Javaアプリケーションのテスト用に証明書チェーンを生成

OpenSSLツールキットを使いこなす

Paul Abbott (pabbott@hursley.ibm.com)
Software Engineer
IBM

2004年 8月 18日

ソフトウェアをテストするためにデジタル証明書チェーンを作成する方法を学びましょう。あらゆる長さの証明書チェーンを作成するために、簡単に入手可能なOpenSSLツールキットをどのようにして使用するかを説明しながら、このあまり文書化される機会に恵まれないプロセスをPaul H. Abbott（IBMソフトウェア技師）は明確にします。それから通常の証明書属性(certificate attributes)を説明し、Java keystore に作成した証明書を読み込むJavaコードのサンプルを示します。

セキュリティー用のPKI（Public Key Infrastructure）に依存するJavaソフトウェアを開発しているのであれば、テストの目的のために（認証パスとしても知られている）デジタル証明書チェーンを作成する必要が頻繁にあります。これは比較的単純なタスクですが、それを説明する明瞭な文書はまれです。この記事は、オープン・ソースのOpenSSLツールキット（参考文献）を使用して、どのように任意の長さの証明書チェーンを作成するかを説明します。また、一般的な証明書属性について学び、Javakeystoreに証明書を読み込むサンプル・プログラムを考察します。

デジタル証明書: 簡単な概観

この記事では読者がPKIの基本について詳しいと言う前提で話を進めます。証明書チェーンの概念を明確にするために、デジタル証明書の構造そして目的の簡単な概観について軽くおさらいをします。

デジタル証明書の主な使用目的は、E-mailやJARファイルのような署名付きデータの発信元を検証することです。証明書で署名付きデータを検証することは、受信者にデータの発信元を知らせそしてデータが通過中に変更されたかどうかを知らせます。

高水準では、デジタル証明書はDN（Distinguish Name、識別名）と公開鍵を含みます。証明書の公開鍵とマッチングする秘密鍵を握るエンティティー（例えば、人間）を、Distinguish Nameは識別します。秘密鍵で証明書に署名をして証明書に署名を配置することにより、その2つをまとめます。

証明書の公開鍵とマッチングする秘密鍵に署名された証明書は、自己署名証明書(self-signed certificate)として知られます。ルート認証局証明書は、この分類に属します。ユーザー証明書は（認証局の秘密鍵のような）異なる秘密鍵により署名されますが、それはよくあることです。これらは2連鎖の証明書チェーンを構成します。ユーザー証明書の正統性を検証する行為は、その証明書からの（認証局の公開鍵を必要とする）署名の検証と関連します。しかし、認証局の公開鍵が使用可能になる前に、それを格納する認証局証明書が検証されるべきです。認証局証明書は自己署名されていますので、証明書を検証するのに認証局の公開鍵を使用します。

ルート認証局の秘密鍵によりユーザー証明書が署名される必要はありません。認証局の秘密鍵により署名された証明書を持つ仲介者の秘密鍵により署名されるかも知れません。「ユーザー証明書、仲介証明書、そして認証局証明書」は、3連鎖の証明書チェーンの一例です。複数の仲介者がチェーンの一部を成す可能性がありますので、証明書チェーンはどのような長さにもなり得ます。

また別の注目すべき点は、証明書が拡張領域(extensions)と言う形で追加的な情報を含むことが可能なことです。特に、拡張領域は証明書の用途を指定できます。証明書の使用用途によっては、拡張領域は特に重要な役割を果たします。

OpenSSlで証明書を作成

証明書作成を支援するツールが数多く存在します。Java SDKと共に出荷されるコマンド行ツール（keytool）を使用して自己署名証明書を作成できます。しかし、証明書チェーン作成にはOpenSSLのようなより複雑なソフトウェアを必要とします。

OpenSSLを入手

OpenSSLを無料でダウンロードできます（参考文献を参照）。UNIXを使用しているのであれば、OpenSSLはO.S.に既にインストールされているか、インストールのオプションとしてO.S.に付属しているでしょう。Linuxのユーザーは、「/usr/share/ssl」を参照すべきです。Microsoft MKS ツールキットもOpenSSLのバージョンと共に出荷されます（著者のマシンでは、「C:\Program Files\MKS Toolkit\etc\openssl」でした）。もしもWindowsを使用しているのにもかかわらずMKSを所有していないのであれば、SourceForge（参考文献を参照）から必要なバイナリーを取得できます。大半のインストールは3種類の主なファイルから構成されます（OpenSSLバイナリー、CA.shシェル・スクリプト、そしてopenssl.cnf構成ファイル）。（SourceForgeから入手可能なパッケージにはCA.shファイルがありません。ソースのバンドルをダウンロードすることにより、その足りないファイルを取得できます。）

インストール終了後、CA.shとOpenSSL実行可能ファイルがパスにあることを確認してください。ここでルート証明書を作成する準備が整いました。

ルート証明書を作成

認証局のシェル・スクリプトは、ルート証明書の作成作業を比較的簡単にします。先ず、認証局のデータ・ストアを入れたいディレクトリー（ここではtemp\OpenSSLを使用します。）へと変更します。次に、以下の項目を入力します。

```
CA -newca
```

リスト1にあるような（プロンプトにて入力したサンプルの情報を含む）ダイアログが結果として得られます。

リスト1. ルート証明書を作成

```
$ CA.sh -newca
CA certificate filename (or enter to create)

Making CA certificate ...
Using configuration from C:/PROGRA~1/MKSTOO~1/etc/openssl/openssl.cnf
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
......++++++
...++++++
writing new private key to './demoCA/private/./cakey.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
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.
You will see a number of 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]:UK
State or Province Name (full name) [Some-State]:Hampshire
Locality Name (e.g., city) []:Winchester
Organization Name (e.g., company) [Internet Widgits Pty Ltd]:IBM UK Ltd
Organizational Unit Name (e.g., section) []:JTC
Common Name (e.g., YOUR name) []:Pauls Root Certificate
Email Address []:Paul_Abbott@uk.ibm.com
$
```

ダイアログの完成と共に、OpenSSLは下記のディレクトリー構造を作成します。

```
demoCA/
   cacert.pem              - root certificate
   index.txt               - empty
   serial                  - text file containing "01"
   certs/                  - empty
   crl/                    - empty
   newcerts/               - empty
   private/cakey.pem       - private key
```

ルート・ディレクトリーの概要を簡単に示します。

- cacert.pem は、この認証局のPEM（サイドバー、PEMファイル・フォーマットを参照してください）でエンコードされたルート証明書です。ルート証明書は、ルートの秘密鍵に署名された証明書を検証します。
- index.txt は、発行された証明書全てのリストを含むファイルです。
- serial は（この認証局から発行された証明書に割り当てられる）次に使用可能なシリアル番号を保持します。別の言い方をすれば、これは署名する要求がこのルート証明書により署名された場合に証明書に割り当てられる固有のシリアル番号です。
- cakey.pem は、ルートの秘密鍵です。証明書要求に署名するときに、この鍵を使います。これもPEMにエンコードされています。

ディレクトリー名（この例ではdemoCA）そしてルート証明書の有効期間（デフォルトは365日間）を、CA.shにて定義します。これらの値を変更したいのであれば、このファイルを編集すべきです。

ユーザー証明書を生成

PEMファイル・フォーマット

OpenSSLは、証明書と鍵を保管するのにPEMファイル・フォーマットを使用します。証明書ファイルの場合、PEMとは本質的に
`-----BEGIN CERTIFICATE-----`
のような開始行や
`-----END CERTIFICATE-----`
のような終了行に挟まれたBase64エンコードされたバイナリーのことを言います。これらのマーカーの外側には、エンコードされたコンテントのテキスト表記のような追加的な情報があるかも知れません。ASCIIファイルですので、どのテキスト・エディターを使ってでも開けます。この PEMファイルの例をご覧ください。

PEMファイルの例

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=UK, ST=Hampshire, L=Winchester, O=IBM, OU=JTC, CN=Paul H Abbott/
Email=Paul_H_Abbott@uk.ibm.com
        Validity
            Not Before: Dec 19 15:17:27 2003 GMT
            Not After : Dec 18 15:17:27 2004 GMT
        Subject: C=UK, ST=Hampshire, L=Winchester, O=Private, OU=Home, CN=Paul Abbott/
Email=Paul.Abbott@bcs.org.uk
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b9:64:00:40:ad:d1:06:45:a2:ae:fb:32:91:64:
                    04:4b:17:23:fa:cf:36:64:17:9a:51:9a:28:46:c9:
                    db:91:ea:d0:ee:72:2c:81:31:ba:35:a6:c6:ce:d8:
                    6d:3d:77:46:57:aa:bd:33:28:19:e4:d6:4f:13:f6:
                    74:af:5f:3c:ee:b8:7f:03:b4:13:53:aa:df:b5:7d:
                    fc:ad:81:4d:c0:11:85:28:82:da:d9:bf:22:e4:6f:
                    34:04:42:9b:d6:f3:da:20:88:b5:38:4f:21:fc:d5:
                    dc:5e:00:dc:eb:19:c6:31:ad:78:88:24:4e:40:38:
                    30:73:80:a7:55:1e:f6:30:9d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: CA:FALSE
            Netscape Comment: OpenSSL Generated Certificate
            X509v3 Subject Key Identifier: F5:7E:D6:46:75:AA:7B:20:7D:B1:6D:96:6B:F9:24:A4:B9:15:A6:55
            X509v3 Authority Key Identifier:
 keyid:D2:90:DA:BA:17:9A:0D:56:21:F8:56:22:04:23:37:C3:72:7C:CC:FE
                DirName:/C=UK/ST=Hampshire/L=Winchester/O=IBM/OU=JTC/CN=Paul H Abbott/
Email=Paul_H_Abbott@uk.ibm.com
                serial:00
    Signature Algorithm: md5WithRSAEncryption
        08:ec:99:13:58:2b:99:b9:77:b8:80:c5:3b:2f:73:e5:6c:97:
        a8:6b:8c:1f:a2:cf:23:48:6c:df:de:d1:23:64:b0:fd:73:bb:
        31:c4:28:99:6a:b1:0f:d5:d7:da:ab:ac:e5:31:85:ff:46:32:
        5b:0d:83:4a:d6:b5:41:a8:4e:c3:8e:2d:2b:2e:f6:ef:a0:ab:
        9a:26:0c:87:16:c8:d4:4f:f2:e6:09:a3:2b:02:5c:8d:ea:b7:
        c2:3c:e1:eb:22:75:7b:68:5e:de:4c:5d:40:59:06:a2:d5:d5:
        75:f6:fe:ac:5a:e7:8d:62:56:9a:fd:15:c6:a0:4a:4a:14:1c:
        c8:62
```

```
-----BEGIN CERTIFICATE-----
MIIDvzCCAyigAwIBAgIBATANBgkqhkiG9w0BAQQFADCBkzELMAkGA1UEBhMCVUsx
EjAQBgNVBAgTCUhhbXBzaGlyZTETMBEGA1UEBxMKV2luY2hlc3RlcjEMMAoGA1UE
ChMDSUJNMQwwCgYDVQQLEwNKVEMxFjAUBgNVBAMTDVBhdWwgSCBBYmJvdHQxJzAl
BgkqhkiG9w0BCQEWGFBhdWxfSF9BYmJvdHRAdWsuaWJtLmNvbTAeFw0wMzEyMTkx
NTE3MjdaFw0wNDEyMTgxNTE3MjdaMIGTMQswCQYDVQQGEwJVSzERMA8GA1UECBMI
SGFtc2hpcmUxEzARBgNVBAcTCldpbmNoZXN0ZXIxEDAOBgNVBAoTB1ByaXZhdGUx
DTALBgNVBAsTBEhvbWUxFDASBgNVBAMTC1BhdWwgQWJib3R0MSUwIwYJKoZIhvcN
AQkBFhZQYXVsLkFiYm90dEBiY3Mub3JnLnVrMIGfMA0GCSqGSIb3DQEBAQUAA4GN
ADCBiQKBgQC5ZABArdEGRaKu+zKRZARLFyP6zzZkF5pRmihGyduR6tDuciyBMbo1
psbO2G09d0ZXqr0zKBnk1k8T9nSvXzzuuH8DtBNTqt+1ffytgU3AEYUogtrZvyLk
bzQEQpvW89ogiLU4TyH81dxeANzrGcYxrXiIJE5AODBzgKdVHvYwnQIDAQABo4IB
HzCCARswCQYDVR0TBAIwADAsBglghkgBhvhCAQ0EHxYdT3BlblNTTCBHZW5lcmF0
ZWQgQ2VydGlmaWNhdGUwHQYDVR0OBBYEFPV+1kZ1qnsgfbFtlmv5JKS5FaZVMIHA
BgNVHSMEgbgwgbWAFNKQ2roXmg1WIfhWIgQjN8NyfMz+oYGZpIGWMIGTMQswCQYD
VQQGEwJVSzESMBAGA1UECBMJSGFtcHNoaXJlMRMwEQYDVQQHEwpXaW5jaGVzdGVy
MQwwCgYDVQQKEwNJQk0xDDAKBgNVBAsTA0pUQzEWMBQGA1UEAxMNUGF1bCBIIEFi
Ym90dDEnMCUGCSqGSIb3DQEJARYYUGF1bF9IX0FiYm90dEB1ay5pYm0uY29tggEA
MA0GCSqGSIb3DQEBBAUAA4GBAAjsmRNYK5m5d7iAxTsvc+Vsl6hrjB+izyNIbN/e
0SNksP1zuzHEKJlqsQ/V19qrrOUxhf9GMlsNg0rWtUGoTsOOLSsu9u+gq5omDIcW
yNRP8uYJoysCXI3qt8I84esidXtoXt5MXUBZBqLV1XX2/qxa541iVpr9FcagSkoU
HMhi
-----END CERTIFICATE-----
```

ユーザー証明書の生成は、2つのステップ（要求の生成、そして要求の署名）を介します。認証局のシェル・スクリプトは`-newreq`（新規の要求を生成）と`-sign`（新規の要求を署名）のオペレーターを使用して両方のステップを実行できます。

新規の要求を生成

`CA -newreq`コマンドの実行は、新規のルート証明書を生成するために入手したものと類似するダイアログから始めます。重要な相違点のひとつとして、（出力ファイルに書き込む前に）OpenSSLが秘密鍵をエンコードするときに使用するPEMパスフレーズ（パスワード）をダイアログが促すことが挙げられます。

newreq.pemと名付けられた出力ファイルは、署名されていない証明書として見なせられる署名する要求と秘密鍵を含みます。リスト2は、新規要求ダイアログの一例を示します。

リスト2. -newreqダイアログの例

```
Using configuration from
C:/PROGRA~1/MKSTOO~1/etc/openssl/openssl.cnf
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
.....................++++++
...++++++
writing new private key to 'newreq.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
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.
You will see a number of 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]:UK
State or Province Name (full name) [Some-State]:Hampshire
Locality Name (e.g., city) []:Winchester
Organization Name (e.g., company) [Internet Widgits Pty Ltd]:IBM Uk Ltd
Organizational Unit Name (e.g., section) []:JET
```

```
Common Name (e.g., YOUR name) []:Paul Abbott
Email Address []:Paul_H_Abbott@uk.ibm.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:qwerty
An optional company name []:
Request (and private key) is in newreq.pem
```

要求を署名

private/cakey.pemにて保持されたルート認証局の秘密鍵を使用することにより、`CA -sign`コマンドの実行は要求を署名します。その要求はnewreq.pemと呼ばれるファイルにあるべきであり、newcert.pemと呼ばれるファイルに生成された証明書は書き込まれます。ファイルは両方とも現行ディレクトリーにあります。要求を署名するダイアログのサンプルを、リスト3に示します。

リスト3. -signダイアログの例

```
$ CA.sh -sign
Using configuration from C:/PROGRA~1/MKSTOO~1/etc/openssl/openssl.cnf
Loading 'screen' into random state - done
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName           :PRINTABLE:'UK'
stateOrProvinceName   :PRINTABLE:'Hampshire'
localityName          :PRINTABLE:'Winchester'
organizationName      :PRINTABLE:'IBM UK Ltd'
organizationalUnitName:PRINTABLE:'JET'
commonName            :PRINTABLE:'Paul Abbott'
emailAddress          :IA5STRING:'Paul_H_Abbott@uk.ibm.com'
Certificate is to be certified until Jun 22 20:50:55 2005 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
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=UK, ST=Hampshire, L=Winchester, O=IBM UK Ltd,
          OU=JTC, CN=Paul H Abbott/Email=Paul_H_Abbott@uk.ibm.com
        Validity
            Not Before: Jun 22 20:50:55 2004 GMT
            Not After : Jun 22 20:50:55 2005 GMT
        Subject: C=UK, ST=Hampshire, L=Winchester, O=IBM Uk Ltd,
          OU=JET, CN=Paul Abbott/Email=Paul_H_Abbott@uk.ibm.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:e2:b5:90:50:e5:dd:7c:79:c3:49:a5:c9:ee:29:
                    3a:da:1d:8b:6a:b0:0b:a0:a1:cf:79:fc:be:50:db:
                    cb:37:b7:54:00:bb:6e:74:e6:a4:11:d4:c6:5a:02:
                    46:3b:b4:33:72:97:5b:cf:9d:9a:32:9b:e6:34:e9:
                    4b:30:4e:b6:68:55:8a:3f:80:f3:5e:c9:63:cc:4e:
                    c2:c0:c3:34:2f:93:9f:fa:ca:1b:44:f5:c8:87:ec:
                    1d:12:a9:8c:3a:b9:28:83:4d:b5:18:ff:34:3a:a9:
                    e7:7e:4e:c4:21:8e:56:e7:dc:f5:07:46:39:c8:d8:
                    ff:00:d3:87:20:2e:06:18:19
                Exponent: 65537 (0x10001)
```

```
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                FA:65:44:FB:3A:E6:15:1F:C5:40:6C:EB:F4:DA:AB:B9:CD:F2:2D:54
            X509v3 Authority Key Identifier:
                keyid:93:8C:B4:F0:30:95:77:59:2E:A1:3B:0C:A5:3A:C6:92:FA:16:31:6D
                DirName:/C=UK/ST=Hampshire/L=Winchester/O=IBM UK Ltd/
                  OU=JTC/CN=Paul H Abbott/Email=Paul_H_Abbott@uk.ibm.com
                serial:00

    Signature Algorithm: md5WithRSAEncryption
        27:43:6d:89:c3:61:d4:af:3e:dc:55:a3:9a:a7:7d:66:4e:29:
        2e:43:f0:90:c6:9c:0f:62:24:b2:4c:9e:2c:f7:d7:84:ce:7f:
        b6:c8:09:3d:b4:80:c8:26:9a:a8:6b:2f:df:8f:e3:8b:80:f5:
        10:28:80:28:5e:94:55:be:61:e5:18:4e:d4:a8:c2:9e:6d:9b:
        52:64:94:33:b3:a5:68:79:e2:85:86:01:e6:aa:0f:1e:54:2d:
        80:b1:37:38:66:cc:09:9a:0e:30:0a:e8:b9:00:7d:da:a2:a1:
        bb:3c:83:37:2f:16:6a:5d:84:25:66:23:d2:67:a9:02:a4:33:
        96:56
-----BEGIN CERTIFICATE-----
MIID0jCCAzugAwIBAgIBAjANBgkqhkiG9w0BAQQFADCBmjELMAkGA1UEBhMCVUsx
EjAQBgNVBAgTCUhhbXBzaGlyZTETMBEGA1UEBxMKV2luY2hlc3RlcjETMBEGA1UE
ChMKSUJNIFVLIEx0ZDEMMAoGA1UECxMDSlRDMRYwFAYDVQQDEw1QYXVsIEggQWJi
b3R0MScwJQYJKoZIhvcNAQkBFhhQYXVsX0hfQWJib3R0QHVrLmlibS5jb20wHhcN
MDQwNjIyMjA1MDU1WhcNMDUwNjIyMjA1MDU1WjCBmDELMAkGA1UEBhMCVUsxEjAQ
BgNVBAgTCUhhbXBzaGlyZTETMBEGA1UEBxMKV2luY2hlc3RlcjETMBEGA1UEChMK
SUJNIFVrIEx0ZDEMMAoGA1UECxMDSkVUMRQwEgYDVQQDEwtQYXVsIEFiYm90dDEn
MCUGCSqGSIb3DQEJARYYUGF1bF9IX0FiYm90dEB1ay5pYm0uY29tMIGfMA0GCSqG
SIb3DQEBAQUAA4GNADCBiQKBgQDitZBQ5d18ecNJpcnuKTraHYtqsAugoc95/L5Q
28s3t1QAu2505qQR1MZaAkY7tDNyl1vPnZoym+Y06UswTrZoVYo/gPNeyWPMTsLA
wzQvk5/6yhtE9ciH7B0SqYw6uSiDTbUY/zQ6qed+TsQhjlbn3PUHRjnI2P8A04cg
LgYYGQIDAQABo4IBJjCCASIwCQYDVR0TBAIwADAsBglghkgBhvhCAQ0EHxYdT3Bl
blNTTCBHZW5lcmF0ZWQgQ2VydGlmaWNhdGUwHQYDVR0OBBYEFPplRPs65hUfxUBs
6/Taq7nN8i1UMIHHBgNVHSMEgb8wgbyAFJOMtPAwlXdZLqE7DKU6xpL6FjFtoYGg
pIGdMIGaMQswCQYDVQQGEwJVSzESMBAGA1UECBMJSGFtcHNoaXJlMRMwEQYDVQQH
EwpXaW5jaGVzdGVyMRMwEQYDVQQKEwpJQk0gVUsgTHRkMQwwCgYDVQQLEwNKVEMx
FjAUBgNVBAMTDVBhdWwgSCBBYmJvdHQxJzAlBgkqhkiG9w0BCQEWGFBhdWxfSF9B
YmJvdHRAdWsuaWJtLmNvbYIBADANBgkqhkiG9w0BAQQFAAOBgQAnQ22Jw2HUrz7c
VaOap31mTikuQ/CQxpwPYiSyTJ4s99eEzn+2yAk9tIDIJpqoay/fj+OLgPUQKIAo
XpRVvmHlGE7UqMKebZtSZJQzs6VoeeKFhgHmqg8eVC2AsTc4ZswJmg4wCui5AH3a
oqG7PIM3LxZqXYQlZiPSZ6kCpDOWVg==
-----END CERTIFICATE-----
Signed certificate is in newcert.pem
```

ダイアログの途中にて要求されるPEMパスワードが、（ユーザーのではなく）ルート認証局の秘密鍵を暗号化するのに使われるパスワードであることに注意してください。

署名後のdemoCAディレクトリーの検査は、index.txtとserialのファイルがアップデートされたことを示します。そのシリアル番号を反映するファイル名（例えば、01.pem）として生成された公開鍵がdemoCA/newcert/ディレクトリーに配置されています。

この時点では、ユーザー証明書、ユーザー秘密鍵、そしてルート証明書を所持していることになります。これだけで済むのであれば、ここで全てが終わります。もしも証明書のコンテントを制御したり3連鎖以上の証明書チェーンを作成する方法を学びたいのであれば、ここから先に進んでください。

CA.shの水面下

ここまでで、証明書の生成のプロセスを簡略化するために、どのようにCA.shシェル・スクリプトを使用するかを説明しました。しかし水面下では、全ての必要とされる鍵の生成と署名の操作を実行するためにCA.shはOpenSSLコマンドを使用します。openssl.cnf構成ファイルを介してOpenSSLを操作します。このファイルのセクションの一部そして（影響を受ける）CA.shの操作について考察します。

CA -newca

前述のとおり、`CA -newca`のオペレーションは認証局に必要なディレクトリー構造を作成します。

```
dir             = ./demoCA              # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key
```

CA.shは構成ファイルに直接参照しないのですが、openssl.cnfのCA_defaultセクションにて定義されるディレクトリーをこのオペレーションは作成します。別の異なる構造を使用したいのであれば、openssl.cnfを修正し必要とされるディレクトリーを手動で作成する必要があります。別の方法として、ディレクトリーを作成するためにCA.shを修正することも可能です。

下記に示されるOpenSSLへの呼び出しは、ルート証明書を生成します。

```
openssl req -new -x509 -keyout ./demoCA/private/cakey.pem
  -out ./demoCA/cacert.pem -days 365
```

`-days`フラグは、ルート証明書の有効期間を365日間（1年間）に設定し、それはopenssl.cnfにて指定されたデフォルト値をオーバーライドします。

CA -newreq

下記の例に示されるとおり、openssl.cnfの`req`セクションは新規証明書要求の生成を制御します。

```
[ req ]
default_bits        = 1024
default_keyfile     = privkey.pem
distinguished_name  = req_distinguished_name
attributes          = req_attributes
```

最初の行は生成された鍵の組み合わせの長さを決定し、2行目は生成された秘密鍵のためのデフォルト宛先を決定します。その次の2行は同じファイル内にある別のセクションを参照します。

`req_distinguished_name`セクションは証明書要求にて配置されるDistinguish Nameのコンポーネントを定義し、これらのコンポーネントは最終的には署名された証明書に移行します。

```
[ req_distinguished_name ]
countryName                 = Country Name (2 letter code)
countryName_default         = AU
countryName_min             = 2
countryName_max             = 2

stateOrProvinceName         = State or Province Name (full name)

localityName                = Locality Name (e.g., city)

organizationName            = Organization Name (e.g., company)

organizationalUnitName      = Organizational Unit Name (e.g., section)

commonName                 = Common Name (e.g., YOUR name)

emailAddress                = Email Address
```

上記の例の中の2行目から5行目にて示されるとおり、このセクションのそれぞれの部分は、最大4つの値から成ります。

- 画面に表示されるメッセージのプロンプト
- デフォルト値（接尾部は`_default`）
- ユーザーが入力するのを許可されるキャラクター数の最小値（接尾部は`_min`）
- ユーザーが入力するのを許可されるキャラクター数の最大値（接尾部は`_max`）

他にもいくつかオプションがあります。バッチ操作にて使える、独自のOID（オブジェクトID）のタイプの定義、そして（ユーザーにより供給されると言うよりも）構成ファイルにて定義されるDistinguish Nameの値の所持をも含みます。特に暗号化パスワードの最小そして最大長さの定義を、`req_attributes`セクションは支援します。

新規要求のオペレーションへのOpenSSL呼び出しです。

```
openssl req -new -keyout newreq.pem -out newreq.pem -days 365
```

`-keyout`オプションが構成ファイル内の`default_keyfile`の行をオーバーライドすることに注意してください。ここでもそうなのですが、生成された証明書の有効期間は365日間に設定されています。この行をルート証明書を生成する行と比較した場合、相違点（`-x509`）が1つだけあるのに気付くはずです。これは自己署名証明書を求めていることをOpenSSLに伝えます。

OpenSSL文書の`req`セクションにて署名の要求を生成するために、どのようにしてOpenSSLを使用するかについてのより詳しい情報を求めることができます（参考文献を参照）。

CA -sign

要求の生成のオペレーションと同じく、署名のオペレーションは証明書を署名するためにOpenSSLを呼び出します。

```
openssl ca -policy policy_anything -out newcert.pem
  -infiles newreq.pem
```

OpenSSLコマンドが`sign`と言うよりも`ca`であり、署名にて使用された秘密鍵が指定されていないのは、興味深い話です。openssl.cnf構成ファイルはこの情報を定義します。これをオーバーライドするのに`-name`オプションを使えますが、関連するセクションは`CA_default`です。

`-policy`パラメーターは構成ファイル内のpolicyセクションを指定します。Distinguish Nameの要求にてどのフィールドが存在すべきか、そしてどれがオプションかを指定するのに、このセクションを使います。典型的な例として、ここでは以下の構成を使用します。

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

ここでお解りのとおり、この構成はデフォルトとしてかなり少量しか必要としません。ここでリストされていないDistinguish Nameフィールドが署名された要求から静かに消去されることを、この文書が記述している事実に注目する価値があります。

第三の証明書を署名

この記事の「ルート証明書を作成」そして「ユーザー証明書を生成」の章で、ルート証明書を作成しそのルート証明書を使ってユーザー証明書を署名するためにCA.shシェル・スクリプトを使用する方法を説明しました。2連鎖の証明書チェーンを求めるのであればこのプロセスは適切なのですが、3連鎖以上のチェーンを証明書に求めるのでしたらどうなのでしょうか？OpenSSLコマンドの直接使用が、その解答をもたらします。CA.shスクリプトがどのように機能するかを把握できたわけですので、この知識を活用して3連鎖証明書チェーンを作成できます。踏まえるべき段階は、2連鎖の証明書チェーンの場合と類似します。過程の簡略化のために、前述の例にある認証局の証明書/秘密鍵そしてユーザー・リクエストを再利用します。

まず手始めに、現存するユーザー・リクエスト・ファイル(user-request file)をuserRequest1.pemに改名します。それから、新規のユーザー証明書を作成し、それをuserCert1と名付けます。現存のユーザー証明書は認証局証明書としてフラグを立てられておらず、別の証明書を署名するために使われません。（ そのような証明書を含む証明書チェーンを検証しようとする場合にのみ、これは一目瞭然となります。 ）

証明書の署名に証明書を使用することを可能にする下記のセクションを、構成ファイルに追加する必要が生じます。

```
[ my_v3_ext ]
basicConstraints = CA:true
```

下記のコマンドを使用して証明書を署名しましょう。

```
openssl ca -policy policy_anything -extensions my_v3_ext
  -out userCert1.pem -infiles userRequest1.pem
```

次に、第三の証明書のために新規の証明書要求を生成します。それをするにあたって`CA -newreq`コマンドを使用するか、`OpenSSL`コマンドを直接使用できます。

要求ファイルを一度入手したならば、それを署名すべきです。前述のとおり、要求を署名するのに使われた秘密鍵は、デフォルトとして（`openssl ca` コマンド行ではなく）構成ファイルにて指定されています。このデフォルトをオーバーライドすることを可能にする`-keyfile` オプションを、`openssl ca`コマンドは包括できます。下記のコマンドの署名は第三の証明書を署名するのに第二の証明書を使用します。

```
openssl ca -policy policy_anything -keyfile userRequest1.pem
  -cert userCert1.pem -out userCert2.pem -infiles newreq.pem
```

`-cert`パラメーターが署名者の証明書を指定し、構成ファイルのデフォルトをオーバーライドすることに注目してください。署名者の証明書と秘密鍵が一致することをOpenSSLがチェックするので、-certパラメーター無しのコマンド呼び出しはエラーにつながります。

全てがうまく行ったのだとすれば、現在のディレクトリーにてuserCert1.pem と userCert2.pem ファイルを、そしてdemoCA/privateディレクトリーにてルート認証局証明書ファイルを入手しているはずです。これら3つのファイルは証明書チェーンを構成します。Windows環境で作業をしているのであれば、単にファイル・ブラウザーにてファイルに.cer 拡張領域を与えダブルクリックをしてファイル名を変更することにより、証明書をインストールできます。

他のオプション

下記のOpenSSLコマンド行オプションが役立つことになるかも知れません。

- -startdate, -enddate そして -days は、生成された証明書の妥当性期間を指定するときに使います。デフォルトとして、証明書は現在日付から1年間有効です。
- -notext は、バイナリー・エンコードされたバージョンよりも前に登場する生成された証明書のテキスト表記のスイッチを切ることを可能にします。多くの場合、テキスト・バージョンは必要とされません。
- -key は、秘密鍵の暗号化に使われるパスワードです。バッチ・ファイルからOpenSSLを呼び出したい場合に、これは役立ちます。
- -batch は、（ユーザー・プロンプト無しで）バッチ・モードにて機能するようにOpenSSLに命令します。
- -extfile は、証明書の拡張領域を含むファイルを指定します。

Java keystore に証明書をロード

Java keystore は秘密鍵そして対応する公開鍵を認証する関連したX.509証明書チェーンのリポジトリーです。Java keystore に証明書チェーンを入れる行為は、簡単なJavaプログラムを必要とします。リスト4のコード・フラグメントは作成した3連鎖の証明書チェーンを存在するkeystore に読み込みます。

リスト4. Java keystore に証明書チェーンを読み込む

```
 // The keystore to add the certificates to
```

```
FileInputStream ksis = new FileInputStream(".keystore");

// The new keystore with the certificates added
FileOutputStream ksos = new FileOutputStream(".newkeystore");

// The certificate files, to be added to keystore
FileInputStream certFile1 = new FileInputStream("cacert.cer");
FileInputStream certFile2 = new FileInputStream("userCert1.cer");
FileInputStream certFile3 = new FileInputStream("userCert2.cer");

CertificateFactory cf = CertificateFactory.getInstance("X.509");

// Read the 3 certificates into memory
Certificate cert1 = cf.generateCertificate(certFile1);
Certificate cert2 = cf.generateCertificate(certFile2);
Certificate cert3 = cf.generateCertificate(certFile3);

// Read the keystore file, type="jks"
KeyStore ks = KeyStore.getInstance("jks");
char[] password = "password".toCharArray();

ks.load(ksis, password);

// Add certificates to keystore
ks.setCertificateEntry("MYROOT_ALIAS", cert1);
ks.setCertificateEntry("MYUSER1_ALIAS", cert2);
ks.setCertificateEntry("MYUSER2_ALIAS", cert3);

// Write modified keystore to file system
ks.store(ksos, password);

ksos.close();
```

リスト4では、keystore のパスワードは`password`で、keystore のタイプは`jks`です。keystore に証明書が入れば、keystore にアクセスするように構成されたどのJavaアプリケーションもそれらを使えるようになります。SunのJDK文書はそれをどう実行するかに関する情報を含みます。

まとめ

既にご存知だとは思いますが、とても簡単な認証局を実装し証明書を発行するのにもOpenSSLを使えます。テスト・アプリケーションが立ち上がると同時に、OpenSSLを使用して証明書チェーンを取得することも可能です。しかしながら、このアプローチには、証明書の要求の発信元を検証し発行された証明書を管理すると言う責任が伴います。別の方法として、thawteやVeriSignのような外部認証局を使用して証明書を発行することも可能です。（ただ、これには特有の長所と短所があります。）

この記事では、PKIベースのセキュリティーを使用するJavaアプリケーションをテストする証明書チェーンをどのように作成するかを考察しました。論点を明確にするために、「生成された証明書を（E-mailのような）別のアプリケーションにて使用する。」、「生成された証明書を別の言語でどう使うか？」、そして「証明書取り消しリスト。」などのような、興味深い別の関連する分野にあえて触れませんでした。もしもこの分野そしてネットワーク・セキュリティーとデジタル証明書の複雑なトピックに関する様々な分野に興味をお持ちでしたら、参考文献の章にて示された役立つ情報源を活用してみてください。

著者について

Paul Abbott

Paul H. AbbottはIBM Java Technology Centreにて7年間以上勤務し、SunのJVMを様々なIBMプラットフォームに移植し続けました。過去2年間においては、これらのJVMでのIBMセキュリティー・コードの統合を任せられています。最近、J2EE/JMS開発部門へ異動しましたが、Javaセキュリティーへの飽くなき興味を抱き続け、developerWorksのJavaゾーンによるJavaセキュリティー・フォーラムの司会役を務めます。