Back to TOC

13.4 THE X.509 CERTIFICATE FORMAT STANDARD FOR PUBLIC KEY INFRASTRUCTURE (PKI)

- The set of standards related to the creation, distribution, use, **and revocation** of digital certificates is referred to as the **Public Key Infrastructure** (PKI). [In addition to PKI, another acronym that you will see frequently in the present context is PKCS, which, as previously mentioned in Section 12.6 of Lecture 12, stands for Public Key Cryptography Systems. If you search for information on the web, you will frequently see references to documents and protocols under the tag PKCS#N where N is usually a small integer. As stated in Lecture 12, these documents were produced by the RSA corporation that has been responsible for many of the PKI standards. Several of these documents eventually became IETF standards under the names that begin with RFC followed by a number. IETF stands for the Internet Engineering Task Force. A large number of standards that regulate the workings of the internet are IETF documents. Check them out at the `http://www.ietf.org` web page and find out about how the internet standardization process works.]

- **X.509** is one of the PKI standards. Besides other things, it is this standard that specifies the format of digital certificates. The X.509 standard is described in the IETF document RFC 5280 (also see its recent update in RFC 6818). [Just googling a string like "rfc5280" will take you directly to the source of such documents.]

- The X.509 standard is based on a strict hierarchical organization of the CAs in which the trust can only flow downwards. As mentioned previously at the beginning of Section 13.3, the CAs at the top of the hierarchy are known as **root CAs**. The CAs below the root are generally referred to as intermediate-level CAs.

- In order to verify the credentials of a particular CA as the issuer of a certificate, you approach the higher level CA for the needed verification. Obviously, this approach for establishing trust assumes that the root level CA must always be trusted implicitly.

- **IMPORTANT: The public keys of the root CAs, of which VeriSign, Comodo, and so on, are examples, are incorporated in your browser software and other applications that require networking so that the root-level verification is not subject to network-based man-in-the-middle attacks.** This also enables quick local authentication at the root level. In Linux machines, you'll find the root CA certificates in "`/etc/ssl/certs/`". [By the way, the status of the root CAs is verified annually by designated agencies. For example, Comodo's annual status as a root CA is verified annually by the global accounting firm KPMG. Again as a side note, Comodo owns 11 root keys. VeriSign is apparently the largest owner of root keys; it owns 13 root keys.]

- **For web-based applications, a certificate that cannot be**

authenticated by going up the chain of CAs all the way up to a root CA generates a warning popup from the browser.

- The format of an X.509 certificate is shown in Figure 2. The different fields of this certificate are described below:

  - **Version Number**: This describes the version of the X.509 standard to which the certificate corresponds. We are now on the third version of this standard. Since the entry in this field is zero based, so you'd see 2 in this field for the certificates that correspond to the latest version of the standard.

  - **Serial Number**: This is the serial number assigned to a certificate by the CA.

  - **Signature Algorithm ID**: This is the name of the digital signature algorithm used to sign the certificate. The signature itself is placed in the last field of the certificate.

  - **Issuer Name**: This is the name of the Certificate Authority that issued this certificate.

  - **Validity Period**: This field states the time period during which the certificate is valid. The period is defined with two

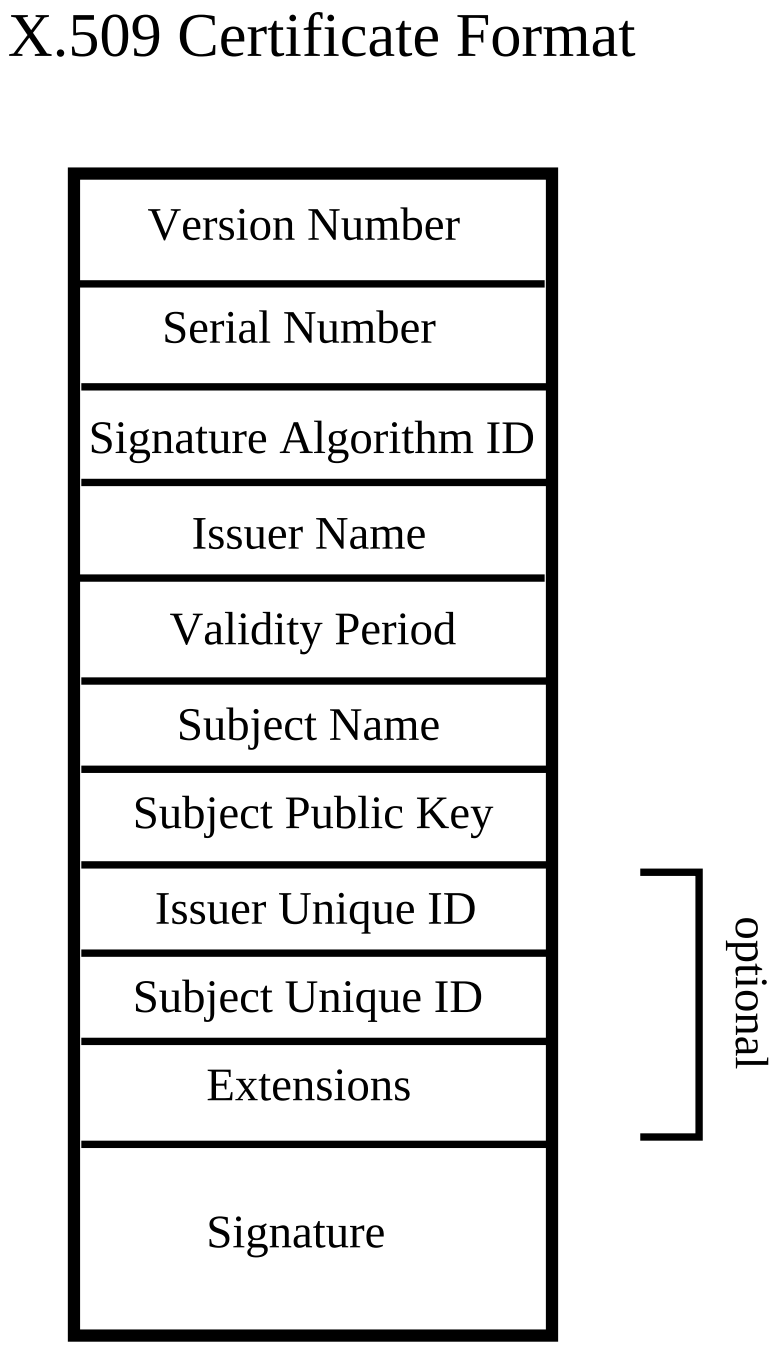

Figure 2: *The different fields of an X.509 certificate.* (*This figure is from Lecture 13 of "Computer and Network Security" by Avi Kak.*)

date-times, a **not before** date-time and a **not after** date-time.

– **Subject Name**: This field identifiers the individual/organization to which the certificate was issued. In other words, this field names the entity that wants to use this certificate to authenticate the public key that is in the next field.

– **Subject Public Key**: This field presents the public key that is meant to be authenticated by this certificate. This field also names the algorithm used for public-key generation.

– **Issuer Unique Identifier**: (optional) With the help of this identifier, two or more different CA's can operate as logically a single CA. The **Issuer Name** field will be distinct for each such CA but they will share the same value for the **Issuer Unique Identifier**.

– **Subject Unique Identifier**: (optional) With the help of this identifier, two or more different certificate holders can act as a single logical entity. Each holder will have a different value for the **Subject Name** field but they will share the same value for the **Subject Unique Identifier** field.

– **Extensions**: (optional) This field allows a CA to add additional private information to a certificate.

– **Signature:** This field contains the digital signature by the issuing CA for the certificate. **This signature is obtained by first computing a message digest of the rest of the fields with a hashing algorithm like SHA-1 (See Lecture 15) and then encrypting it with the CA's private key.** Authenticity of the contents of the certificate can be verified by using CA's public key to retrieve the message digest and then by comparing this digest with one computed from the rest of the fields.

- The digital representation of an X.509 certificate, described in RFC 5280, is created by first using the following ASN.1 representation to generate a byte stream for the certificate and converting the bytestream into a printable form with Base64 encoding. [As mentioned in Section 12.8 of Lecture 12, ASN stands for Abstract Syntax Notation and the ASN.1 standard, along with its transfer encoding DER (for Distinguished Encoding Rules), accomplishes the same thing in binary format for complex data structures that the XML standard does in textual format.] Shown below is the ASN.1 representation of an X.509 certificate:

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
     version         [0]  EXPLICIT Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
     issuer               Name,
     validity             Validity,
```

```
     subject              Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     extensions      [3]  EXPLICIT Extensions OPTIONAL
                          -- If present, version MUST be v3
     }

Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }

CertificateSerialNumber  ::=  INTEGER

Validity ::= SEQUENCE {
     notBefore      Time,
     notAfter       Time }

Time ::= CHOICE {
     utcTime        UTCTime,
     generalTime    GeneralizedTime }

UniqueIdentifier  ::=  BIT STRING

SubjectPublicKeyInfo  ::=  SEQUENCE  {
     algorithm            AlgorithmIdentifier,
     subjectPublicKey     BIT STRING  }

Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

Extension  ::=  SEQUENCE  {
     extnID      OBJECT IDENTIFIER,
     critical    BOOLEAN DEFAULT FALSE,
     extnValue   OCTET STRING
                 -- contains the DER encoding of an ASN.1 value
                 -- corresponding to the extension type identified
                 -- by extnID
```

- It is the hash of the bytestream that corresponds to what is stored for the field `TBSCertificate` that is encrypted by the CA's private key for the digital signature that then becomes the value of the `signatureValue` field. You may read `TBSCertificate` as the "To Be Signed" potion of what appears in the final certificate. As to what algorithms are used for hashing and for encryption with the CA's private key, that is identified by the value of the field **`signatureAlgorithm`**.

- Using the Base64 representation (see Lecture 2), an X.509 certificate is commonly stored in a printable form according to the RFC 1421 standard. In its printable form, a certificate will normally be bounded by the first string shown below at the beginning and the second at the end.

```
-----BEGIN CERTIFICATE-----


-----END CERTIFICATE-----
```

  Shown below is an example of a certificate in Base64 representation and it resides in a file whose name carries the ".pem" suffix. The programming problem in Section 13.9 has more to say about the PEM format for representing keys and certificates.


```
-----BEGIN CERTIFICATE-----
MIIDJzCCApCgAwIBAgIBATANBgkqhkiG9w0BAQQFADCBzjELMAkGA1UEBhMCWkEx
FTATBgNVBAgTDFdlc3Rlcm4gQ2FwZTESMBAGA1UEBxMJQ2FwZSBUb3duMR0wGwYD
VQQKExRUaGF3dGUgQ29uc3VsdGluZyBjYzEoMCYGA1UECxMfQ2VydGlmaWNhdGlv
biBTZXJ2aWNlcyBEaXZpc2lvbjEhMB8GA1UEAxMYVGhhd3RlIFByZW1pdW0gU2Vy
dmVyIENBMSgwJgYJKoZIhvcNAQkBFhlwcmVtaXVtLXNlcnZlckB0aGF3dGUuY29t
MB4XDTk2MDgwMTAwMDAwMFoXDTIwMTIzMTIzNTk1OVowgc4xCzAJBgNVBAYTAlpB
MRUwEwYDVQQIEwxXZXN0ZXJuIENhcGUxEjAQBgNVBAcTCUNhcGUgVG93bjEdMBsG
A1UEChMUVGhhd3RlIENvbnN1bHRpbmcgY2MxKDAmBgNVBAsTH0NlcnRpZmljYXRp
b24gU2VydmljZXMgRGl2aXNpb24xITAfBgNVBAMTGFRoYXd0ZSBQcmVtaXVtIFNl
cnZlciBDQTEoMCYGCSqGSIb3DQEJARYZcHJlbWl1bS1zZXJ2ZXJAdGhhd3RlLmNv
bTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0jY2aovXwlue2oFBYo847kkE
VdbQ7xwblRZH7xhINTpS9CtqBo87L+pW46+GjZ4X9560ZXUCTe/LCaIhUdib0GfQ
ug2SBhRz1JPLlyoAnFxODLz6FVL88kRu2hFKbgifLy3j+ao6hnO2RlNYyIkFvYMR
uHM/qgeN9EJN50CdHDcCAwEAAaMTMBEwDwYDVR0TAQH/BAUwAwEB/zANBgkqhkiG
9w0BAQQFAAOBgQAmSCwWwlj66BZ0DKqqX1Q/8tfJeGBeXm43YyJ3Nn6yF8Q0ufUI
hfzJATj/Tb7yFkJD57taRvvBxhEf8UqwKEbJw8RCfbz6q1lu1bdRiBHjpIUZa4JM
pAwSremkrj/xw0llmozFyD4lt5SZu5IycQfwhl7tUCemDaYj+bvLpgcUQg==
-----END CERTIFICATE-----
```

- Ordinarily you would request a CA for a certificate for your public key. But that does not prevent you from generating your own certificates for testing purposes. If you have Ubuntu installed on your machine, try out the following command:

```
openssl req -new -newkey rsa:1024 -days 365 -nodes -x509 -keyout test.pem -out test.cert
```

  where the first argument `req` to `openssl` is for generating an X509 certificate, the rest of the arguments being self-explanatory. This command will deposit a new private key for you in the file `test.pem` and the certificate in the file `test.cert`. [By the way, OpenSSL, the open-source library that supports the command `openssl` used above, is an amazingly useful library in C that implements the SSL/TLS protocol (that we will take up in greater depth in Lecture 20). It contains production-quality code for virtually anything you would ever want to do with cryptography — symmetric-key cryptography, public-key cryptography, hashing, certificate generation, etc. Check it out at `www.openssl.org`. If you are running Ubuntu and you have OpenSSL installed, do `man openssl` to see all the things that you can do with the command shown above as you give it different arguments.] When you invoke the above command, it will ask you for information related to you and your organization. It is not necessary to supply the information that you are prompted for, though.

- You can also use OpenSSL to make your own organization a CA. Visit `http://sandbox.rulemaker.net/ngps/m2/howto.ca.html` to find out how you can do it.

- Shown on the next page is the X.509 certificate that belongs to the InCommon root CA (https://www.incommon.org/). InCommon is used by several universities and research organizations in the US for data encryption for web servers. The certificate shown below can be downloaded from

https://www.incommon.org/cert/repository/InCommonServerCA.txt.

- To see the role played by the InCommon's certificate shown on the next page, let's say the web browser in your computer requests a page from the engineering.purdue.edu web server that I use for hosting my computer and network security lecture notes. This server supplies all its content using the TLS/SSL protocol, meaning that all interactions with this server are encrypted. In order to create an encrypted session with the server, your browser first downloads engineering.purdue.edu's certificate — which is signed by InCommon — and then authenticates it through InCommons's public key that is supplied by their own certificate shown on the next page. **IMPORTANT**: Note that InCommon is an intermediate level CA whose own certificate is signed by a root CA called AddTrust. Being a root CA, AddTrust's public key (in the form of a self-signed certificate) comes preloaded in your computer and resides in the directory "/etc/ssl/certs/". As mentioned earlier in this lecture, you can view any of these certificates by executing the command "openssl x509 -text < cert_file_name". Being preloaded in your computer, the acquisition of AddTrust's public key is NOT vulnerable to man-in-the-middle attack. The web browser running in your computer and the engineering.purdue.edu's web server use the

SSL/TLS protocol to create a session that cannot be eavesdropped on. For that, your browser first downloads the `engineering.purdue.edu`'s certificate as already mentioned. From the URL provided in this certificate to the InCommon web site, your browser next downloads the InCommon's certificate that is shown below. Next, it verifies InCommon's certificate using the pre-stored AddTrust certificate in the directory `/etc/ssl/certs/`. Subsequently, it uses the public key in the authenticated InCommon's certificate to authenticate the public key in `engineering.purdue.edu`'s certificate. Shown below is InCommon's certificate:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7f:71:c1:d3:a2:26:b0:d2:b1:13:f3:e6:81:67:64:3e
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=SE, O=AddTrust AB, OU=AddTrust External TTP Network, CN=AddTrust External CA Root
        Validity
            Not Before: Dec  7 00:00:00 2010 GMT
            Not After : May 30 10:48:38 2020 GMT
        Subject: C=US, O=Internet2, OU=InCommon, CN=InCommon Server CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:97:7c:c7:c8:fe:b3:e9:20:6a:a3:a4:4f:8e:8e:
                    34:56:06:b3:7a:6c:aa:10:9b:48:61:2b:36:90:69:
                    e3:34:0a:47:a7:bb:7b:de:aa:6a:fb:eb:82:95:8f:
                    ca:1d:7f:af:75:a6:a8:4c:da:20:67:61:1a:0d:86:
                    c1:ca:c1:87:af:ac:4e:e4:de:62:1b:2f:9d:b1:98:
                    af:c6:01:fb:17:70:db:ac:14:59:ec:6f:3f:33:7f:
                    a6:98:0b:e4:e2:38:af:f5:7f:85:6d:0e:74:04:9d:
                    f6:27:86:c7:9b:8f:e7:71:2a:08:f4:03:02:40:63:
                    24:7d:40:57:8f:54:e0:54:7e:b6:13:48:61:f1:de:
                    ce:0e:bd:b6:fa:4d:98:b2:d9:0d:8d:79:a6:e0:aa:
                    cd:0c:91:9a:a5:df:ab:73:bb:ca:14:78:5c:47:29:
                    a1:ca:c5:ba:9f:c7:da:60:f7:ff:e7:7f:f2:d9:da:
                    a1:2d:0f:49:16:a7:d3:00:92:cf:8a:47:d9:4d:f8:
                    d5:95:66:d3:74:f9:80:63:00:4f:4c:84:16:1f:b3:
                    f5:24:1f:a1:4e:de:e8:95:d6:b2:0b:09:8b:2c:6b:
                    c7:5c:2f:8c:63:c9:99:cb:52:b1:62:7b:73:01:62:
                    7f:63:6c:d8:68:a0:ee:6a:a8:8d:1f:29:f3:d0:18:
                    ac:ad
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:AD:BD:98:7A:34:B4:26:F7:FA:C4:26:54:EF:03:BD:E0:24:CB:54:1A

            X509v3 Subject Key Identifier:
```

```
                48:4F:5A:FA:2F:4A:9A:5E:E0:50:F3:6B:7B:55:A5:DE:F5:BE:34:5D
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Certificate Policies:
                Policy: X509v3 Any Policy

            X509v3 CRL Distribution Points:
                URI:http://crl.usertrust.com/AddTrustExternalCARoot.crl

            Authority Information Access:
                CA Issuers - URI:http://crt.usertrust.com/AddTrustExternalCARoot.p7c
                CA Issuers - URI:http://crt.usertrust.com/AddTrustUTNSGCCA.crt
                OCSP - URI:http://ocsp.usertrust.com

    Signature Algorithm: sha1WithRSAEncryption
        93:66:21:80:74:45:85:4b:c2:ab:ce:32:b0:29:fe:dd:df:d6:
        24:5b:bf:03:6a:6f:50:3e:0e:1b:b3:0d:88:a3:5b:ee:c4:a4:
        12:3b:56:ef:06:7f:cf:7f:21:95:56:3b:41:31:fe:e1:aa:93:
        d2:95:f3:95:0d:3c:47:ab:ca:5c:26:ad:3e:f1:f9:8c:34:6e:
        11:be:f4:67:e3:02:49:f9:a6:7c:7b:64:25:dd:17:46:f2:50:
        e3:e3:0a:21:3a:49:24:cd:c6:84:65:68:67:68:b0:45:2d:47:
        99:cd:9c:ab:86:29:11:72:dc:d6:9c:36:43:74:f3:d4:97:9e:
        56:a0:fe:5f:40:58:d2:d5:d7:7e:7c:c5:8e:1a:b2:04:5c:92:
        66:0e:85:ad:2e:06:ce:c8:a3:d8:eb:14:27:91:de:cf:17:30:
        81:53:b6:66:12:ad:37:e4:f5:ef:96:5c:20:0e:36:e9:ac:62:
        7d:19:81:8a:f5:90:61:a6:49:ab:ce:3c:df:e6:ca:64:ee:82:
        65:39:45:95:16:ba:41:06:00:98:ba:0c:56:61:e4:c6:c6:86:
        01:cf:66:a9:22:29:02:d6:3d:cf:c4:2a:8d:99:de:fb:09:14:
        9e:0e:d1:d5:c6:d7:81:dd:ad:24:ab:ac:07:05:e2:1d:68:c3:
        70:66:5f:d3
-----BEGIN CERTIFICATE-----
MIIEwzCCA6ugAwIBAgIQf3HB06ImsNKxE/PmgWdkPjANBgkqhkiG9w0BAQUFADBv
MQswCQYDVQQGEwJTRTEUMBIGA1UEChMLQWRkVHJ1c3QgQUIxJjAkBgNVBAsTHUFk
ZFRydXN0IEV4dGVybmFsIFRUUCBOZXR3b3JrMSIwIAYDVQQDExlBZGRUcnVzdCBF
eHRlcm5hbCBDQSBSb290MB4XDTEwMTIwNzAwMDAwMFoXDTIwMDUzMDEwNDgzOFow
UTELMAkGA1UEBhMCVVMxEjAQBgNVBAoTCUludGVybmV0MjERMA8GA1UECxMISW5D
b21tb24xGzAZBgNVBAMTEkluQ29tbW9uIFNlcnZlciBDQTCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAJd8x8j+s+kgaq0kT46ONFYGs3psqhCbSGErNpBp
4zQKR6e7e96qavvrgpWPyh1/r3WmqEzaIGdhGg2GwcrBh6+sTuTeYhsvnbGYr8YB
+xdw26wUWexvPzN/ppgL5OI4r/V/hWOOdASd9ieGx5uP53EqCPQDAkBjJH1AV49U
4FR+thNIYfHezg69tvpNmLLZDY15puCqzQyRmqXfq3O7yhR4XEcpocrFup/H2mD3
/+d/8tnaoS0PSRan0wCSz4pH2U341ZVm03T5gGMAT0yEFh+z9SQfoU7e6JXWsgsJ
iyxrx1wvjGPJmctSsWJ7cwFif2Ns2Gig7mqojR8p89AYrK0CAwEAAaOCAXcwggFz
MB8GA1UdIwQYMBaAFK29mHo0tCb3+sQmVO8DveAky1QaMB0GA1UdDgQWBBRIT1r6
L0qaXuBQ82t7VaXe9b40XTAOBgNVHQ8BAf8EBAMCAQYwEgYDVR0TAQH/BAgwBgEB
/wIBADARBgNVHSAECjAIMAYGBFUdIAAwRAYDVR0fBD0wOzA5oDegNYYzaHR0cDov
L2NybC51c2VydHJ1c3QuY29tL0FkZFRydXN0RXh0ZXJuYWxDQVJvb3QuY3JsMIGz
BggrBgEFBQcBAQSBpjCBozA/BggrBgEFBQcwAoYzaHR0cDovL2NydC51c2VydHJ1
c3QuY29tL0FkZFRydXN0RXh0ZXJuYWxDQVJvb3QucDdjMDkGCCsGAQUFBzAChi1o
dHRwOi8vY3J0LnVzZXJ0cnVzdC5jb20vQWRkVHJ1c3RVVE5TR0NDQS5jcnQwJQYI
KwYBBQUHMAGGGWh0dHA6Ly9vY3NwLnVzZXJ0cnVzdC5jb20wDQYJKoZIhvcNAQEF
BQADggEBAJNmIYB0RYVLwqvOMrAp/t3f1iRbvwNqb1A+DhuzDYijW+7EpBI7Vu8G
f89/IZVWO0Ex/uGqk9KV85UNPEerylwmrT7x+Yw0bhG+9GfjAkn5pnx7ZCXdF0by
UOPjCiE6SSTNxoRlaGdosEUtR5nNnKuGKRFy3NacNkN089SXnlag/l9AWNLV1358
xY4asgRckmYOha0uBs7Io9jrFCeR3s8XMIFTtmYSrTfk9e+WXCAONumsYn0ZgYr1
kGGmSavOPN/mymTugmU5RZUWukEGAJi6DFZh5MbGhgHPZqkiKQLWPc/EKo2Z3vsJ
FJ4O0dXG14HdrSSrrAcF4h1ow3BmX9M=
-----END CERTIFICATE-----
```

- Since all valid certificates are cached by your browser, if you previously visited the `engineering.purdue.edu` domain, the InCommon certificate I showed above is probably already in your computer. You can check whether or not that's the case through your browser's certificate viewer tool. For FireFox, you can get to the certificate viewer by clicking on the "edit" button in the menu bar of the browser and by further clicking as shown below:

```
Preferences -->
    Advanced -->
        Certificates -->
            "View Certificates" button -->
                "Authorities" to view the CA certificates -->
                    Scroll down to "AddTrust AB" -->
                        Further scroll down to "InCommon Server CA"
```

  where the last item will show up only if you previously visited the `engineering.purdue.edu` domain. Assuming it is there, when you double-click on the last item, you will see a popup with two buttons. The left button leads you to general information regarding the root CA and the right button shows the details regarding the root certificate through a tree structure. When you click on "Subject's public key", you will see the modulus and the public exponent used by this root. In the general information provided by the left button, you will notice that the serial number of the root certificate matches that of the root certificate that I downloaded directly from InCommon's web site and that is reproduced above.

- If you want to view the root CA certificates that have been deposited in your browser by different internet service provides (after they were verified by your browser), in the fifth action item in the indented list of actions shown above, click on "Servers".