以外に電子署名を意味する **signature** があります。**-ss** は証明書の保存先で、**My** は「個人」の証明書ストアに格納されます。**-pe** にて秘密鍵をエクスポート可能にしておきます。これを指定しておかないと証明書ストアから秘密鍵を取り出せません。最後の**-r** は自分自身で電子署名を行い自己署名証明書とする指定です。

これで、とりあえず HTTPS で使用できる証明書が作成され、証明書ストアに格納されます。必要に応じて証明書ストアから証明書を DER 形式で、あるいは証明書と秘密鍵のセットを PKCS#12 形式でエクスポートし、使用します。

[*1] Windows10 の場合でも、Windows SDK for Windows 8.1 をインストールします。なお、Windows SDK for Windows 8 で、8.1 でない場合はなぜか makecert.exe が付属していません。かならず 8.1 の方を選択してください。

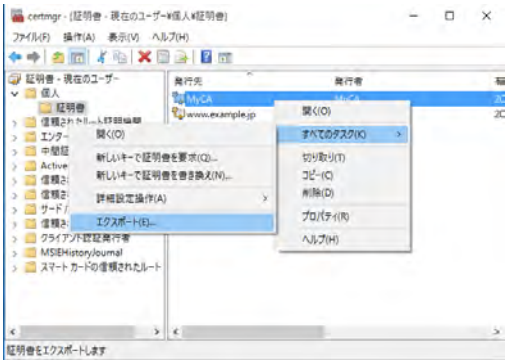

図 3.3 証明書をエクスポート

すこし頑張った例が以下になります。

```
c:\> makecert -n "CN=MyCA" -sky signature  -ss My -pe -r
c:\> makecert -n "CN=www.example.jp" -sky exchange -ss My -pe
 -in "MyCA" -is My
```

1 行目のコマンドで CommonName が MyTestCA の自己署名証明書を作成します。これは電子署名のためのものです。そして 2 行目でサーバ証明書を作成してますが、自己署名証明書ではなく、**-is** で指定した個人の証明書ストアに格納されている、**-in** で指定した CommonName を持つ証明書で電子署名がされます。このサーバ証明書は自己署名証明書ではありません。1 行名の証明書をルート CA としてブラウザや OS に登録すると、2 行目のサーバ証明書は正当な証明書として認証されます。ここで、最低限の PKI の仕組みが実現されております。*2

makecert.exe はテスト用の証明書作成ツールにしては高機能で、この通り最低限の PKI を構築できます。ただ、拡張用途を設定できないなど細かいところで機能が不足しているのが残念なところです。

*2 厳密には、ルート CA に必要な制約条件が記載されないという限界があります。

3.2　キーチェーンアクセスでの証明書作成

「2.4.3 OS X の場合」 でも使用したキーチェーンアクセスですが、**証明書アシスタント** という、証明書を作成する仕組みも備わっています。

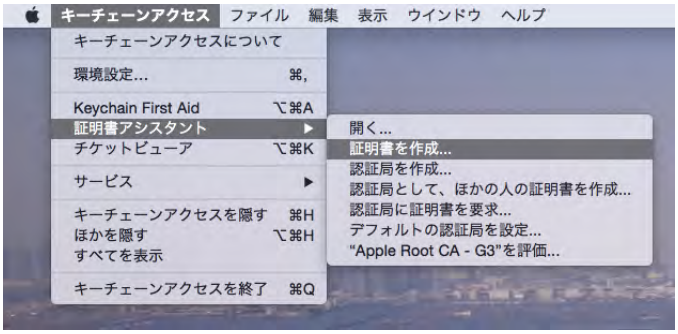

図 3.4　キーチェーンのアプリケーションメニューより、証明書アシスタント

自己署名証明書を作るだけでしたら、証明書アシスタントのサブメニューから「証明書を作成」を選択、サーバ証明書なら名前にホスト名、固有名のタイプを自己署名ルートにしたまま、証明書のタイプに「SSL サーバ」を選択します。あと、作成を押すだけでサーバ証明書が作成されます。

図 3.5　自己署名証明書の作成

作成された証明書は「ログイン」キーチェーンに格納されています。

図 3.6　作成された証明書はログインキーチェインに格納される

証明書を選び「書き出す」から PEM 形式の証明書や、PKCS#12 形式で対になる鍵と証明書をあわせて 1 ファイルに書き出すことが可能です。Windows の証明書ストアと同じで、PKCS#12 形式で書き出す際にはパスワードを設定する必要があります。なお、拡張子は .pfx ではなく .p12 である点にご注意ください。

キーチェーンアクセスの証明書アシスタントでできる事は以下の通りです。

- 自己署名証明書の作成
- 証明書発行要求を作成
- ルート CA や中間 CA を作成
- CA として証明書発行要求に電子署名を施し証明書を作成

一方、証明書発行要求や証明書の属性がほとんど操作できません。例えば X.509v3 の SAN は作られますが CommonName と同じ名前ひとつだけしか記入できない、証明書の有効期限も 1 年に固定です。GUI の簡単操作で証明書が作成できる反面、属性があまりいじれないことからアプリケーションの開発向けなど限定した使い方に特化されているという感じです。

3.4 OpenSSL で自己署名証明書を作成

テスト用の証明書から実際の証明書作成まで、広く使われているのが OpenSSL です。Linux や OS X では標準で搭載されておりすぐ使えるという強みもあります。[*4] Windows でも別途インストールすることで割と簡単に利用できます。[*5]

自己署名証明書を作るだけでしたら、以下になります。

リスト 3.1: OpenSSL での自己署名証明書の作成手順

```
$ openssl genrsa  -out server.key
$ openssl req -new -key server.key -out server.csr
$ openssl x509 -req -in server.csr -signkey server.key -out server.crt -days 3650
```

リスト 3.1 の 1 行目、 **genrsa** サブコマンドは RSA 暗号の鍵を生成します。-out オプションで server.key というファイルに鍵を格納します。なお、この server.key ファイルには公開鍵と秘密鍵がどちらも入っています。つづく 2 行目の **req** サブコマンドは **証明書発行要求**（Certificate Signing Request）と呼ばれる、CA に提出し電子署名をしてもらうデータに関する処理をおこなうもので、-new により新規に生成を指定しております。実際は、次の実行例のように Subject に相当する部分の手入力を求められます。

[*4] ただし、OS X では Security.framework が SSL や証明書関連の処理を担っており、OS も標準アプリも、また Cocoa や CoreService を正しく実装したアプリケーションも OpenSSL を全く使っていません。OpenSSL はあくまで UNIX から移植されたツール向けに用意しているだけ、というスタンスです。Marvericks や Yosemite といった最近の OS X の変化を見ると、いつまで OpenSSL が標準搭載されるかは微妙なところではあります。

[*5] Windows で OpenSSL を利用する場合は、「3.6 Windows での OpenSSL のインストール」 を参照してください。

証明書発行要求作成の実行例

```
$ openssl req -new -key server.key -out server.csr
Loading 'screen' into random state - done
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) [AU]:JP
State or Province Name (full name) [Some-State]:Shiga
Locality Name (eg, city) []:Kusatsu
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Nagisaworks
Organizational Unit Name (eg, section) []:Nashibatake, Dept.
Common Name (e.g. server FQDN or YOUR name) []:nagisaworks.com
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Country Name から最後の行までが入力を求められるところで、Subject に含める C, ST, L, O, OU, CN といった要素の入力を求められています。なお、証明書発行要求は慣習的に .csr という拡張子をつけるようになってます。

リスト 3.1 の 3 行目、x509 が証明書関連のサブコマンドで、ここでは -req で入力が証明書発行要求であることを指摘して、-in で先ほどの証明書発行要求ファイルを指定、-signkey で電子署名を行う鍵を格納したファイルを指定、そして -out で電子署名がなされたファイルの出力先を指定しています。ここでは、-signkey で自分自身の鍵ファイルを指定しているので自己署名証明書が作られてしまいます。慣習に従い拡張子 .crt で証明書ファイルを作成しましたが、拡張子は別に .pem でも構いません。デフォルトでは PEM 形式が生成されますが、-outform オプションを使用して DER バイナリ形式で出力しても構いません。最後に記載されている -days オプションは有効期限を指定するもので、 3650 、つまり本日から見て 10 年後までを有効としています。テスト用なら充分ですね。

最初に作成した server.key 鍵ファイルと、最後にできた server.crt 証明書ファイルの二つが実際に使用されます。server.csr ファイルは使用しませんので削除しても構いません。

ここでは、-out オプションを使って出力ファイルを全て指定しました。-out

を指定しない場合は結果を標準出力に出力しますので、リダイレクトを使ってファイルに書き込ませる事もできます。Linux や OS X など UNIX 系の環境、Windows でも CMD.EXE をシェルにしている場合はリダイレクトを使っても問題ありません。しかし、Windows でシェルを PowerShell にしてしまった方には注意が必要です。

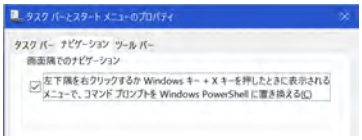

図 3.7　タスクバーのプロパティからシェルを PowerShell に設定できます。

OpenSSL は PEM 形式などテキストファイルを入力に取る際、改行が LF だけの、ASCII もしくは UTF-8(BOM なし) [*6] を想定しております。一方、PowerShell で「>」を使ったリダイレクトでファイルを書き出す際は、内部的に Out-File コマンドレットが動作しており、UTF-16 で CR+LF 区切りのファイルが作成されてしまいます[*7]。`| Out-File -Encoding utf8 .\server.key` といった感じで明示的に Out-File コマンドレットを呼び出すことで文字コードは UTF-8 にできますが、BOM がつくことや改行コードが CR+LF になるのは回避できません。リダイレクトを使用せず、 リスト 3.1 のように出力 OpenSSL に任せておけば、こんな問題は発生しません。

[*6] BOM は ByteOrderMark の略で、UTF-16 など 2 バイト 固定サイズの文字コードを扱う際、リトルエンディアンかビックエンディアンかどちらで格納したかを識別するのに使用されます。具体的にはテキストファイルの先頭に 0xfeff を書き込んでおき、2 バイト読み取こんで 0xfffe と読まれたらそれは逆のエンディアンで格納されていると考え、以後読み込んだバイトをひっくり返して取り扱います。BOM そのものおよび、仮に文中に 0xfeff (0xfffe) が登場した場合、幅 0 の区切り文字ではない空白、として扱われる、つまりは無視されます。8bit 可変サイズの UTF-8 の場合、1 バイトずつ読み込むためエンディアンにまつわる問題がなく、従って BOM を先頭につける必要はないのですが、メモ帳など Windows の標準環境では UTF-8 を指定してテキストフィアルを保存すると先頭に「0xef 0xbb 0xbf」BOM を UTF-8 で使われる方法で展開した 3 バイトが書き込まれます。これはファイルの先頭が「#!」(Shebang) であることを前提としている UNIX のスクリプトファイルなどと干渉します。一言で言って余計なことをしてくれた、わけです。Windows 界隈ではこの BOM ありの UTF-8 のテキストを単に UTF-8 と呼び、BOM をつけないようにした UTF-8 のテキストを UTF-8N と N を付けたり、BOM なし UTF-8 と呼ぶ慣習があります。

[*7] UNIX でも Windows でも、リダイレクトの処理はシェルが行います。

3.5 OpenSSL での証明書の表示

PEM 形式で作成された証明書の場合、テキストのため notepad や TextEdit.app といったテキストエディタで表示させることも可能です。リスト 2.1 にあるような属性の表示がある場合もあれば、--BEGIN CERTIFICATE-- からの Base64 でエンコードされた実体しか存在しない場合もあります。属性まで表示させたい場合は、以下のコマンドで可能です。

```
$ openssl x509 -text -in server.crt -inform PEM
```

x509 サブコマンドで -text オプションを指定すると、属性の表示まで含む、リスト 2.1 にあるような証明書の表示を行います。-inform PEM は -in で指定したファイルの形式 (PEM, DER もしくは NET) を指定します。PEM の場合、省略が可能なはずですが、指定をしておいた方が安心です。

同じように、鍵ファイルの中身を表示するには以下のコマンドが利用できます。

```
$ openssl rsa -text -in server.key
```

証明書発行要求に何を設定したかを確認したい場合は、以下のコマンドになります。

```
$ openssl req -text -in server.csr
```

第 4 章

OpenSSL で PKI

前章では Makecert.exe, キーチェーンアクセス、そして OpenSSL を使って自己署名証明書を作ってみました。ここでは、もう少し頑張って OpenSSL で独自 CA を作って運用してみます。

4.1 CA.sh

そもそも OpenSSL の配布物には CA.sh というシェルスクリプトが含まれており、CA として必要な操作がまとめられております。が、CA.sh はいまいち使い勝手がよろしくないので、ここでは openssl コマンドを直接使用し、いくつかの手順に分けて説明していきます。

4.2 openssl.cnf

openssl コマンドは openssl.cnf [*1]という定義ファイルからデフォルトの設定を読み込み、動作しております。リスト 3.1 の 2 行目のコマンドでは Subject の内容を手入力しましたが、ここで国名の AU や組織名の Internet Widgits Pty Ltd などは openssl.cnf に記載されています。毎回同じ入力をするなら openssl.cnf の設定を書き換えた方が手っ取りばやいです。

[*1] OpenSSL for Win32 では openssl.cfg になってたりします。これは、.cnf という拡張子がダイアルアップによるインターネット接続時代に使われていた短縮ダイアルファイルの拡張子として登録されており混乱が発生したためです。同様の問題は Windows 向け MySQL の my.cnf でも起きております。

PKI unworks. 68ページ程度?

C89 12/31 木曜日 東ム45b にて 頒布予定です。