这是TLS / SSL和X.509（SSL）证书（包括自签名证书）的普见主题的生存指南。这些是松散称为公钥基础设施（PKI）的元素。

俗称的SSL证书应被称为X.509证书。由于Netscape在设计SSL（安全套接字层）协议的原始版本时采用X.509（国际电联X.500目录标准中的一个）证书格式，因此术语SSL证书很常见，世界还很年轻，dinosuars仍然漫游，互联网是一个友好的地方。术语"SSL证书"已经持续，并且可能在可预见的未来持续，因为给定选择"SSL证书"或"X.509证书"，前者倾向于更舒适地滚动舌头。毫无疑问，一个语言专家可以嘲笑S声音与X声音。对于我们，只是凡人，它的主要优点可能是它更短。

当前指南包括SSL，TLS，有关X.509的一些详细信息及其用法以及关于证书类型（包括EV证书）和信任过程的一些说明。创建自签名证书是作为使用OpenSSL包的工作示例。

您可以购买SSL（X.509）证书或生成自己的（自签名证书）进行测试，或者根据应用程序，甚至在生产环境中。**好消息**：如果你自己签署你的证书，你可以节省一大笔钱。**坏消息**：如果你自己签署你的证书，没有人，但你和你的亲人（也许）可能信任他们。但在你为了一个明亮，闪亮的新X.509（SSL）证书或更昂贵的EV SSL（X.509）证书，你可能想知道他们做什么和他们做什么之前，你所有的肮脏的卢布。如果你的眼睛眩晕，当人们开始谈论SSL，安全和证书 - 现在开始上釉。这东西不好玩。

<ingrained habits>以下页面中的RFC超链接链接到在发布RFC时复制到我们网站的纯文本版本。我们开始这样做很长很久以前，当RFC在一些奇怪的地方，偶尔移动位置，性能和可靠性的存储库是非常变化（慷慨）。这些条件都不适用于今天，远不如此。IETF像IANA一样拥有坚实的网站，具有卓越的性能和不断改进的功能。然而，我们坚持我们的根深蒂固的习惯没有特别好的理由（老狗...新技巧..）。

注意：如果你想/需要/渴望更多的选项超过RFC格式，那么你现在有一个真正的玉米选择。RFC的主存储库由IETF维护，文本版本（规范性参考）可以在**www.ietf.org/rfc/rfcXXXX.txt**或**www.rfc-editor.org/rfc/rfcXXXX.txt**（其中XXXX是4位RFC号码 - 根据需要左侧用零填充）。当前发布的RFC指向**https://www.rfc-editor.org/info/rfcXXXX**，其中包含各种信息和指向文本（规范性）引用和PDF（非规范性）版本的链接。RFC也可以在**http：//datatracker.ietf查看**。org / doc / rfcXXXX / ，它还包含各种RFC状态信息（包括勘误）以及替代格式列表，如文本，PDF和HTML（这是文档的工作区域版本）。最后，现在有一个可搜索的RFC列表。

我们不时地更新页面，当我们可以想到没有什么比我们的生活更好，现在保存更改日志。

内容：

TLS / SSL协议概述
TLS / SSL协议图
TLS / SSL - 详细说明
加密 - 概述
X.509证书概述

X.509证书类型和术语
X.509证书链
X.509用法（为什么我买这个东西）
X.509证书在网络托管组织
证书协议（CMP，CMC / CMS，SCVP，OCSP，HTTP）
在线证书状态协议（OCSP）
X.509证书格式
证书主题和subjectaltName注释
X.509证书吊销列表（CRL）
过程和信任 - CA和X.509证书
扩展验证（EV）X.509证书
示例 - 创建自签名证书（OpenSSL - 多种方法）
示例 - 创建从属CA，中间证书和交叉证书（OpenSSL）
示例 - 创建多主机证书（OpenSSL）
SSL相关文件格式注释
PEM格式
PEM BEGIN关键字（标签）
文件名和内容
证书包
OpenSSL转换，提取和操作
PKCS #X到RFC表
在常见浏览器中处理证书
相关RFC

TLS / SSL协议

SSL（X.509）证书的主要用途是与TLS / SSL协议结合使用。安全套接字层（SSL）是最初在1992年创建的Netscape协议，用于在Web服务器和底层网络不安全的浏览器之间安全地交换信息。它经历了各种迭代，现在是版本3（约1995年），并在各种客户端 - 服务器应用程序中使用。由于Netscape的停止，SSL规范将不会进一步更新。因此，它是一个死标准（死在死鹦鹉），确实RFC 7568最终废弃了SSL v3。它现在正式是一只死鹦鹉，以后不能使用伟大和良好的秩序（在这种情况下，显着敏感）。IETF标准化传输层安全（TLS）版本1，RFC 2246中的较小变体， RFC 4346中的 1.1 版本和RFC 5246中的 V1.2版。此外，在RFC 3546中定义了许多扩展，当在带宽受限系统（例如无线网络）中使用TLS时，RFC6066定义了以扩展客户端问候格式（用TLS 1.2引入）携带的多个TLS扩展，RFC6961定义了一种方法用于当客户端请求服务器提供证书状态信息时减少流量。和RFC 7935 在RFC 3546中定义了多个扩展，当在带宽受限的系统（例如无线网络）中使用TLS时，RFC6066定义以扩展的客户端问候格式（用TLS 1.2引入）携带的多个TLS扩展，RFC6961定义了用于减少业务当客户端请求服务器提供证书状态信息时。和RFC 7935 在RFC 3546中定义了多个扩展，当在带宽受限的系统（例如无线网络）中使用TLS时，RFC6066定义以扩展的客户端问候格式（用TLS 1.2引入）携带的多个TLS扩展，RFC6961定义了用于减少业务当客户端请求服务器提供证书状态信息时。和RFC 7935

现在定义了在物联网（物联网或物联网）中使用TLS（和DTLS）时发生了什么，我们以我们的偶像方式，更喜欢）。

当最初建立安全连接时，根据实现，它将从集合SSLv3，TLSv1，TLSv1.1或TLSv1.2协商特定协议的支持。这就是名称SSL的普遍性，在大多数情况下，所谓的SSL最有可能使用TLS - 例如OpenSSL支持SSL（v3）和TLS（TLSv1，TLSv1.1和TLSv1.2）协议。虽然SSL和TLS之间存在细节差异，但以下描述适用于这两种协

议。**注意：** SSLv2被RFC 6176禁止，其中包含其缺点的可怕列表。SSLv3现在已经加入了他的哥哥被RFC 7568驱逐。

TLS / SSL在TCP顶部运行，但低于它确保的最终用户协议，例如HTTP或IMAP，如图1所示。

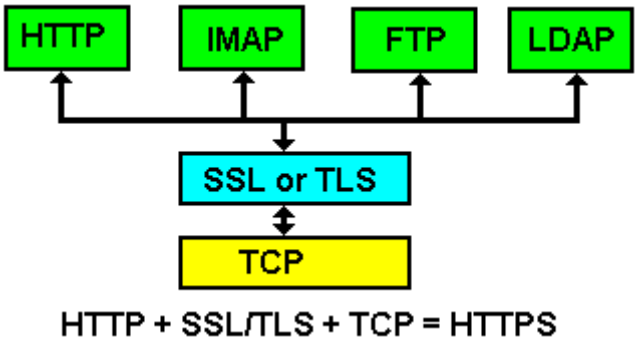

图1 - TLS / SSL分层。

TLS / SSL没有已知的端口号 - 而是与较高层协议（例如HTTP）一起使用时，该协议指定安全变体，在HTTP的情况下为HTTPS，其具有公知的（或默认）端口号。HTTPS的标识简单地表示正在运行在通过TCP运行的TLS / SSL连接之上的正常HTTP。在HTTPS的情况下，众所周知的端口号是443，在IMAPS的情况下，端口993，POP3S - 端口995等。

下一级描述需要一些熟悉术语MAC（消息认证码），安全散列，对称和非对称加密算法。对于那些不习惯这些术语的人，他们在这个加密生存指南中。你可能想在阅读这些东西后躺在一个黑暗的房间里躺一会儿。

笔记：

1. 相关协议（数据报传输层安全（DTLS））在与UDP（由RFC7507更新的RFC 6347）一起使用时定义安全服务。虽然priciples类似于TLS本指南不讨论DTLS进一步。

2. 术语TLS 1.2 Suite B（由RFC 6460定义）定义了与NSA Suite B加密技术兼容的加密套件（见下文），仅当TLS用于美国国家安全应用程序时才适用。

概述 - 建立安全连接

当使用TLS / SSL（例如使用HTTPS（默认端口443））建立安全连接时，在始终启动连接的客户端与服务器之间发生消息交换。第一组消息称为握手协议，之后客户端和服务器都进入记录（或数据）协议。在握手协议期间的消息交换实现以下目的：

1. 从SSLv3，TLSv1，TLSv1.1，TLSv1.2的支持集（取决于实现）建立要使用的协议变体 - 将始终使用最新的可能变体，因此TLSv1将总是优先于SSLv3使用客户端和服务器都支持。客户端提供一个列表 - 服务器从提供的列表中进行选择。

2. 发送认证数据。服务器最常见地以X.509（SSL）证书的形式（包装）发送认证信息，但协议支持其他方法。

3. 建立会话ID，以便如果需要可以重新启动会话。

4. 协商由密钥交换算法以及后续数据会话（记录协议）中使用的批量数据加密算法类型和MAC类型组成的密码套件。密钥交换算法通常使用诸如RSA，DSA或ECC（椭圆曲线密码 - 参见RFC5289）的不对称（公钥 - 私钥）算法。不对称算法在资源（CPU）中非常昂贵，因此对称密码用于随后的批量数据加密（使用记录协议）。密钥交换算法用于传送信息，从该信息可以为对称（批量数据）密码独立地计算会话密钥。MAC在记录协议期间保护所发送/接收的数据的完整性。

这可以交换简化的概述和附加数据，例如，可以请求客户端在称为相互认证的过程中发送认证X.509（SSL）证书，但是上述描述了*最常见的*情况，并且在图1中示出2下面：

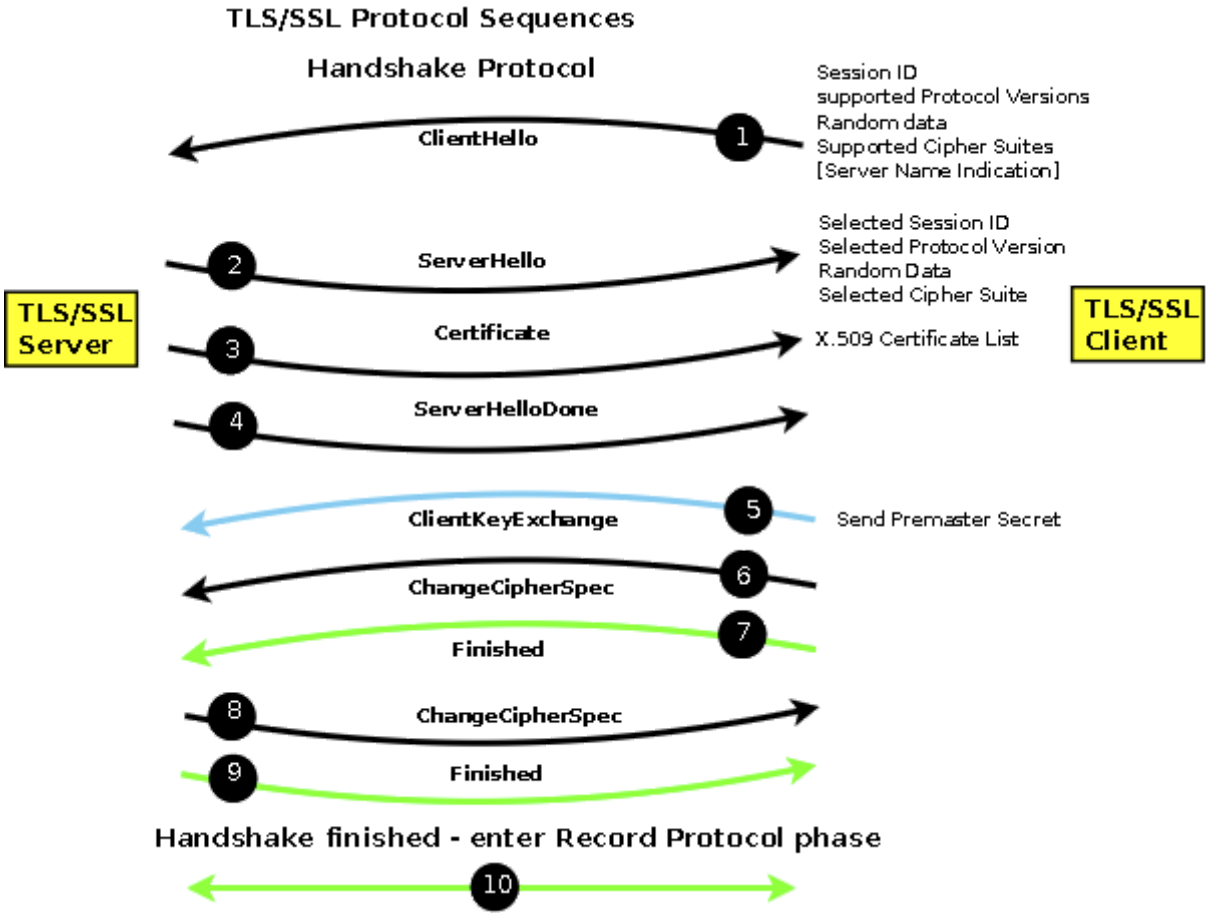

图2 - TLS / SSL协议序列

笔记：

1. 握手协议协商并建立连接，并且记录协议传输（封装）加密的数据流，例如HTTP，SMTP或IMAP。

2. 在图2中，黑色消息以明文（未加密）发送；蓝色消息使用服务器提供的公钥（使用密钥交换密码）发送，并要求服务器访问相应的私钥；绿色消息使用协商的批量数据密码发送，并由协商的MAC保护。

3. TLS / SSL允许将数据压缩算法作为密码套件的一部分进行协商。考虑到现代网络的速度，数据压缩很少（如果曾经使用过的话），并且通常在协商的密码组中被设置为值NULL（未使用）。

TLS / SSL - 详细说明

本节提供有关TLS / SSL协议消息交换（图2上面）的更多细节，为那些谁知道的精彩细节。如果你喜欢"东西发生"级别跳过本节，保留你的理智。

1. **ClientHello（1）：** ClientHello提供了支持的协议版本/变体的列表，支持的密码套件按照优选顺序以及压缩算法列表（通常为NULL）。客户端还发送32个八位字节随机值（随机数）（稍后用于计算对称密钥），以及会话ID（如果没有先前会话存在则为0，如果客户端认为存在先前会话，则为非零）。

    - 每个密码组包括密钥交换算法，批量密码算法和MAC（哈希）算法。
    - 客户端和服务器在初始连接时使用的密码套件是：

```
TLS_NULL_WITH_NULL_NULL (0x00, 0x00)
#第一个NULL是密钥交换算法
#下面的WITH_NULL定义了批量密码
#final final定义MAC
```

      此值表示不会发生加密，因此所有消息将以明文形式发送，直到客户端密钥交换（ClientKeyMessage）。

    - 典型的加密套件是：

```
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x00, 0x0A)
#RSA是密钥交换算法
#WITH_3DES_EDE_CBC定义批量密码
# （具有循环块链的三重DES）
#SHA是MAC（Hash）
```

      笔记：

        1. 要添加到您的术语表中，每个密码套件都被编码，这个编码值（两个八位字节）被称为信令密码套件值（SCSV）。现在信息可能已经使你的一天。
        2. 有效的加密套件值可以在相关TLS RFC的附录C中找到（TLS 1的RFC 2246，TLS 1.1的RFC 4346和TLS 1.2的RFC 5246）。这些是由RFC 4492和RFC 7027为ECC（椭圆曲线密码）更新的。
        3. 出口一词出现在一些有效的加密套件描述中，是指出口强度密码，也就是说，某些密码只允许在某些国家（见美国商务部，工业和安全局（BIS）和Wassenaar安排）和在配置可能在国际上使用的系统时应牢记。
        4. 在RFC 3546中定义并主要用于无线网络中的扩展可以以向后兼容的方式在ClientHello被调用。RFC6066显着增加了TLS扩展的数量，包括可以在常规（非无线）网络上使用的许多TLS扩展。服务器可以随意忽略任何不了解的扩展。
        5. RFC6066引入了TLS证书状态扩展（status_request）类型，本质上说，我（客户端）绝对不信任您的（服务器）证书，但我绝对相信您的（服务器）响应我的（证书状态）有效性（！）"。对证书状态请求（通常通过使用OCSP获得）的响应在**证书**消息之后立即在**证书状态**消息中发送（见下文）。显然，证书状态（状态请求）已经被广

泛地流行（对于无线设备）具有破坏OCSP服务器的危险。RFC6961引入了"certificate_request_v2"

- RFC 6066定义了一个可选的扩展（服务器名称指示 - SNI），允许客户端在进行初始TLS / SSL连接时发送服务器名称，例如www.example.com。此功能（由大多数现代浏览器支持）允许支持多个网站的Web服务器（例如Apache的VirtualHost功能）在其**证书（3）**消息中发送站点特定证书。（配置Apache 2以支持SNI。）

- RFC 7250定义了扩展**client_certificate_format**其可以被用于指示证书的格式被使用，并且可以是在正常的X.509格式或**RawPublicKey**格式，其中，证书被减少到仅SubjectPublicKeyInfo进行属性在随后的证书传送消息（ s）的握手协议。

- 许多TLS / SSL客户端将尝试回退到较低的协议版本，如果网络错误，可能不必要地削弱了连接。RFC 7507定义了一个新的密码套件：

  ```
  TLS_FALLBACK_SCSV {0x56, 0x00}
  ```

  TLS / SSL客户端可以在尝试协商降低的协议版本的任何尝试中包括此消息。较旧的服务器将忽略此类消息，并且将使用减少的协议版本正常进行连接。如果所提供的协议版本低于服务器支持的协议版本，则识别该消息的较新的服务器将终止具有故障警报**unsat_fallback（86）**的客户端连接，因此限制了协商不必要的版本减少的情况。

- RFC 7685定义了一个扩展，可以用来填充（使用零）ClientHello的大小，以改善错误的TLS实现的效果（我们没有做这个东西）。

- RFC 7633定义了一个新的X.509证书扩展，其中包括证书支持的TLS扩展列表。如果服务器不提供引用的TLS扩展，客户端可能承担潜在的安全违规并放弃会话。显然，客户端决定必须推迟到TLS握手的证书阶段之后。

2. **ServerHello（2）：** ServerHello返回所选的协议变体/版本号，加密套件和压缩算法。服务器发送32个八位字节的随机值（随机数），稍后用于计算对称密钥。如果ClientHello中的会话ID为0，服务器将创建并返回会话ID。如果ClientHello提供了服务器已知的先前的会话ID，则协商减少的握手。如果客户端提供了服务器无法识别的会话ID，则服务器返回新的会话ID和完全握手结果。

   RFC 7250定义了扩展**server_certificate_format**其可以被用于指示证书的格式被使用，并且可以是在正常的X.509格式或**RawPublicKey**格式，其中，证书被减少到仅SubjectPublicKeyInfo进行属性在随后的证书传送消息（ s）的握手协议。

3. **证书（3）：** 服务器发送其X.509证书，其包含服务器的公钥，其算法必须与所选加密套件套件的密钥交换算法相同。协议提供的其他方法来传输公钥 - 例如可以简单地指向DNS KEY / TLSA RR - 但是X.509证书是正常的方法并且被普遍支持。这个消息的目的是客户端将从可信源获得服务器的公钥，它可以用来发送加密消息。

   笔记：

1. 虽然在此消息中仅发送单个证书是正常的，但可以发送所谓的证书包（单个PEM文件中的多个证书）。例如，可以使用Apache指令**SSLCertificateChainFile**定义证书捆绑，而使用SSLCertificateFile**指令**定义**单个**证书。当在某种形式的证书重构期间存在交叉证书时，通常使用捆绑包，例如，公司接管另一个CA公司或从CA更改密钥/ keysize。

2. DNSSEC DANE协议（RFC6698）允许客户端从DNS获取服务器X.509证书的副本。然而，使用DANE从DNS获得的证书是除了在TLS / SSL的正常证书交换中获得的证书之外，并且允许对常年偏执的额外验证，而不增加安全性。

3. RFC 7250定义了一种残留证书格式，它将原始公钥封装在由SubjectPublicKeyInfo（必须描述公钥属性）组成的包装器中。这是朝着直接从经认证的源（例如，DNSSEC安全的DNS RR）获得公钥的saner解决方案的适度移动。

4. **ServerDone（4）**：表示此对话序列的服务器部分的结束，并邀请客户端继续协议序列。**注意：**服务器可能在此时请求客户端证书以完成相互验证。这个客户证书交换序列已经从协议序列描述中省略，因为它不常用，并且不必要地使描述复杂化。

   注意：如果在初始TLS / SSL协商期间，服务器请求客户端证书，则客户端必须发送此格式（以与服务器定义的格式相同的格式，但RFC 6066允许任何客户端发送证书URL，而不是一个完整的证书）紧随**ServerDone**到和以前**ClientKeyExchange**。

5. **ClientKeyExchange（5）**：客户端使用服务器和客户端随机数（或随机数）计算所谓的主主密钥。它使用从提供的X.509证书获取的服务器的公钥对此密钥进行加密。只有服务器可以使用其私钥解密此消息。双方使用在标准中定义的算法从预主密钥中独立地计算主密钥。任何所需的会话密钥都来自此主密钥。

   注意：

   1. 已经表明，TLS（和DTLS）可能容易受到中间人（MTM）攻击。为了消除这种可能性，RFC 7627重新定义了通过用从**ClientHello**到**ClientKeyExchange**的完整会话的哈希替换服务器和客户机随机数来计算主秘密（最初在RFC 5246中定义）的**方法**。客户端指示它可以通过在其**ClientHello**中发送一个空的**extended_master_secret**扩展来使用新的（重新定义的）主秘密算法。如果服务器可以支持新算法，它还将在其**ServerHello中**镜像空的**extended_master_secret 扩展**。

6. **ChangeCipherSpec - Client（6）**：此消息指示来自客户端的所有后续流量将使用所选（协商的）批量加密协议进行加密，并将包含协商的MAC。名义上，这个消息总是用当前密码加密的，在连接的初始状态下，它是NULL，因此消息在图中显示为明文发送。虽然此消息在协议图中显示为单独发送，但它经常与客户端密钥交换消息级联。

7. **完成 - 客户端（7）**：此消息包含在握手协议期间发送和接收的所有消息，但不包括完成消息，并且使用协商的批量加密协议加密并使用协商的MAC进行散列。如果服务器可以使用其独立计算的会话密钥来解密和验证此消息（包含所有先前的消息），则对话成功。如果不是，会话在此时被服务器发送具有一些（可能是模糊的）错误消息的Alert消息而终止。

注意：

RFC 7918规定在某些条件下，客户端可以在发送该消息之后立即开始发送数据，以减少连接延迟。随后处理以下服务器消息失败将导致中止连接，但没有数据泄露。

8. ChangeCipherSpec - 服务器（8）：此消息指示来自服务器的所有后续流量将使用协商的批量加密协议进行加密，并包含协商的MAC。这个消息通常用当前密码加密，该密码在连接的初始状态为空，因此该消息在图中显示为明文发送。此消息的接收也告诉客户端服务器接收到并能够处理客户端的完成消息。

9. 完成 - 服务器（9）：该消息包含在握手协议期间发送和接收的所有消息，但不包括完成消息，并且使用协商的批量加密协议加密，并且包括协商的MAC。如果客户端可以使用其独立计算的密钥解密此消息，则对话成功。如果不是，客户端将终止与警告消息和一个合适的（可能是模糊的）错误代码的连接。

10. 记录协议：服务器和客户端之间的后续消息使用协商的批量加密协议进行加密，并包括协商的MAC。

笔记：

1. 由客户端和服务器发送的随机值和随后的预主秘密都包括两个八位字节时间值（以避免重放攻击），因此，如在所有密码系统中，客户端和服务器应当使用同步的时间源，例如作为NTP。

2. 当客户端或服务器终止与警报消息的连接时，提供的错误代码可能不准确（并且很少有帮助），以避免向可用于细化攻击的另一方提供信息。

X.509（SSL）证书概述

最初的ITU-T标准是证书获得其臭名昭着的名称，是X.509 - X.500目录规范标准套件之一。在因特网上使用X.509证书由IETF用定义X.509证书格式的RFC 5280来标准化，并且RFC 4210定义用于访问和处理X.509证书请求的证书管理协议（CMP）协议（尽管下面讨论的许多附加协议用于处理和验证证书）。最后，RFC 3739定义了它称之为一个合格证书这是有关电子签名欧盟指令（1999/93 / EC）。

对于好奇的ITU-T X.500目录标准，其中定义了旨在支持X.400邮件服务（基于OSI的错误服务）的DAP（目录访问协议）。IETF希望目录服务没有所有的OSI开销和创建的LDAP（Lighweight Directory Access Protocol）。因此，与X.509证书相关的所有技术架构都源于DAP / LDAP。

X.509使用了一个全新的，美妙的术语世界。具体来说，它使用术语"可分辨名称（DN）"来引用证书中的字段。DN由RFC系列的RFC中的IETF定义- 特别是RFC 4514.也使用术语抽象语法记法1（ASN.1）和对象标识符（OID），它们都在ITU X.680系列中描述，最后，编码使用ITU X.690中描述的区分编码规则（DER）。

还存在与标记为PKCS＃X（其中X是数字）的证书处理相关的大量标准，例如PKCS＃10定义证书签名请求（CSR）的格式。这些参考由RSA实验室定义的标准。由于RFC，例如上面提到的PKCS＃10已经被公布为RFC 2986（由RFC 5967更

新），因此已经复制了许多这些标准，基本上没有改变。除了IETF，RSA和ITU-T外，X.509已经被一些国家和行业组织标准化。观察员和实施者指出，标准的多样性可能导致执行和解释方面的严重问题。

X.509证书提供两种不同的功能：

1. X.509证书（当前为X.509v3）充当公钥的容器，可用于验证或验证终端实体（EE），例如网站或LDAP服务器。实体在证书的主题字段中定义。该课题 - 是在一个专有名称（DN）的形式描述在LDAP背景介绍有关DNS -这是由许多相对标识名（RDN的）每一个都含有数据元素称为属性的。具体地，可分辨名称的CN（commonName）属性（RDN）通常包含由证书覆盖的最终实体。CN的示例可以是网站地址，例如CN = www.example.com。

2. X.509证书由受信任的机构（通常称为证书颁发机构或简称为CA）进行数字签名 - 由证书的颁发者字段中的可分辨名称（DN）标识 - 以确保证书未被篡改与证明（或证明）该主题字段的公钥真的，真的是这个主题的公钥。进一步描述这种信任过程。签名机构可以是认证机构（CA），注册机构（RA）或一些其他中间机构（例如从属CA），或者它可以是自签名的。注意：与用户X的公钥相关联的私钥。

称为证书撤销列表（CRL - 当前为CRLv2）的单独X.509结构提供了由于各种原因已被撤销或失效的证书的信息。CRL本质上是一个老式的"批处理"过程。它们包含已撤销的所有证书的大（ish）列表。如果正在检查（使用其序列号）的证书不在CRL中，则假定它仍然有效。CRL有多个问题：CRL可能只能由CA（证书颁发者）定期更新，因为CRL很大，浏览器只会下载它，如果有的话，勉强和偶尔。总之，这不是一个非常有用或有效的过程。

本节中的大部分信息集中在使用X.509验证服务器通信，X.509也可以用于其他目的，例如个人识别（包括数字签名）和S / MIME， （如果）我们的头停止伤害。

X.509证书类型和术语

X.509（SSL）证书使用令人迷惑的术语范围 - 有时甚至一致地应用，大多数不是，这进一步增加了混乱。即使RFC不严格定义他们的术语，虽然RFC 4210最接近。CA通常提供多种证书类型。除了具有准确含义和定义的EV证书和合格证书之外，这些证书类型主要是营销概念 - 旨在提供不同的价格/功能点 - 因此，通常他们的描述只提供点头的建议 -术语。最后，不是所有的CA都是相等的。提醒读者，虽然大多数认证机构完全专业，定期审计或由国家组织认证，但不是所有（在任何CA网站上查找和跟踪，认证，认证和审计链接）。买方小心（注意说明）是口号。

下面的列表尝试定义最常用的术语，涵盖前面段落的注意事项的证书颁发机构和证书类型。提供的解释从技术文档（主要是源RFC，特别是RFC 4210）和CA网站中挑选出来。

证书机构（也称为CA）：术语证书机构定义为签署证书的实体，其中以下是真实的：发布者和主题字段相同，KeyUsage字段具有keyCertSign集合和/或basicConstraints字段在CA属性设置为TRUE。通常，在链接证书的根CA证书是在链中的最顶层，但RFC 4210定义了一个"根CA"是任何发行者的量，最终实体，例如，浏览器具有由受信任的出而获得的证书带外过程。

注册机构（也称为注册CA）：在某些环境中可能需要注册机构（RA）来处理特定的证书特性，例如，国家证书机构（CA）可授权RA来专门用于个人证书，而另一个可能专门从事服务器证书。RA，如果存在，基本上是行政方便。RA可以签署证书（作为从属CA），但是已经进行了适当的最终实体验证，也可以将该请求传递到根CA以进行签名。

下级机构（也称为下级CA）：通用术语。任何签署证书但不是根CA的实体。一些从属CA（特别是那些完全在根CA所有者的控制下操作的）可以被标记为CA（扩展BasicContraints将存在并且cA将被设置为True）。因此，假设RA签署证书，RA将作为从属CA进行，并且如果在根CA的控制下操作，则RA也可以被标记为CA.

中级机构（aka Intermediate CA）：不定期术语偶尔用于定义创建中间证书的实体，因此可以包含RA或下级CA.

交叉证书（aka Chain或Bridge证书）：交叉证书是其中主题和发放者不相同，但在两种情况下都是CA（BasicConstraints扩展存在并且cA设置为True）的证书。它们通常用于CA已更改其发布策略的一些要素（新密钥到期日或新密钥）或一个CA已由另一CA接管的情况，并且由合并的CA发布的证书链接回新所有者以允许先前发布的根证书退役。术语链证书当在此上下文中应用时，指示已经创建了新的链，并且偶尔可以被称为桥证书（其链接或桥接到新的CA）。交叉证书可以安装在服务器上（作为证书捆绑的一部分 - 参见TLS协议 - 证书下的注意事项），但是当用于向后兼容性时，例如，当EV证书由非EV兼容客户端处理时，交叉证书安装在客户端。除了将cA字段设置为True（坦率地说，几乎没有什么区别），交叉证书是正常的中间证书。

中间证书（aka Chain certificates）：不适用于没有由根CA签名的任何证书的术语。中间证书形成链，并且可以存在从终端实体证书到根证书的任何数量的中间证书。中级证书可以由下级CA，RA和甚至CA直接发布（尽管在技术上这些应该被称为交叉证书）以帮助转换，接管或甚至仅仅区分品牌。在这个上下文中的术语链是无意义的（但是听起来复杂和昂贵），并且简单地指示证书形成链的一部分。

证书捆绑：通用术语，指示多个X.509证书连接到单个文件（通常为PEM格式文件）中。可以在TLS / SSL握手期间发送证书包。通常，在某些CA过渡期间（例如接管，策略更改，密钥更改，密钥到期日期更改等），证书捆绑包用于管理目的。

合格证书：在RFC 3739中定义的术语合格证书涉及个人证书（而不是服务器或终端实体证书），并引用欧洲电子签名指令（1999/93 / EC），该指令的重点是统一定义个人数字签名，授权或认证的目的。具体来说，RFC允许主题字段以优先级顺序commonName（CN =），givenName（GN =）或pseudonym = 包含，并且subjectDirectoryAttributes可以存在并且包含dateOfBirth =，placeOfBirth =，gender =，countryOfCitizenship =和countryOfResidence =。最后，两个新的可选扩展生物计量信息和合格证书语句（qcStatements）在RFC中定义。合格证书通过qcStatements扩展的存在来识别， qcStatements扩展的值为qcStatement-2。大多数国家政府都定义了许多其他领域，以纳入这些证书。在某些情况下，出于真正的原因，在其他情况下，只是弯曲民族肌肉，使执行者的生活彻底悲惨。大多数国家政府都定义了许多其他领域，以纳入这些证书。在某些情况下，出于真正的原因，在其他情况下，只是弯曲民族肌肉，使执行者的生活彻底悲惨。大多数国家政府都定义了许多其他领域，以纳入这些证书。在某些情况下，出于真正的原因，在其他情况下，只是弯曲民族肌肉，使执行者的生活彻底悲惨。

非合格证书：通常是不符合为合格证书定义的标准的个人证书。也可以作为合格证书发行者滥用的条款，暗示质量差的证书。

终端实体证书（aka Leaf Certificate）：很复杂。术语端实体（或端实体，两者可互换地使用）最初在X.509中定义，并且随后在RFC 4949和RFC 5280中定义。在所有情况下的意义是终端实体证书是其中私钥（在终端实体证书中引用的公钥的私钥）用于保护在主题的CN =属性中描述的终端实体，或subjectAltName。否定地，该术语有时用于指示（在最终实体证书中引用的公钥的）私钥不用于签署证书，即，末端实体证书不是中间证书，通常不是根（CA）证书，因此不在任何签名验证过程中使用。术语叶证书用于指示终端实体证书通常是链中的最后一个证书。这种术语是否有助于或阻碍理解是可以猜测的。

多主机证书：服务器证书通常包含一个CN =主机名属性，例如CN = www.example.com，在主题。主机名由DNS解析，并可以生成多个IP地址（如果DNS中有多个A或AAAA RR）。在这种情况下，相同主机名的任何X.509（SSL）服务器证书都可以复制到所有这样的主机上（显然，用户私钥还需要复制到每个主机，如果使用硬件加密设备，可能会出现问题 - 在此后一种情况下，一些CA将很高兴向您出售额外的证书称为多主机证书，这绕过了问题）。存在多个主机名的情况，例如www.example.com和example.com或www1.example。

多域证书：某些CA销售多域证书以涵盖www.example.com和example.com或甚至www.example.net等情况。这通过在subjectAltName字段中使用多个条目来实现，并且进一步描述。从技术上讲，这个过程没有什么限制，例如，www.example.com和www.example.net可以由单个X.509（SSL）证书支持，但大多数CA具有一些商业限制 - 这通常可以克服如果你部分与更肮脏的lucre。通配符证书（如下所述）有时可用于此目的，但仅限于单个域名。

通配符证书：许多CA销售通配符证书，其中主题字段包含CN = *。example.com（*是通配符）。此类证书将支持域中的任何主机名，因此*.example.com将支持www.example.com和mail.example.com，但不支持example.com或（显然）example.net（请参阅上面的多域证书）。

EV证书（又名扩展证书）：扩展验证（EV）证书是由存在区别CertificatePolicies含注册OID在扩展policyIdentifier领域。详细描述EV证书。

DV证书：有时称为域验证（DV）证书由某些CA颁发。该术语不是通用的，而是意味着证书请求者只有域名的所有权已经由CA验证。因此，主体或subjectAltName中的CN =值（例如www.example.com）可以被视为有效，但组织信息（C =，ST =，L =，OU =或O =）不应被视为有效并且应为空白或包含适当的文本，例如“无效”。

OV证书：有时称为组织验证（OV）证书由某些CA颁发。该术语未被普遍使用，但暗示证书请求者的域名的所有权及其组织细节已经由CA验证。因此，主体或subjectAltName中的CN =，C =，ST =，L =，OU =或O =值可被视为有效。虽然这看起来，在其表面上，相当详尽的证书不是需要进一步资格的EV证书。

仅域名证书：通用术语（通常为滥用）适用于其最终用户验证过程在术语用户视图中从粗略到不存在的变化的证书。

数字传输内容保护（DTCP）：这些证书由数字传输许可管理员（DTLA-www.dtcp.com）发布和控制，并且当使用许可的材料时，通常由智能TV，媒体播放器等使用。DTCP证书不使用X.509格式，但它们可以在TLS握手协议（RFC 7562）中使用。它们不在本页进一步描述。

X.509证书链

X.509证书可以链接，也就是说，它们可以由一个或多个中间授权机构以分层方式签名，或者证书可以简单地由CA直接签名。在上述RFC中引入了注册机构（RA）作为中间签名中心的概念。注册机构（RA）（有时在EV标准中也称为从属CA）似乎描述实际购买X.509证书的组织，例如，使用CA签署证书的许可代理（最终签署机构）提供最终或根权威。在结构上与DNS注册管理运行机构和注册服务商类似，熟悉DNS组织。

签名层次结构的最上层证书称为根证书，有时是CA证书或甚至根CA证书。根证书通过可信任的带外过程获得（在浏览器的情况下，它们与浏览器软件一起分发并且周期性地更新），并且当被用于验证证书链时通常被称为信任锚。当在TLS / SSL握手期间从服务器获得终端实体证书（或证书包）时，必须由接收软件一直验证到根证书或CA证书，包括（如果适当的话）任何中间证书（再次是这些证书通常使用浏览器软件分发）。如果发行者和主题字段相同，则认证根证书或CA证书，如果KeyUsage字段具有keyCertSign集合和/或BasicConstraints字段的cA属性设置为TRUE。RFC 4158描述了构建证书链的过程，RFC 5280进行了链验证。链接过程如下图3所示：

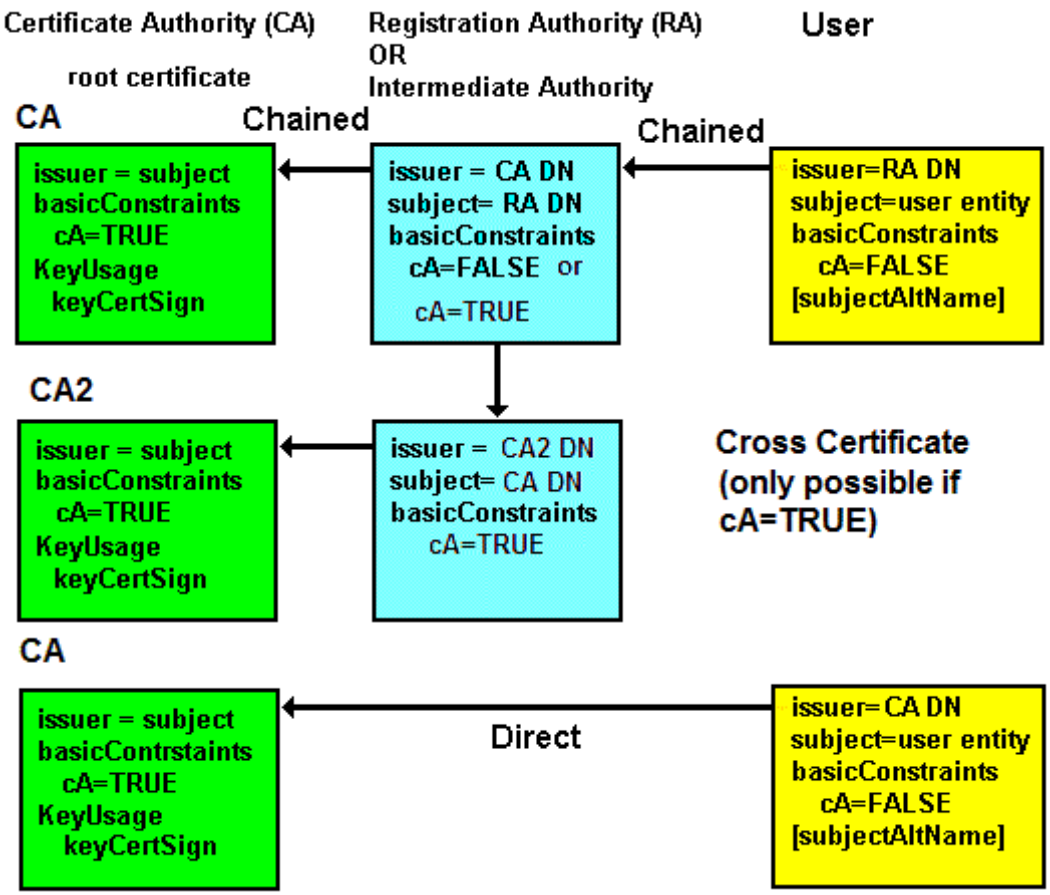

图3 - X.509链接

X.509证书使用

证书的颁发者使用最初设计为表示DAP或LDAP DIT（数据信息树）中的位置的可分辨名称（DN）格式来标识。DN不应与网络地址或URL / URI混淆。DN通常具有诸如CN =证书类型，OU =证书分区，O =证书公司名称，C =国家（CN =，

OU =，O =，C =格式）的格式，但可以简单地使用OU = ，O =，C =格式或甚至CN =，O =，C =格式，并且使事情更复杂，它可以使用CN =，OU =，DC =，DC =格式。它可以变化 - 很多。DN由多个逗号分隔的RDN（相对识别名）组成，因此CN =或C =是DN中的RDN。一个X. 509证书不包含通过网络接口获取任何链接证书的URI - 但是它可能包含（在其他字段中）用于获取CRL的URI 。使用证书的应用程序（如浏览器或客户端电子邮件软件）必须先获得根证书，并且如果证书链接 - 所有中间证书，通过某些带外或离线过程。最常见的CA根证书随浏览器分发（并提供给其关联的客户端电子邮件软件）。使用常见浏览器处理证书。通过一些带外或离线过程。最常见的CA根证书随浏览器分发（并提供给其关联的客户端电子邮件软件）。使用常见浏览器处理证书。通过一些带外或离线过程。最常见的CA根证书随浏览器分发（并提供给其关联的客户端电子邮件软件）。使用常见浏览器处理证书。

注意：使用通用浏览器（和电子邮件客户端）软件分发的根证书根据浏览器供应商定义的标准添加，并且从"支付我们大量现金"到充分证明CA审核和其他要求有所不同。

当应用程序访问基于TLS / SSL的服务时，在初始TLS / SSL握手对话期间获取服务器证书（或证书包）。应用程序（例如浏览器）将提取主题 DN字段的CN （和/或subjectAltName - 参见RFC 6125）以验证实体（例如web服务器地址，例如www.example.com）。然后，它使用服务器的X.509证书的issuer字段DN从其本地存储中查找适当的根证书（并生成异常 - 通常包含高度混乱且可能有危险的用户对话框（如果找不到）。如果找到有效的根证书，则认证服务器提供的证书。最终结果，

服务器通常只需要自己的证书，并且不需要根证书 - 它们只是将证书发送到客户端，而不必验证证书。但是TLS / SSL允许相互验证 - 服务器和客户端都发送证书。如果应用程序中需要客户端证书，则服务器需要验证客户端证书，并且必须通过某些带外过程（例如电子邮件或从CA网站获取）提供所有必需的根证书和中间证书。

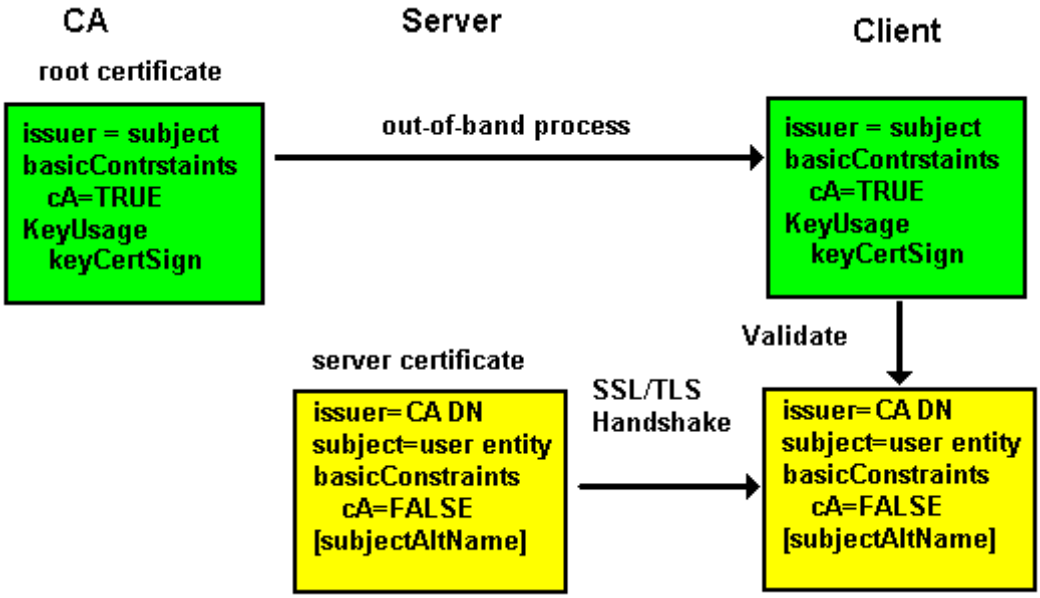

图4 - X.509使用

关于证书主题和subjectAltName的附加注释

X.509网络托管组织中的证书

网站所有者正面临着巨大的压力，要从各种有意的（但极其错误的）来源实施**X.509（SSL）**。

在网站所有者是网站运营商的情况下，这呈现出几个问题。网站所有者/运营商只需要决定是否在实施**TLS**方面具有成本效益。然而，许多网站所有者已经选择将站点操作委托给专门的**web**托管公司（所谓的多租户站点）。这里的问题变得更加严重。

那么什么问题呢?

该问题涉及**X.509**证书的颁发和与那些**X.509**证书相关联的私钥的所有权。特别地，存在要求服务器访问证书的私钥的单个（在大多数情况下）事务（在该**TLS**协议的该图中的项（**5**））（回想在公共 - 私钥系统中客户端用公钥加密，只有私钥的所有者（在这种情况下为服务器）可以解密消息）。

现在，假设**example.com**的所有者已将其网域的网站的操作委派给域名为**example.net**的网站托管组织。如果用户的浏览器在**www.example.com**上连接到**TLS**服务，则希望看到名称为**www.example.com**（连接到的网站的名称）的证书。如果它收到一个名称为**www.example.net**（主机组织的域名）的证书，它会得到一点失望（事实上，它得到快乐生气，开始输出讨厌的消息或诉诸愤怒的颜色（红色）在地址栏）。

为了缓解这个问题，**RFC 6066**引入了**SNI**（服务器名称指示），它允许客户端（浏览器）明确地声明它正在**ServerHello**消息中连接到**www.example.com**，使我们的超级智能服务器能够提供期望的**example.com**）证书。**hew**，固定那一个。快乐的浏览器又在这里，**tra**。

不是那么快。

没有正当的证书颁发机构（**CA**）会将**example.com**的证书颁发给**example.net**，只有域所有者（必须以某种方式证明所有权）才能获取其域的证书。然而，**TLS**图上的项目**5** 提醒我们，主机服务器不仅需要证书，而且需要与证书相关联的私钥。哎哟。提示律师从阶段左进入（最可能阶段右和中心也）。双**uch**。

RFC 7711提出了一个解决方案，借此用户（**example.com**）可以（使用**DNS SRV**资源记录（**RR**））通过指向**web**托管的**JSON**格式化来将**SSL**证书覆盖显式委托给第三方（在我们的**example.net**中）记录。用户不必购买昂贵的**SSL**证书，因此不必向其托管提供商提供其私钥。理论上，在访问用户的网站之前，我们的浏览器将进行**DNS SRV**查找，然后，使用生成的**URL**，将读取一个委托记录（假设在这种情况下委托给**example.net**）。收到此信息后，将非常高兴接受来自**example.net**的证书，当它连接到**example.com**。我们的浏览器不会生气。它不会输出讨厌的信息，它不会打开愤怒的颜色。除了律师，每个人都会乐意开心。

注意： RFC还可以选择允许用户指示已经故意给予其主机提供商的X.509证书（和隐式其私钥）。

还有另一个可能的解决方案使用DNSSEC和DANE，我们将轮到记录这些美好的日子之一。只是不要保持你的呼吸。

关于证书主题和subjectAltName的附加注释

证书协议（CMP，CMC / CMS，CRMF，SCVP，OCSP，HTTP）

X.509证书是一种数据结构。各种协议允许经由通信网络操纵证书。这些**X.509证书操作**（例如发送签名请求，返回签名的证书等）在其中包括证书管理协议（CMP） - RFC 4210以及PKCS＃10（RFC 2986）中定义，并且进一步在本指南中描述。

概述：定义了许多协议来操作证书和CRL（证书吊销列表）。证书管理协议（由RFC 6712更新的CMP RFC 4210）提供了操纵证书的协议方法（证书请求消息格式（CRMF）的格式在RFC 4211中定义）。RFC 4210定义了许多可用于通过网络传输证书请求但不明确定义用于证书处理的传输协议的通信方法（例如HTTPS）（参见下一个CMC / CMS）。

在RFC 5272，RFC 5273，RFC 5274中定义并且由RFC 6402更新的CMC / CMS（CMS上的证书管理）允许将证书请求从请求者到CA（包括任何中间RA）和返回的安全传输。消息格式可以是PKCS＃10（RFC 2896）或CRMF（RFC 4211）。此协议应始终称为CMC，但偶尔称为CMS。技术上，CMS是加密消息语法，并且仅描述请求和响应的包络格式（在PKCS＃10或CRMF格式中）。该协议定义了可以在证书上执行的多个操作，包括更新由证书请求者维护的信任锚（根证书）。这是一个大而多汁（阅读复杂）协议。

在线证书状态协议（OCSP RFC 6960）是用于证书在线验证的协议。RFC 6066扩展TLS以允许客户端在握手协议阶段期间请求OCSP证书状态（并且RFC 6961定义了简化的"certificate_request_v2"，其尝试减少OCSP服务器业务量）。

基于服务器的证书验证协议（SCVP RFC 5055）允许将多个客户端功能委派给不受信任的服务器，特别是证书路径验证和证书路径发现。在SCVP服务器是可信的情况下，可以提供诸如证书状态（也由OCSP提供）和获得中间证书（也由CMP提供）的功能。

RFC 4387定义了使用GET方法（这是我们上次为X.509证书支付肮脏的lucre时使用的方法）对X.509证书，密钥或CRL的一些有限操纵。RFC 4386定义了使用DNS SRV RR来发现证书存储库和OCSP服务。

证书管理协议（CMP）

证书管理协议（CMP）在RFC 4210（由RFC 6712更新）中定义，并且消息格式在证书请求消息格式（CRMF RFC 4211）中定义。

供应。

基于服务器的证书验证协议（SCVP）

供应。大部分被OCSP变种取代。

通过CMS（CMC / CMS）进行证书管理

供应。

Online Certificate Status Protocol (OCSP)

The Online Certificate Status Protocol (OCSP) is defined in RFC 6960 and a streamlined, high-throughput, message format is defined in RFC 5019 (The Lightweight Online Certificate Status Protocol (OCSP) Profile for High-Volume Environments) which, in essence, sensibly reduces OCSP requests to a single certificate and removes most of the OPTIONAL fields in requests and responses as noted below. RFC 6960 (which obsoletes RFC 2560 and 6277) simply clarifies a few points in the original RFCs, makes the RFC more compatible with RFC 5019 and adds its own, entirely new, obfuscation to the specification. RFC 6961 defines a new 'certificate_request_v2' which allows servers to cache (save) responses and allows information about all relevant certificates (including intermediary ones) to be sent in a single message request.

OCSP is an on-line alternative to a Certificate Revokation List (CRL). The URI of the service is typically identified in the AuthorityInfoAccess (AIA) field of a certificate if the CA supports an OCSP service (issuers of EV certificates are mandated to support OSCP). A client will send an optionally signed request identifying the certificate to be verified and receive a signed reply from the OCSP server indicating the status as being good, revoked or unknown. The RFC allows for a number of different transport protocols (and specifically mentions LDAP, SMTP and HTTP as examples) to transmit the request and responses with the specific transport scheme identified in the URI of the AIA field of the certificate being validated. The RFC also defines the format of HTTP messages (using GET and POST) when used for OCSP. The request and response messages are outlined below:

```
# request format
OCSPRequest
  TBSRequest
    version                      0 = v1
    requestorName                OPTIONAL
    requestList (one or more)    (single request in the case of RFC 5019)
      reqCert
        hashAlgorithm            AlgorithmIdentifier
        issuerNameHash           Hash of Issuer's DN
        issuerKeyHash            Hash of Issuers public key
        serialNumber             CertificateSerialNumber
        singleRequestExtensions  OPTIONAL
    requestExtensions            OPTIONAL
  optionalSignature              OPTIONAL
```

Notes:

1. Signature of the OCSP request is optional and if present the **requestorName** will indicate the signer. Clearly, for the receiver to verify the signature it must have the public key of the signer (or its delegated agent).

2. Providers of EV (Extended Validation) certificates must provide an OCSP service, for all other certificate types it is optional.

```
# response format
OCSPResponse
  responseStatus
    successful        0  --Response has valid confirmations
    malformedRequest  1  --Illegal request format
```

```
   internalError     2  --Internal error in issuer
   tryLater          3  --Try again later
                        --(4) is not used
   sigRequired       5  --Must sign the request
   unauthorized      6  --Request unauthorized
 responseBytes       OPTIONAL
   responseType      OID (1.3.6.1.5.5.7.48.1.1 =BasicOCSPResponse)
     BasicOCSPResponse
      response
       ResponseData
         version          0 - Version DEFAULT v1
         responderID      EITHER OF
           byName         1 - Name
           byKey          2 - Hash of Issuers Public Key
         producedAt       GeneralizedTime
         responses
           certID
             hashAlgorithm       AlgorithmIdentifier
             issuerNameHash      -- Hash of Issuer's DN
             issuerKeyHash       -- Hash of Issuers public key
             serialNumber        CertificateSerialNumber
           certStatus       CertStatus
             good           0
             revoked        1
             unknown        2
           thisUpdate       GeneralizedTime
           nextUpdate       GeneralizedTime OPTIONAL
           singleExtensions OPTIONAL
         responseExtensions OPTIONAL
       signatureAlgorithm   AlgorithmIdentifier,
       signature            Hash of Response data
       certs                OPTIONAL
```

Notes:

1. A **certStatus** of **good** is defined in the RFC to mean 'not revoked' but additional information may be available in Extension fields.

2. The **responseStatus** of **unauthorized** indicates the responder has no authoritative information about this certificate.

3. A number of extensions are defined in RFC 6960 and in addition any of the CRL Extensions (RFC 2459) may be included.

4. The response is normally signed by the CA that issued the certificate identified in **serialNumber** but the protocol allows for a delegated authority to sign the response in which case the response must include a certificate (carrying the delegated signers public key) in **certs** and which must be signed by the issuer of the certificate defined in **serialNumber**.

5. RFC 5019 defines a streamlined message format - which is entirely compatible with the full OCSP format - by removing most of the OPTIONAL fields to assist message throughput. PROTOCOL: Supports OCSP over HTTP only using GET and POST methods. REQUESTS: Limits the request to a single certificate (**requestList** will be 1), dispenses with the OPTIONAL field **singleRequestExtensions**, and the OPTIONAL structures **requestExtensions** and **optionalSignature** (if the request is signed, responders are free to ignore the signature). RESPONSES: By insisting on a single certificate per request RFC 5019 has already

reduced complexity (and size) in the response, in addition the OPTIONAL **responseExtensions** is removed (but **singleResponseExtensions** may be included). Again, if the response is signed by a delegated authority the response must include a certificate (carrying the delegated signers public key) in **certs** and which must be signed by the issuer of the certificate defined in **serialNumber**.

OCSP Issues

OCSP is designed as a real-time system (unlike the batch nature of classic CRLs) thus, clients can theoretically, verify the status of any certificate before acceptance and use. Particularly when handling an EV certificate any client implementation, for example, a browser is really obliged to perform an OCSP check to meaningfully implement EV security. This means, for example, that every access to an HTTPS service can (in the case of EV should) result in an additional check to the OCSP service of the CA. As more sites implement the well intentioned, but ultimately misguided, policy of using HTTPS for everything (rather than the more meaningful and secure DNSSEC) OCSP server performance is rapidly degraded and servers are brought to their knees in a kind of benignly intended DDoS onslaught.

RFC 6066 extends TLS by allowing the client to request certificate status from the TLS server (**OCSPStatusRequest**) and may have further exacerbated the OCSP load problem by making the OCSP interrogation trivial to implement - encouraging every client to request it. RFC 6961 tries to fix the OCSP load problem by using a new TLS 'certificate_request_v2' which seems (it is not definitive on when a server should interrogate OCSP) to allow caching of OCSP responses (thus reducing OCSP loads at the expense of real-time validity checks) at the server and enables multiple certificate status messages (including all intermediate certificates) to be sent in a single frame. And finally, one hopes, because all those intermediate certificates can build up to a serious volume RFC 7924 defines a method whereby the client can tell the server that it already has all that intermediate stuff.

X.509 Certificate Format

This section describes, in gruesome - but incomplete - detail, the format and meaning of the major fields (technically called attributes) within an X.509 certificate. The format is defined in RFC 5280 (Updated by RFC 6818).

An X.509 certificate is an ASN.1 structure encoded using the Distinguished Encoding Rules (DER) of X.690 and includes multiple references to globally unique Object Identifiers (OID)..

Note:

1. Much use is made in X.509 (and LDAP) of that gruesome pseudo-Hungarian notation (or lowerCamelCase if you prefer the term). In general, poor X.509 implementations are case sensitive, good ones are not. Further, all the LDAP matching rules related to DN handling are case insensitive meaning that attribute names are not case sensitive.

2. RFC 7250 defines a vestigal certificate format for cases where the public key has been obtained by other (out-of-band) trusted methods. This minimal certificate format uses only the subjectPublicKeyInfo attribute (and its two sub-fields). Use of this minimal certificate format is indicated during the ClientHello and ServerHello messages during the TLS/SSL handshake.

An X.509 certificate printed by Openssl looks like this (the major fields are described below):

```
Certificate:
 Data:
  Version: 3 (0x2)
  Serial Number:
   bb:7c:54:9b:75:7b:28:9d
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: C=MY, ST=STATE, O=CA COMPANY NAME, L=CITY, OU=X.509, CN=CA ROOT
  Validity
   Not Before: Apr 15 22:21:10 2008 GMT
   Not After : Mar 10 22:21:10 2011 GMT
  Subject: C=MY, ST=STATE, L=CITY, O=ONE INC, OU=IT, CN=www.example.com
  Subject Public Key Info:
   Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
     Modulus (1024 bit):
      00:ae:19:86:44:3c:dd...
      ...
      99:20:b8:f7:c0:9c:e8...
      38:c8:52:97:cc:76:c9...
   Exponent: 65537 (0x10001)
 X509v3 extensions:
  X509v3 Basic Constraints:
   CA:FALSE
 Netscape Comment:
  OpenSSL Generated Certificate
 X509v3 Subject Key Identifier:
  EE:D9:4A:74:03:AC:FB...
 X509v3 Authority Key Identifier:
  keyid:54:0D:DE:E3:37...

 Signature Algorithm: sha1WithRSAEncryption
  52:3d:bc:bd:3f:50:92...
  ...
  51:35:49:8d:c3:9a:bb...
  b8:74
```

Note Lines have been truncated and omitted (replaced with ... in both cases) in the above since the deleted material does nothing to aid understanding of the process.

The X.509 certificate consists of the following fields (Attributes):

Version	In general this should indicate version 3 (X.509v3). Since the numbering starts from zero this will have a value of 2. If omitted

	version 1 (value 0) is assumed.
Serial Number	Positive number up to a maximum of 20 octets. Defines a unique serial number for each certificate issued by a particular Certification Authority (thus is not, of itself, a unique number) and used, among other purposes, in Certificate Revocation Lists (CRLs).
Signature	Must be the same OID as that defined in SignatureAlgorithm below.

Issuer	The DN (Distinguished Name) of the Authority that signed and therefore issued the certificate. This may or may not be the Certification Authority (CA). The **issuer** may comprise a subset of domainComponent (DC=), countryName (C=), commonName (CN=), surname (SN=), givenName (GN=), pseudonym=, serialNumber=, title=, initials=, organizationName (O=), organizationalUnitName (OU=), stateOrProvinceName (ST=) and localityName (L=) attributes. Only CN=, C=, ST=, O=, OU= and serialNumber= must be supported the rest are optional (serialNumber= is rarely present - but mandated for EV certificates - and L= is frequently present though widely misunderstood but its use is clarified for EV certificates) RFC 5280 appears to allow either of the globally unique methods (X.500 format O=, C= or the IETF format DC=, DC=) but the X.500 format seems universally preferred. An example issuer DN would look something like:
	
	`# shown split across two lines for presentation reasons only`
	`C=MY,ST=some state,L=some city,O=Expensive Certs inc,`
	`OU=X.509 certs,CN=very expensive certs`
	`# various interpretations of the RDN fields exist`
	`# the following are presented as generally accepted`
	`# values.`
	`# C = ISO3166 two character country code`
	`# ST = state or province`
	`# L = Locality; generally - city`
	`# O = Organization - Company Name`
	`# OU = Organization Unit - typically certificate type or brand`
	`# CN = CommonName - typically product name/brand`

Validity	Provides two sub-fields (Attributes) **notBefore** and **notAfter** which define the time period for which the certificate is valid - or if one is of a cynical persuasion the **NotAfter** value defines the point at which the user has shell out more filthy lucre to the CA or issuer. Dates up to 2049 are encoded as UTCTime (format is YYMMDDHHMMSSZ - yes a 2 digit year) after 2050 as GeneralizedTime (format is YYYYMMDDHHMMSSZ - finally a 4 digit year). Where Z is a literal constant indicating Zulu Time (UCT aka Greenwich Mean Time) - its absence would imply local time.
Subject	A DN defining the entity associated with this certificate. If this is a root certificate then issuer and subject will be the same DN (and BasicConstraints CA TRUE will be set). In an end user/server certificate the subject DN identifies in some way the end entity.

Typically the CN attribute (RDN) of the DN will identify some unique name of the end entity being certified or authenticated. The **subject** may comprise a subset of domainComponent (DC=), countryName (C=), commonName (CN=), surname (SN=), givenName (GN=), pseudonym, serialNumber, title, organizationName (O=), organizationalUnitName (OU=), stateOrProvinceName (ST=) and localityName (L=) atributes. An example subject DN that will authenticate access to a web site could look something like:

```
# shown split across two lines for presentation reasons
only
C=MY,ST=another state,L=another city,O=my
company,OU=certs,
 CN=www.example.com
# various interpretations of the RDN fields exist
# the following are presented as generally accepted
# values. In the case of personal certificates GN=, SN=
or pseudonym=
# can appear in the fields
# C = ISO3166 two character country code
# ST = state or province
# L = Locality; generally means city
# O = Organization - Company Name
# OU = Organization Unit - division or unit
# CN = CommonName - end entity name e.g. www.example.com
```

However some certs use the issuer's DN and replace the CN attribute (RDN) with the name of the entity being authenticated thus the above could, based on the issuer example above, become:

```
# shown split across line for presentation reasons only
C=MY,ST=some state,L=some city,O=Expensive Certs inc,
 OU=X.509 certs,CN=www.example.com
```

In the above example CN=www.example.com will work when the web site access URL is http://www.example.com, if the web site access also uses http://example.com the certificate checking process will fail and in this case the subjectAltName can be used to extend the certificate scope to include example.com (or any other domain name that is covered by this certificate).

While the majority of certificates currently (2011) use the CN= RDN of the **subject** field to describe the entity, RFC 6125 now recommends that the subjectAltName be used (containing the end entity name) and that the **subjectAltName** field be used in preference to CN= in the **subject** for validating the entity name. It may be some time before all applications and certificate issuers follow these principles and therefore to handle all possibilities the entity should appear in both a CN= (in the **subject** field) and in a **subjectAltName** field.

The **subject** field can be empty in which case the entity being authenticated is defined in the subjectAltName.

The form CN=www.example.com/emailAddress=me@example.com is frequently seen and generally created by default by OpenSSL tools

	when generating a Certificate Signing Request (CSR) and defines a second attribute (**emailAddress**) and which may, or may not, be present. Most CAs request that the email address prompt is left blank during the creation of the CSR when using OpenSSL - perhaps because they can sell you a second SSL certificate to protect your email addresses. Or is that excessively cynical? Additional Notes on Certificate Subject and subjectAltName.
SubjectPublicKeyInfo	Contains two sub-fields (attributes), **algorithm** (the OID of the public key algorithm from the list defined in RFC 3279) and **subjectPublicKey** (the entity's public key as a bit string). An example of a the **algorithm** attribute when using an RSA algorithm (rsaEncryption) OID is shown below: `1.2.840.113549.1.1.1` **Note** The vestigal certificate format defined by RFC 7250 consists only of this attribute and its two sub-fields.
IssuerUniqueIdentifier	Optional. Defines a unique value for the issuer. The RFC recommends that this field is NOT present.
SubjectUniqueIdentifier	Optional. Defines a unique value for the entity being authenticated. The RFC recommends that this fields is NOT present.
Extensions	Tons of extensions are defined in the various RFCs - the following are only the most, IOHO, significant. Extensions may be marked as CRITICAL. **Note:** CRITICAL has an interesting and nuanced interpretation. If the software handling the certificate sees the CRITICAL value (which it can always interpret) but does not understand the extension it MUST abandon processing and NOT accept the certificate. **Notes:** 1. RFC 5280 defines standard extensions (defined by the original X.509 standards under the OID 2.5.29) and what it terms Private Extensions (defined under the OID 1.3.6.1.5.5.7.1). The Private Internet extension OID numbering is maintained by IANA. Unless noted otherwise below the following extensions are standard (under OID 2.5.29). 2. RFC 7633 defines a Private Internet X.509 certificate extension (OID 1.3.6.1.5.5.7.1.24 id-pe-tlsfeature) that allows a certificate to include TLS feature extensions (identified by their TLS code values) - essentially allowing the client to detect a potentially fraudulent server. Thus, if the certificate contains the TLS feature extension status_request but the server did not return this information (in a **CertificateStatus** message) then the client may conclude the session is potentially insecure.

AuthorityInfoAccess	Private Internet Extension (OID 1.3.6.1.5.5.7.1.1) Frequently known by the abbreviation AIA (though this is not its LDAP alias name - it does not have one). Used to contain information about CA services including any Online Certificate Status Protocol (OCSP). Interestingly while EV Certificates demand a 24/7 certificate status service and most frequently provide it using OCSP the actual EV standard does not reference the use of this field.
	
	`AuthorityInfoAccess = AuthorityInfoAccessSyntax`
	`# multiple AuthorityInfoAccessSyntax elements may exist`
	`# each is comprised of:`
	`accessMethod = OID`
	`# may take the values:`
	`# 1.3.6.1.5.5.7.48.1 = ocsp`
	`# accessLocation = URI of the CA's OCSP service`
	`# for the issuer CA`
	`# 1.3.6.1.5.5.7.48.2 = caIssuers`
	`# used only if certificate signer is not`
	`# the root CA`
	`# accessLocation = URI of root CA description`
	
	`accessLocation = URI (exceptionally an email address`
	`or X.500 DirectoryString)`

authorityKeyIdentifier	[OID: 2.5.29.] Optional. May contain three fields:
	
	`keyIdentifier (0) KeyIdentifier`
	`authorityCertIssuer (1) GeneralNames`
	`authorityCertSerialNumber (2) CertificateSerialNumber`
	
	[OID 2.5.29.35] The standard recommends the use of the **keyIdentifier** value for all but root certificates. The **keyIdentifier** is normally a 160 bit SHA-1 hash of the subjectPublicKeyInfo but other methods are defined. Presence of this field facilitates certificate path (chain) creation and allows CAs to have multiple root certificates each of which may have a different key referenced by this extension.

subjectKeyIdentifier	[OID: 2.5.29.14] Optional but the standard recommends the use of this value in all certificates as an aid to certificate path (chain) construction. The **SubjectKeyIdentifier** is normally a 160 bit SHA-1 hash of the subjectPublicKeyInfo but other methods are defined.

KeyUsage	[OID: 2.5.29.15] Defines the purposes for which the public key may be used and may take the following values:
	
	`digitalSignature (0)`
	`nonRepudiation (1)`
	`keyEncipherment (2)`
	`dataEncipherment (3)`
	`keyAgreement (4)`
	`keyCertSign (5) # indicates this is CA cert`
	`cRLSign (6)`
	`encipherOnly (7)`
	`decipherOnly (8)`

If keyCertSign is set then BasicConstraints cA TRUE MUST also be set, however if BasicConstraints CA TRUE is present then KeyUsage keyCertSign need not be present.

ExtendedKeyUsage	[OID: 2.5.29.37] When present refines the purposes for which the public key may be used and must be compatible with the **KeyUsage** field. It may take the following values:
	
	`serverAuth (1) TLS WWW server authentication`
	`(valid with digitalSignature, keyEncipherment or keyAgreement)`
	`clientAuth (2) TLS WWW client authentication`
	`(valid with digitalSignature or keyAgreement)`
	`codeSigning (3) Signing of downloadable executable code`
	`(valid with digitalSignature)`
	`emailProtection (4) Email protection`
	`(valid with digitalSignature, nonRepudiation, and/or (keyEncipherment or keyAgreement)`
	`timeStamping (8) Binding the hash of an object to a time`
	`(valid with digitalSignature and/or nonRepudiation)`
	`OCSPSigning (9) Signing OCSP responses`
	`(valid with digitalSignature and/or nonRepudiation)`
	
	If keyCertSign is set then BasicConstraints cA TRUE MUST also be set, however if BasicConstraints CA TRUE is present then KeyUsage keyCertSign need not be present.

Basic Constraints	[OID: 2.5.29.19] A boolean which defines whether the certificate is a CA or root certificate (TRUE) or not (FALSE). Can take an optional attribute (pathLenConstraint) which defines the maxim chaining depth.
	
	`cA TRUE | FALSE`
	`pathLenConstraint INTEGER`

CRL Distribution Points	[OID: 2.5.29.31] Optional but RECOMMENDED by the RFC (go figure). Defines one or more URLs (and other optional information) where Certificate Revocation Lists (CRLs) may be obtained for the CA that issued the certificate (the **issuer**). Each CRL location (called a DistributionPoint) has the following format:
	
	`# (O) = OPTIONAL`
	`distributionPoint DistributionPointName (O)`
	`# points to a structure defined below`
	`reasons ReasonFlags (O)`
	`cRLIssuer GeneralNames (O)`
	`# DN of CRL Issuer if not the same as this`
	`# certificates issuer`
	`# the DN could be used in an LDAP search request`
	
	`# DistributionPointName is either`
	`fullName GeneralNames`
	`# may contain either a URI`
	`# e.g. http://crl.example.com`
	`# OR a Protocol name such as HTTP, LDAP, FTP etc.`
	`# used to obtain the CRL`
	`# OR`
	`nameRelativeToCRLIssuer`
	`# RDN appended to issuer DN to get CRL`

```
# The ReasonFlags may take one of the values:
unused                  0
keyCompromise           1
cACompromise            2
affiliationChanged      3
superseded              4
cessationOfOperation    5
certificateHold         6
privilegeWithdrawn      7
aACompromise            8
```

Multiple entries are allowed.

CertificatePolicies	[OID: 2.5.29.32] Optional. Can be used to identify the particular policies of the issuer CA. The extension allows both an OID (in **CertPolicyId**) and other optional attributes that essentially provide displayable text but the RFC RECOMMENDS only the use of the OID. The OID in this field is also used to identify a EV Certificate. May take any of the attributes below:

```
policyIdentifier   CertPolicyId # OID
policyQualifiers  # multiple values allowed
                  # typically a URI to a text
                  # statement of policy or just
text
```

subjectAltName	[OID: 2.5.29.17] Sometimes abbreviated as SAN. Optional, but RFC 6125 recommends that for server certificates this field always be present and contain the entity name for which the certificate is issued (the rationale being that the **dNSName** field was defined to contain a server name whereas the CN= field of the **subject** can contain a multitude of formats). Can also be used to extend the entities covered by the certificate (the subject is non-empty) or as an alternative (subject is empty and the **subjectAltName** extension is marked as CRITICAL). May take any of the attribute types below:

```
otherName                   type=, value= pairs
including
                            Kerberos names (RFC
4556):
                             oid = 1.3.6.1.5.2.2
                             kerberos-principal
(IA5String)
                            OR
                            SRVName (RFC 4985):
                             oid = 1.3.6.1.5.5.7.8.7
                             srv-name (IA5String)
rfc822Name                  email me@example.com
dNSName                     DNS Name
host1.example.com
x400Address                 If you are into
                            X.400 mail addresses
directoryName               Alternative DN
ediPartyName                EDI stuff
uniformResourceIdentifier   URI
ldap://ldap.example.com
iPAddress                   IP V4/V6 192.168.0.1
registeredID                OID
```

Multiple entries are allowed. Use of **subjectAltName** is a way of handing the problem where a server may appear under

multiple names in the DNS. For example, if a server is accessed using https://www.example.com and https://example.com then www.example.com could appear in the CN= part of the subject field and example.com could appear as a dNSName field or more commonly both www.example.com and example.com would appear as dNSName entries. **Note:** Any domain name can appear in these fields, for example, www.example.com and www.example.net could both appear. There is no requirement for a common domain name root. Not all CAs support use of **subjectAltName**. Generating subjectAltName entries is a tad messy with OpenSSL self-signed certificates and the process is fully detailed. Additional Notes on Certificate **subject** and **subjectAltName**.

SignatureAlgorithm	The algorithm used to sign the certificate identified by its OID and any associated parameters. The following illustrates the RSA with SHA1 digital signature OID (sha1WithRSAEncryption): `1.2.840.113549.1.1.5` This value must be the same as that identified in the signature field of the certificate. The currently valid OIDs for use with X.509 certificates are defined in RFC 3279.
SignatureValue	Bit string containing the digital signature.

All fields with the exception of **SignatureAlgorithm** and **SignatureValue** are encoded using ASN.1 DER (Distinguished Encoding Rules) defined by ITU-T X.690. The signature covers all DER encoded fields thus, obviously, excluding itself.

Multiple standards bodies (countries, industry organizations or governmental agencies) define specific profiles that refine the meaning of certain fields or attributes or mandate specific fields. All very confusing.

Certificate Subject and subjectAltName Notes

X.509 Certificates are used for a number of purposes. The following notes describe the constraints or additional rules used or imposed to cover certain functionality.

X.509 Certificate Domain Name Validation

RFC 6125 defines a set of rules to reduce the confusion between clients, servers and issuing CAs over how end entities should or could be described in the **subject** and **subjectAltName** attributes. (RFC 6186) describes email host discovery using SRV and RFC 7817 clarifies RFC6125 for email.) Historically the end entity was defined by the CN= RDN in the **subject** attribute, while this is still supported (for mostly historical

reasons) RFC 6125 prefers the use of **subjectAltName** attribute to provide greater flexibility and clarity. RFC 6125 defines defines four possibilities for end system validation:

Note: In all cases, when using **subjectAltName** more than one name type may be present and that more than one entry in each type may be present. The end entities described by all types present constitute the certificate's end entity coverage. In the case of **subject** only a single end entity described by a single CN= RDN is covered (though in this case a **subjectAltName** attribute may also be present).

1. **subjectAltName** contains a type **dNSName** then the name is the end entity covered by the certificate.

   RFC 7817 indicates that for email the end enity should be either the domain part of the RFC822 address, for example if the email address is user@example.com then the end entity will be example.com, and/or the end entity can be the FQDN of the receiving mail host, for example, mail.example.com.

2. **subject** contains a single **cn=hostname** attribute value (other RDNs may exist but are ignored for domain name validation purposes) where **hostname** is the end entity covered by the certicate.

   RFC 7817 indicates that for email the end entity should be either the domain part of the RFC822 address, for example if the email address is user@example.com then the end entity will be example.com and/or the FQDN of the receiving mail host, for example, mail.example.com.

3. **subjectAltName** contains a type **otherName** attribute and the type of otherName is SRV (oid = 1.3.6.1.5.5.7.8.7) (deifined in RFC 4985)

   RFC 6186 defines the use of SRV RRs in email (and has a - relatively - strange format), RFC 7817 follows this recmmendation.

4. **subjectaltName** contains a type **uniformResourceIdentifier** attribute in which the servive type is explicity identified.

Certificate Revocation Lists (CRLs)

Certificate Revocation Lists (CRLs) are a method by which certificates may be invalidated before their NotAfter date has expired. CRLs are normally issued by the CA that issued the certificate and can be obtained by a variety of methods, for instance, LDAP, HTTP or FTP. CRLs (specifically CRLv2 the current version) are defined in RFC 5280 and updated by RFC 6818.

Theoretically, when a certificate has been received from a server the receiving software should verify that the certificate obtained does not appear in a CRL (has not been revoked). To minimise delay in catching revoked certificates the CRL check should be done by fetching the latest CRL whenever a certificate is received from a server. Since a CRL contains a list of all revoked certificates from any given CA it can be of considerable size thus creating, perhaps, an unacceptable overhead to an TLS/SSL initial connection. Further, depending on the CA's policy, the CRL may be updated every hour or 12 hours or daily etc.. The bottom line being that fetching a CRL file(s), even if it is done for every received certificate, may still yield an out-of-date result.

In practice a per certificate CRL fetch is rarely, if ever, performed and many systems rely on CRL caches that are periodically updated. RFC 5280 uses the weasely term **suitably-recent** to describe the CRL update frequency. An on-line version of the revocation list, known as Online Certificate Status Protocol (OCSP) is defined in RFC 2560, streamlined by RFC 5019 and updated by RFC 6277. A CRL has the following format:

Version	Optional. In general this should indicate version 2 (CRLv2). Since the numbering starts from zero this will have a value of 1. If omitted version 2 (value 1) is assumed.
Signature	Must be the same OID as that defined in SignatureAlgorithm below.
Issuer	A DN of the Authority that signed and therefore issued the CRL. This may or may not be the Certification Authority. See also notes on issuer DN
ThisUpdate	The time and date at which the CRL was created. Date format may be UTCTime (format is YYMMDDHHMMSSZ) or GeneralizedTime (format is YYYYMMDDHHMMSSZ). Where Z is a literal constant indicating offset from Zulu Time (Greenwich Mean Time(GMT)/ UCT) - its absence would imply local time.
NextUpdate	The time and date at which the next CRL will be created. Date format may be UTCTime UTCTime (format is YYMMDDHHMMSSZ) or GeneralizedTime (format is YYYYMMDDHHMMSSZ). Where Z is a literal constant indicating Zulu Time (Greenwich Mean Time) - its absence would imply local time.

RevokedCerticates	The CRL identifies the revoked certificate by its serial number. Serial numbers are not globally unique but are defined to unique within CA therefore any client must use the a combination of the issuer to obtain a unique match.
	`userCertificate CertificateSerialNumber`
	`revocationDate Time`
	`crlEntryExtensions Extensions OPTIONAL`

Extensions	Multiple CRL extensions are defined in RFC 3280 none of which are marked as being CRITICAL and many are only relevant if the CRL is indirect, that is the CRL was issued by a party (an issuer) which was not the issuer of the certificates being revoked.

SignatureAlgorithm	The algorithm used to sign the CRL identified by its OID and any associated parameters. The following illustrates the RSA with SHA1 digital signature OID (sha1WithRSAEncryption): `1.2.840.113549.1.1.5` This value must be the same as that identified in the signature field of the certificate. The currently valid OIDs for use with X.509 certificates are defined in RFC 3279.
SignatureValue	Bit string containing the digital signature for the CRL.

Process and Trust - CA's and X.509 Certificates

As described previously an X.509 certificate is trusted to contain the public key of the entity defined (normally specifically the CN of the subject or subjectAltName). Trust is created during the certificate creation process. The process of obtaining an X.509 certificate will vary in detail from CA to CA but generally consists of the following steps:

1. The first, or topmost, link in the chain of trust is the Certification Authority (CA). CAs are deemed to be trusted organizations either because they say so or because they have passed some standard. In North America WebTrust is the most widely recognized CA standards organization and most CAs carry their seal indicating they have undergone an audit by a WebTrust accredited auditor (increasingly the major International auditing firms are also performing such audits). The very existence of the CA marks the first step in the establishment of the line of trust. The CA has, at some point in time, generated one or more asymmetric keys and, using the private key, has self-signed a certificate (the issuer and subject attributes of the X.509 certificate are the same and BasicConstraints cA is TRUE). This certificate is the root or CA certificate and the private key, whose public key is contained in the root certificate, is used to sign user certificates. Root (or CA) certificates are distributed to end user clients by a trusted out-of-band process, most frequently with browser installations.

2. A user who desires a certificate will review the products from the various Certificate Authorities (CAs) and select a preferred CA. An application is made to the selected CA - or one of its agents or Registration Authorities (RA) - for a particular type of SSL (X.509) certificate. Depending on the certificate type various information is required and (usually) verified such as name of business, business registration number or other identifying information. Again depending on the type of certificate further proof of identity may be required. This is essentially a clerical process (which may be automated to a greater or lesser extent) that establishes the next trust link.

The CA, which is trusted, is trusted to have established that the user is who they say they are. More or less.

3. Having selected the SSL product, supplied the required identification information and had it verified by the CA the user is then requested to generate a set of asymmetric keys and use the private key to sign a Certificate Signing Request (CSR) which will contain the public key of the generated public-private pair among other information. The CSR format is defined by RFC 2986 (a re-publication of the RSA standard PKCS#10 - updated by 5967) which consists of the following data:

Version	Version 0.
Subject	A DN defining the entity to be associated with this certificate. In general, depending on the CA policies, this will be the subject that will appear in the returned X.509 certificate.
SubjectPublicKeyInfo	Contains two sub-fields (attributes), **algorithm** (the OID of the public key algorithm from the list defined in RFC 3279) and **subjectPublicKey** (the entity's public key as a bit string). An example of a RSA algorithm (rsaEncryption) OID is shown below: `1.2.840.113549.1.1.1`
Attributes	Optional. Attributes may contain a number of sub-fields (attributes) of which the following are noted, **challengePassword** (a password to further secure the CSR - most CAs insist this attribute is NOT present) and **unstructuredName** (any suitable text - again normally not required by CAs or ignored if present).
SignatureAlgorithm	The algorithm used to sign the CSR identified by its OID and any associated parameters. The following illustrates the RSA with SHA1 digital signature OID (sha1WithRSAEncryption): `1.2.840.113549.1.1.5` **Notes:** 1. The CSR is signed using the private key of the private-public key pair whose public key appears in the **SubjectPublicKeyInfo** of this CSR. 2. Certificate extensions that will appear in the final certificate can also be present is self-signing is being used. Few commercial CAs support extensions.
SignatureValue	Bit string containing the digital signature.

A CSR is created using this procedure.

4. The CSR is uploaded to the CA (typically using FTP or HTTP) which uses the data in the CSR and perhaps other information obtained during the SSL certificate application to create the user's X.509 certificate which typically has a validity period ranging from 1 to 3 years. The CA finally signs the user's certificate (the SignatureAlgorithm and SignatureValue) using the private key of the public-private key pair whose public key is contained in the CA's root certificate. The X.509 certificate is sent to the user using a variety of processes (FTP/HTTP/EMAIL).

5. The trust loop is therefore completed by the CA's digital signature. The digital signature of the user's certificate can only be verified by using the public key of the issuer which is contained in the CA's root certificate obtained by a (trusted) off-line process (normally via a browser installation).

When a certificate is received by a browser from a web site (during the Handshake Protocol of TLS/SSL) it must be verified all the way to the root or CA certificate (there may be one or more levels of certificates depending on the certifcate vendor). The root/CA (and any required intermediate) certificates are typically distributed with major browser software. Root/CA certificates distributed in this way are generically called trust anchors - a widely used term used to describe any base structure or information obtained by a trusted distribution route that may be used to validate/authenticate received information. RFC 5914 (with some support from RFC 5937) defines how such trust anchors may be organized, used and processed in the specific case of X.509/SSL certificates.

EV Certificates

Extended Validation (EV) Certificates are defined by the CA/Browser Forum and include a mixture of improved clerical validation as well as technical processes. The objective behind EV certificates is to provide increased confidence to end users though the standard explicitly states that it does not guarantee the business practices of the certificate owner - merely that the owner does exist.

When using a supporting browser the user sees a normal padlock icon but the status bar shows up in GREEN when an EV certificate is encountered. Currently EV supporting browsers are: MSIE 7 only and the latest versions of Konqueror, Firefox (v3+) and Opera(9.5+). In general EV certificates are significantly more expensive than other types of certificates. The EV issuing standard has the following characteristics:

1. The CA has to be audited for EV compliance and undergo a yearly audit renewal.

2. Enhanced user verification - notably it demands that the CA verify that the applicant does indeed own the domain name that it is seeking to authenticate!

3. CAs follow the practices defined in RFC 3647 which has INFORMATIONAL status for the rest of us.

4. Standardizes the use and meaning of certain fields in the subject DN and adds some new ones. Specifically it defines the following to be REQUIRED (the EV standard's version of the RFC MUST)

    1. O - organizationalName (OID 2.5.4.10) Full legal name of the user organization
    2. businessCategory (OID 2.5.4.15) defines which category of certificate is involved - may be section 5b, 5c or 5d.
    3. C - Country - (OID 2.5.4.6), ST - stateOrProvinceName - (OID 2.5.4.8) and L - localityName - (OID 2.5.4.7) are all defined to be the legal jurisdiction of the business entity (not, say, the location of a web server).
    4. serialNumber (OID 2.5.4.5) business registration number
    5. CN - CommonName - (OID 2.5.4.3) host domain name, for instance, www.example.com

5. Mandates EV CAs provide on-line capabilities (24x7) to verify the status of any EV certificate. In general this is done using the Online Certificate Status Protocol (OCSP) (RFC 2560).

6. Mandates that EV certificates may only be issued for server authentication at this time, that is, CN= must be a server name such as www.example.com or mail.example.com.

7. EV certificates are recognised definitively by the use of a registered OID (unique for each CA) in the CertificatePolicies field.

Example - Self-Signed Certificates

This section illustrates the use of OpenSSL commands to perform tasks associated with X.509 certificates. To illustrate the certificate process in its entirety it covers generation of what are loosely called **self-signed certificates**. Self-signing is one of those terms that needs a little explanation since it has two potential meanings.

At a strict level it means that the issuer and subject fields in the certificate are the same. In this sense every root certificate is a self-signed certificate. In the second sense it means that the user becomes their own Certification Authority (CA) and is an alternative to purchasing an X.509 certificate from a recognized CA such as Verisign or Thawte (and others) which are already trusted (their root/CA certificates are pre-installed on the user's computer) by most tools, for example, browsers. A user (end-entity) certificate obtained by a browser from a web site during the TLS/SSL Handshake Protocol phase and issued by one of the recognized CAs can be tracked back (and therefore authenticated) by the browser to the signing authority (the CA) by use of the issuer field in the certificate. The browser must have the trusted CA's root certificate installed (as well as any intermediate certificates) to avoid any browser

error messages. The term **trust anchor** is used to defined such installed certificates. Trusted CA root certificates for most of the commercial CAs and many National CAs are distributed and installed with browsers such as MSIE, Firefox, Opera etc.

The rest of this section deals with the second definition of self-signing - the user becomes their own CA. This form of **self-signing** can be used either during testing or operationally where, for instance, a user wishes to control access within a private network, a closed user group or some other community of interest. Here the trust relationship, normally associated with an external CA, may be regarded as being implicit due to the nature of the organization signing the certificates.

The first time a self-signed certificate is presented to a user's TLS/SSL enabled software, such as a browser, and assuming the browser does not have a copy of the relevant root certificate, it will generate a message asking if the user wishes to accept the certificate. Alternatively, the self-signed root certificate can be pre-installed or imported to a user's computer in which case no browser error message will be generated.

Note about Certificate Validity Periods and Key Sizes: Many of the certificates in the following sections are valid for 1 to 3 years. However most of the commercial root certificates installed in browsers have validity periods of 10 to 20 years! Don't be afraid to use quite lengthy time periods to avoid the pain of renewing certificates. The current RSA key length recommendation (2011) is 2048 bits for lifetimes until 2030 (consequentially many of the current root certificates have expiry dates around 2028 giving them a couple of years to phase in extended bit sizes in time for the 2030 expiry of 2048 bits). RSA Key Strength Note and recommendations over required lifetime. The same keysize (2048 bits) and lifetime recommendation (until 2030) is also contained in US NIST (National Institute or Standards and Technology) Special Publication 800-57 Part 1 Rev2 Table 4.

Process Overview

The following sequences use OpenSSL commands (tested with OpenSSL 0.9.8n and 1.0.0e) to generate a self-signed X.509 certificate that may be installed and used by TLS/SSL server systems such as Web, FTP, LDAP or mail (SMTP Agent). OpenSSL commands use a bucket load of parameters - including reply defaults, certificate duration and certificate fields - from the **openssl.cnf** file (/etc/ssl/openssl.cnf - FreeBSD) and if you are going to do serious work with certificates it is well worth looking though this file and editing as appropriate to save some typing.

Note: There are dire text/comment warnings that editing either openssl.cnf or messing around with CA.pl (a useful script file) may cause the sun to fall out of the sky. Make a copy of both files before you do anything so you can always restore your system to a pristine state in the event anything goes wrong. The bottom line: mess with stuff at your own peril.

The files created when working with certificates will be a mixture of those containing public keys which are generally innocuous and those containing private keys which need to be rigorously protected. Whether both these types of files can be maintained in a single directory structure or not will be a matter of local security policy. The actual process depends significantly on the requirement. As with all sophisticated software there are usually about 6,000 ways of doing anything - we think there are just three Really Useful Methods (RUM)™.

FreeBSD Operational Note: When running tests on a new FreeBSD 8.1 install during which **openssl** is installed by default (0.9.8n) CA.pl was not installed. If you have requested installation of source then it may be found in /usr/src/crypto/openssl/apps/CA.pl. Alternatively, use the ports system (/usr/ports/security/openssl) and then **make patch** (just downloads and untars the latest version of openssl but does not install it) and then use **find ./ -name "CA.pl"**. Since PERL is no longer installed by default with FreeBSD you may also need to install it.

Method 1: Quick root and server certificate

Method 2: Quick Single root and server certificate

Method 3: Root CA and Multiple certificates.

Method 3A: Creating Subordinate CAs, Intermediate and Cross certificates.

Method 1: Root Certificate, Server Certificate

Create a CA, a root certificate that can be imported into a browser for testing purposes and a single server certificate that can be used by a server such as Apache. The standard OpenSSL CA.pl script (which has been moved to /etc/ssl for convenience in the examples - choose an appropriate location) is used to simplify the process:

Note: If a certificate suitable for multiple server name use, for instance, example.com and www.example.com or www.example.net is required follow this procedure instead.

```
# by default the RSA algorithm is used with 1024 bit keys (2011)
# to change defaults see Method 3: Bullet 1
# create directories and self-signed CA root certificate
cd /etc/ssl
./CA.pl -newca
# the DN sequence requested here will be for the CA
# by default this creates a root certificate in
# demoCA/cacert.pem
# and the CA's private key in
# demoCA/private/cakey.pem
# cacert.pem is valid for 3 years

./CA.pl -newreq
# creates a Certificate Signing Request (CSR)
# the requested DN sequence will be for the
# server name the only important value is
# CN (CommonName) which should be the web or other server name
# such as www.example.com, ldap.example.com or mail.example.com
```

```
./CA.pl -sign
# signs the CSR
# and leaves the certificate in
# /usr/local/openssl/newcert.pem
# newcert.pem is valid for 1 year
# private key in
# /usr/local/openssl/newkey.pem

# remove the pass phrase from newkey.pem
# to stop server requesting password on load
openssl rsa -in newkey.pem -out keyout.pem

# CAUTION: this file contains a private key and should be
# given read only permissions for the user

# move and rename as required for the server
# both files newcert.pem and keyout.pem are required
# for example for apache
SSLCertificateFile /path/to/newcert.pem
SSLCertificateKeyFile /path/to/keyout.pem
# examples for OpenLDAP
TLSCACertificateFile /path/to/cacert.pem
TLSCertificateFile /path/to/newcert.pem
TLSCertificateKeyFile /path/to/keyout.pem
```

Further explanation and notes. The file demoCA/cacert.pem, which is the root certificate, can be copied and imported to a browser.

Method 2: Single Server Certificate

This is the simplest (one command) way to create a single X.509 certificate that may be used both as a server and a root (CA) certificate. This certificate, because it is self-signed (and the CA:True field is present), can be imported into browsers and other client software as a root (CA) certificate. The DN requested in this sequence references the server that this certificate is being generated for. In particular the CN should define the host service, such as, www.example.com and not the host name of the computer. Thus, if a web service is addressed externally as https://www.example.com but runs on a host called webserver.example.com, CN=www.example.com should be used. The following shows the standard OpenSSL dialog, values in #description# should be replaced with the required value, for example, #server-name# should be replaced with the valid server name being authenticated, say, www.exmple.com or ldap.example.com.

Note: If a certificate suitable for multiple server name use, for example example.com and www.example.com or www.example.net is required follow this procedure instead.

```
cd /etc/ssl
openssl req -x509 -nodes -days 1059 -newkey rsa:2048 \
 -keyout testkey.pem -out testcert.pem

Generating a 2048 bit RSA private key
.....++++++
..................++++++
writing new private key to 'testkey.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) [AU]:#your-country-code#
State or Province Name (full name) [Some-State]:#your-state-county-province#
Locality Name (eg, city) []:#your-city#
Organization Name (eg, company) [Internet Widgets Pty Ltd]:#your-organization-
name#
Organizational Unit Name (eg, section) []:#optional#
Common Name (eg, YOUR name) []:#server-name#
Email Address []:.

# creates testcert.pem as certificate
# and testkey.pem as unencrypted private key
# which should immediately be given
# read only access to user
# chown user:group testkey.pem
# chmod 0400 testkey.pem

# move and rename as required for the server
# both files testcert.pem and testkey.pem are required
# e.g. for apache
SSLCertificateFile /path/to/testcert.pem
SSLCertificateKeyFile /path/to/testkey.pem
# for OpenLDAP TLS Server (slapd.conf)
TLSCertificateFile /path/to/testcert.pem
TLSCertificateKeyFile /path/to/testkey.pem
# for OpenLDAP TLS Client (ldap.conf)
TLS_CACERT /path/to/testcert.pem
```

Notes: -x509 causes it to self sign as a root CA. **-nodes** suppresses the pass phrase dialog. **-days 1059** provides a 2 year 329 day certificate.

<ouch> Readers may wonder why any sane person would want to create a certificate valid for 2 years and 329 days. There are two reasons. First, it's possible, so why not. Second, observant readers will have noted that 3 years is 1095 days not the 1059 used in the tests. In short, when running the intial tests we incorrectly transposed the 5 and 9 (put it down to age, eyesight or stupidity as you choose) and rather than rerun the tests with 1095 we let the 1059 stand and have come up with this crummy justification instead. Finally, since the current recommendation for 2048 bit keys suggest they will remain valid until 2030 you could sensibly push the value to (currently, in 2013) **-days 6205** (17 years, minus leap year adjustments).**</ouch>**

Method 3: Root CA and Multiple Certificates

If you are going to generate multiple certificates for use, say, with an internal system then it is worth investing some time and effort. Having tried this using multiple methods this is, IOHO, the simplest. It uses the standard CA.pl script to establish the CA which initialises a bunch of directories and files that are otherwise a serious pain to set up but then uses openssl commands to generate CSRs and sign certificates since this provides greater control over variables and is, relatively, painless.

1. **Location and Preparation** Optional to save some typing. Decide where you are going to build your certificate repository. For the sake

of illustration we will create it in /etc/ssl and that location will be used throughout. Change as required to suit your circumstances. Move the CA.pl script (use *locate CA.pl* if you can't find it) to /etc/ssl/CA.pl or wherever you are going to create the certificate repository since we will edit this file. We also rename the script to ca.pl due to a perverse loathing of the shift key but assuming you don't share our pathological characteristics you can substitute CA.pl in the examples.

The files ca.pl (or CA.pl) and /etc/ssl/openssl.cnf have significant impact on the running of both the script and the openssl commands. We made the following edits for convenience to CA.pl (ca.pl) only the changed lines are shown:

```
# WARNING: You should make and keep a clean copy of both
#          CA.pl and openssl.cnf before messing with either
# root certificate valid for 10 years - matter of choice
# but most public CAs provide a root certificate valid until 2028
$CADAYS="-days 3650";   # 10 years
# default is $CADAYS="-days 1059";      # 3 years

# changes ca directory name
$CATOP="./ca";
# default is $CATOP="./demoCA";
# if changed needs corresponding change in openssl.cnf
```

We also made the following changes in /etc/ssl/openssl.cnf to make life easier. Again, only the changed lines are shown:

```
# WARNING: You should make and keep a clean copy of both
#          CA.pl and openssl.cnf before messing with either
[ CA_default ]
# mirrors directory change in ca.pl
dir = ./ca                  # Where everything is kept
# dir = ./demoCA   # Where everything is kept (default)

# Extension copying option: use with caution
copy_extensions = copy
# uncomment above directive if multiple DNS name certificates
# are required, else leave commented (default)

# default certificate validity changed to 3 years
# means you can omit the -days option
default_days    = 1059 # 3 years
# default is default_days       = 365 # 1 year

[ policy_match ]
# add this line in the above section
# adds L= to issuer and subject DN
# otherwise it is omitted - matter of taste
localityName            = optional

[req]
default_bits = 2048 # current 2011 - 2030 recommendation
# default_bits = 1024 # original file value

[ req_distinguished_name ]
# change _default values to save typing
# edit or add in appropriate place
countryName_default             = MY
stateOrProvinceName_default     = STATE
localityName_default            = CITY
0.organizationName_default      = ONE INC
```

```
organizationalUnitName_default  = IT
```

It is worthwhile checking this file for any other values that you might want to change.

2. **Create Certificate Authority** The first command creates a Certification Authority (CA) root certificate and some other files used for maintenance. A public-private key pair is created. The public key is written into the root certificate - /etc/ssl/ca/cacert.pem - the private key into /etc/ssl/ca/private/cakey.pem. The default file format is PEM. The DN details requested will be used to populate the issuer and subject fields of the root certificate and into the issuer field of all subsequent signed certificates and should be customized to suit user requirements. The pass phrase is required and used to protect access to the private key - it will be used on all subsequent certificate signings so it might be useful to remember it:

```
cd /etc/ssl
./ca.pl -newca

CA certificate filename (or enter to create) #ENTER

Making CA certificate ...
Generating a 2048 bit RSA private key
.....++++++
..................++++++
writing new private key to './ca/private/cakey.pem'
Enter PEM pass phrase:
Verifying - 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.
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) [MY]:
State or Province Name (full name) [STATE]:
Locality Name (eg, city) [CITY]:
Organization Name (eg, company) [ONE INC]:CA COMPANY NAME
Organizational Unit Name (eg, section) [IT]:X.509
Common Name (eg, YOUR name) []:CA ROOT
Email Address []:.

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from /etc/ssl/openssl.cnf
Enter pass phrase for ./ca/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
  Serial Number:
     bb:7c:54:9b:75:7b:28:9c
  Validity
     Not Before: Apr 15 21:07:36 2008 GMT
     Not After : Apr 13 21:07:36 2018 GMT
  Subject:
     countryName               = MY
```

```
     stateOrProvinceName       = STATE
     organizationName          = CA COMPANY NAME
     localityName              = CITY
     organizationalUnitName    = X.509
     commonName                = CA ROOT
  X509v3 extensions:
   X509v3 Subject Key Identifier:
    54:0D:DE:E3:37:23:FF...
   X509v3 Authority Key Identifier:
    keyid:54:0D:DE:E3:37...
    DirName:/C=MY/ST=STATE/O=CA COMPANY NAME/L=CITY/OU=X.509/CN=CA
ROOT
    serial:BB:7C:54:9B:75:7B:28:9C
   X509v3 Basic Constraints:
    CA:TRUE
Certificate is to be certified until Apr 13 21:07:36 2018 GMT (3650
days)

Write out database with 1 new entries
Data Base Updated
```

Note Lines may have been truncated and omitted (replaced with ... in both cases) above since the deleted material does nothing to aid understanding of the process.

A standard directory structure has been created:

```
ca                  # cacert.pem (root certificate)
                    # serial (tracks serial numbers)
                    # crlnumber (serial number for CRLs)
                    # index.txt
ca/private/cakey.pem # private ca key
ca/newcerts         # copy of all certs created
ca/crl              # optional location for CRLs created
ca/certs            # optional location for certs created
```

The root certificate created (ca/cacert.pem) is valid for 10 years (3650 days) based on the ca.pl file edit. The root private key (ca/private/cakey.pem is protected by its PEM pass phrase but should still be protected by the lowest permissions possible (defaulted to 0644 which is unnecessarily high). The Challenge Password is a simple password that may be used to guard access to Certificates, CSRs and the subsequent X.509 certificate. Most, if not all, commercial CAs do not want one or the accompanying Optional Company Name and all the examples leave the field blank. If you want to permanently disable the prompts edit /etc/ssl/openssl.cnf:

```
[ req_attributes ]
# comment out all following lines
#challengePassword              = A challenge password
#challengePassword_min          = 4
#challengePassword_max          = 20

#unstructuredName               = An optional company name
```

3. **Create a certificate signing request (CSR)**. The requested DN in this sequence will appear in the subject field of the final certificate.

   This command may also be used to generate a CSR (Certificate Signing Request) for a commercial CA and since most CAs do not

want a password just hit ENTER at the **A challenge password** prompt (see notes above about permanently removing the prompts). The optional emailAddress value is also left blank. Mostly because it achieves no purpose in a server certificate and indeed most commercial CAs request it is left blank.

```
openssl req -nodes -new -newkey rsa:2048 \
-keyout ca/private/cert1key.pem -out ca/certs/cert1csr.pem

Generating a 2048 bit RSA private key
..................++++++
............++++++
writing new private key to 'ca/private/cert1key.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) [MY]:
State or Province Name (full name) [STATE]:
Locality Name (eg, city) [CITY]:
Organization Name (eg, company) [ONE INC]:
Organizational Unit Name (eg, section) [IT]:
Common Name (eg, YOUR name) []:www.example.com
Email Address []:.

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

This creates a new CSR (in /etc/ssl/ca/certs/cert1csr.pem) which must then be signed using the CA's private key and a private key (not secured by a pass phrase - **-nodes** option - suitable for use in a server application) in /etc/ssl/ca/private/cert1key.pem. The CN value (www.example.com in the example above) is the service name used by the application. It could have been ldap.example.com or any other similar name. Note also that if the normal server access is https://www.example.com this certificate will work - however if the access will also use https://example.com then you need to force use of a **subjectAltName** certificate attribute by using this procedure before creating the CSR (and use **ServerAlias** directive in the VirtualHost section for Apache). The service name should not be confused with the server's host name. Thus, if a web service is addressed externally as https://www.example.com but runs on a host called webserver.example.com, www.example.com should be used for the CN.

To view the certificate request:

```
openssl req -in ca/certs/cert1csr.pem - noout -text

Certificate Request:
 Data:
  Version: 0 (0x0)
```

```
  Subject: C=MY, ST=STATE, L=CITY, O=ONE INC, OU=IT,
CN=www.example.com
  Subject Public Key Info:
  Public Key Algorithm: rsaEncryption
  RSA Public Key: (1024 bit)
   Modulus (1024 bit):
    00:ae:19:86:44:3c:dd...
    ...
    99:20:b8:f7:c0:9c:e8...
    38:c8:52:97:cc:76:c9...
   Exponent: 65537 (0x10001)
   Attributes:
     a0:00
 Signature Algorithm: sha1WithRSAEncryption
  79:f5:20:45:6c:ec:8e:ae...
  ...
  bd:61:cd:c5:89:7c:e0:0d...
  40:7d
```

Note Lines have been truncated and omitted (replaced with ... in both cases) above since the deleted material does nothing to aid understanding of the process.

4. **Create and Sign the end-user certificate** This takes the input CSR (ca/certs/cert1csr.pem) and creates the end user (end-entity) certificate (ca/certs/cert1.pem):

```
openssl ca -policy policy_anything \
-in ca/certs/cert1csr.pem -out ca/certs/cert1.pem

Using configuration from /usr/local/openssl/openssl.cnf
Enter pass phrase for ./ca/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
  Serial Number:
    bb:7c:54:9b:75:7b:28:9d
  Validity
    Not Before: Apr 15 22:21:10 2008 GMT
    Not After : Mar 10 22:21:10 2011 GMT
  Subject:
    countryName               = MY
    stateOrProvinceName       = STATE
    localityName              = CITY
    organizationName          = ONE INC
    organizationalUnitName    = IT
    commonName                = www.example.com
 X509v3 extensions:
  X509v3 Basic Constraints:
   CA:FALSE
  Netscape Comment:
   OpenSSL Generated Certificate
  X509v3 Subject Key Identifier:
   EE:D9:4A:74:03:AC:FB:2C...
  X509v3 Authority Key Identifier:
   keyid:54:0D:DE:E3:37:23...

Certificate is to be certified until Mar 10 22:21:10 2011 GMT (1059
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
```

Note Lines have been truncated and omitted (replaced with ... in all cases) above since the deleted material does nothing to aid understanding of the process.

The resulting certificate issuer field is from the root certificate (ca/cacert.pem) and the subject field from the CSR. The command line defaults many values from openssl.cnf: the validity period (default_days =), the signing key (private_key =) and the issuer from the root certificate (certificate =). The **-policy policy_anything** option may be necessary depending on your requirements. It references the **policy_anything** section defined in the openssl.cnf file which allows any fields in the subject DN certificate to have different values from those in the issuer DN field. If not present it defaults to **-policy policy_match** which places restrictions on C=, ST= and O= RDN values.

To view the resulting certificate:

```
openssl x509 -in ca/certs/cert1.pem -noout -text

Certificate:
 Data:
  Version: 3 (0x2)
  Serial Number:
   bb:7c:54:9b:75:7b:28:9d
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: C=MY, ST=STATE, O=CA COMPANY NAME, L=CITY, OU=X.509, CN=CA
ROOT
  Validity
   Not Before: Apr 15 22:21:10 2008 GMT
   Not After : Mar 10 22:21:10 2011 GMT
  Subject: C=MY, ST=STATE, L=CITY, O=ONE INC, OU=IT,
CN=www.example.com
  Subject Public Key Info:
   Public Key Algorithm: rsaEncryption
   RSA Public Key: (1024 bit)
    Modulus (1024 bit):
     00:ae:19:86:44:3c:dd...
     ...
     99:20:b8:f7:c0:9c:e8...
     38:c8:52:97:cc:76:c9...
    Exponent: 65537 (0x10001)
 X509v3 extensions:
  X509v3 Basic Constraints:
   CA:FALSE
  Netscape Comment:
   OpenSSL Generated Certificate
  X509v3 Subject Key Identifier:
   EE:D9:4A:74:03:AC:FB:2C...
  X509v3 Authority Key Identifier:
   keyid:54:0D:DE:E3:37:23...

 Signature Algorithm: sha1WithRSAEncryption
  52:3d:bc:bd:3f:50:92...
  ...
  51:35:49:8d:c3:9a:bb...
  b8:74
```

Note Lines have been truncated and omitted (replaced with ... in both cases and which never appear legitimately in these fields) in the above since the deleted material does nothing to aid understanding of the process.

Useful Commands

Checking or viewing Certificates Corruption can occur so the certificate can be verified using the following commands:

```
openssl x509 -in certificate-name.pem -noout -text
# display whole certificate

openssl x509 -in certificate-name.pem -noout -dates
# validity dates only

openssl x509 -in certificate-name.pem -noout -purpose
# list of all purposes certificate may be used for

openssl x509 -in certificate-name.pem -noout -purpose - dates
# list validity period and purpose
```

5. **Importing a self-signed root certificate** The PEM format file created in step 2 above (ca/cacert.pem) may be directly imported into MSIE and Firefox to inhibit them from prompting for untrusted certificates. Follow procedure.

Method 3A: Creating Subordinate CAs, Intermediate and Cross Certificates

Method 3 creates a simple two certificate chain comprising an end-entity (server) certificate and a root CA certificate. Method 3A explores how we can add to this structure, for reasons best known to ourselves, to create a subordinate CA, perhaps a second root CA and create a whole variety of Cross and Intermediate certificates based on this structure:

1. **Base Setup:** Run steps 1 and 2 of Method 3. This simply creates a directory structure and our first root CA. If you have already run the whole of Method 3, none of the file names created in the next steps will clash - so nothing will be lost and your system will not self-destruct. At the end of this process the following directories will have been created:

```
ca                    # cacert.pem (root certificate)
                      # serial (tracks serial numbers)
                      # crlnumber (serial number for CRLs)
                      # index.txt
ca/private/cakey.pem  # private ca key
ca/newcerts           # copy of all certs created
ca/crl                # optional location for CRLs created
ca/certs              # optional location for certs created
```

Strategy Note: Openssl commands take many of their advanced parameters from the configuration file (openssl.cnf). Up to this point we have suggested that changes should be made to this standard file since they are typically used consistently accross all commands. From this point on this is not the case. Instead, we recommend that you copy and rename the standard configuration file (openssl.cnf) to a new configuration file with an appropriate name (that you can remember) and use the **-config filename** argument to pick up the

appropriate config file based on usage. It's quicker (eventually), less confusing (eventually) and re-useable (always).

2. **Create a subordinate CA:**

   Create a new directory called, say, subca and subca/private (giving this directory the same permissions as ca/private). This step simply keeps things organized (kinda):

```
cd /etc/ssl
mkdir ca/subca
mkdir ca/subca/private
# create necessary empty database file for subca signer
touch ca/subca/index.txt
```

   Edit openssl.cnf and create a section called [sub_ca] - it can be anywhere in the file but for convenience we have shown it located above the [user_certs] section. The various entries in this section will be used to override the normal options used when a certificate is signed.

```
# new section
[ sub_ca ]
basicConstraints=CA:TRUE
# alternatively
# basicConstraints= CA:TRUE,pathlen:0
# pathlen:0 constrains the subCA to
# only sign end-entity certificates

# optional entry and string could be
# nsComment="Most Trusted Certificate in the World"
nsComment="OpenSSL Generated Certificate"
# next two are normal for all non-root certificates
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# next line just shows where sub_ca section is positioned in file
[ user_cert ]
```

3. **Create the subordinate CA certificate:**

   Create a CSR for the subordinate CA and sign it with the root CA certificate:

```
# NOTE: These steps will use the root CA created in Method 3 step 1
and 2

# create the CSR for the subordinate CA
openssl req -new -keyout ca/subca/private/subca1key.pem \
  -out ca/subca/subca1csr.pem
Generating a 2048 bit RSA private key
..................++++++
............++++++
writing new private key to 'ca/subca/private/subca1key.pem'
Enter PEM pass phrase:
Verifying - 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.
```

```
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) [MY]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) [Some City]:
Organization Name (eg, company) [My Company Name]:
Organizational Unit Name (eg, section) []:Certs
Common Name (eg, YOUR name) []:subCA1

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

# Notes:
# 1. Organizational Unit Name is not important simply descriptive
# 2. Common Name is the name to be given to the subordinate CA

# now sign this request using the root CA and force the defined
extensions
# using -extensions sub_ca
openssl ca -policy policy_anything -in ca/subca/subca1csr.pem \
  -out ca/subca/subca1cert.pem -extensions sub_ca

# display the resulting certificate
openssl x509 -in ca/subca/subca1cert.pem -noout -text
Certificate:
 Data:
  Version: 3 (0x2)
  Serial Number:
     c6:bd:b2:ce:22:bc:4d:57
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: C=MY, ST=Some-State, O=My company name, OU=Certs, CN=Root
CA1
  Validity
   Not Before: Dec 9 20:40:18 2011 GMT
   Not After : Dec 6 20:40:18 2021 GMT
  Subject: C=MY, ST=Some-State, L=Some City, O=My company name,
OU=Certs, CN=subCA1
  Subject Public Key Info:
   Public Key Algorithm: rsaEncryption
   RSA Public Key: (2048 bit)
    Modulus (2048 bit):
     00:a9:f3:02:01:c9...
     01:b6:27:c8:a0:9c...
     ...
     f0:37:71:5d:e3:c7:3d:59:ff...
     55:87
    Exponent: 65537 (0x10001)
   X509v3 extensions:
    X509v3 Basic Constraints:
     CA:TRUE
    X509v3 Key Usage:
     Certificate Sign, CRL Sign
    Netscape Comment:
     OpenSSL Generated Certificate
    X509v3 Subject Key Identifier:
     58:47:30:77:3F:EF...
    X509v3 Authority Key Identifier:
     keyid:FB:7B:FB:7B...

 Signature Algorithm: sha1WithRSAEncryption
  43:b5:e2:8d:4d:07:56...
  ...
  12:2c:a2:7c:eb:dc:45...
  e0:f3:2b:72
```

Note Lines have been truncated and omitted (replaced with ... in both cases) in the above since the deleted material does nothing to aid understanding of the process.

The four **X509v3 extensions** are the direct consequence of the **-extensions sub_ca** argument and override the normal certificate features. If the root CA is required to sign a normal end-entity certificate just omit this argument. Technically, the resulting certificate (ca/subca/subca1cert.pem) is a **cross-certificate** because both the signer and the certificate being signed have **basicConstraints** with **cA True**.

4. **Sign an end-entity certificate with the subordinate CA:**

To sign an end-entity certificate using the subordinate CA we first need to copy and save openssl.cnf to another name, say, subca1.cnf. Now make the following edits to subca1.cnf (these changes simply tell openssl where to obtain various files relating to the subCA whereas the standard openssl.cnf will continue to refer to the root CA information):

```
[ CA_default ]
# only values to be changed are shown
database        = $dir/subca/index.txt  # database index file.
certificate     = $dir/subca/subca1cert.pem     # The subCA
certificate
# the parameter below will cause all certicates from both the root
# and the sub CA to use the same numbering sequence
# to change copy ca/serial to ca/subca/serial and modify parameter
serial          = $dir/serial           # The current serial number
# if you need to keep CRLs separate or leave unchanged
crl_dir         = $dir/subca/crl                # Where the issued crl
are kept
crlnumber       = $dir/subca/crlnumber  # the current crl number

# must be commented out to leave a V1 CRL
crl             = $dir/subca/crl.pem            # The current CRL

private_key=$dir/subca/private/subca1key.pem # The subCA private key
RANDFILE        = $dir/subca/private/.rand      # private random
number file
```

Now create an end-entity CSR and sign it with the subCA keys:

```
# create CSR
openssl req -new -nodes -keyout ca/private/user1key.pem \
  -out ca/certs/user1csr.pem
# sign it with the subCA key by using -config subca1.cnf
openssl ca -policy policy_anything -in ca/certs/user1csr.pem \
  -out ca/certs/user1cert.pem -config subca1.cnf
# print the resulting certificate
openssl x509 -in ca/certs/user1cert.pem -noout -text
Certificate:
 Data:
  Version: 3 (0x2)
  Serial Number:
   c6:bd:b2:ce:22:bc:4d:58
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: C=MY, ST=Some-State, L=Some City, O=My company name,
OU=Certs, CN=subCA1
  Validity
    Not Before: Dec 9 21:06:43 2011 GMT
```

```
    Not After : Dec 6 21:06:43 2021 GMT
  Subject: C=MY, ST=Some-State, L=Some City, O=My company name,
OU=Server, CN=www.example.com
  Subject Public Key Info:
  Public Key Algorithm: rsaEncryption
   RSA Public Key: (2048 bit)
    Modulus (2048 bit):
     00:c3:f4:dc:07:08:30:3a...
     ...
     a8:45:fd:c5:d7:a4:04:82...
     af:dd
    Exponent: 65537 (0x10001)
  X509v3 extensions:
   X509v3 Basic Constraints:
    CA:FALSE
   Netscape Comment:
    OpenSSL Generated Certificate
   X509v3 Subject Key Identifier:
    B1:5A:23:4E:C8:2B:FD:98...
   X509v3 Authority Key Identifier:
    keyid:58:47:30:77:3F:EF...

 Signature Algorithm: sha1WithRSAEncryption
  50:4b:8e:50:8f:fa:f4:98...
  ...
  3d:97:52:28:1f:a6:9d:2e...
  ac:58:be:eb
```

Note Lines have been truncated and omitted (replaced with ... in both cases) in the above since the deleted material does nothing to aid understanding of the process.

The certificate issuer in this case is **subCA1** not **root CA1** as when signing the subordinate CA certificate request. To allow chain validation of the end-entity certificate BOTH the root CA certificate (ca/cacert.pem) and the subordinate CA certificate (ca/subca/subca1cert.pem) must be imported into the browser.

5. **Other Possibilities:**

   Using variations of this technique, a second root CA (root CA2) could be created (another copy of openssl.cnf would be needed clearly), or a second subordinate CA (subCA2 - again with a unique .cnf file). In short, all sorts of ghastly permutations are possible.

Multi-Server Certificates

If a certificate may be used by a host which has more than one DNS name, say, https://www.example.com and https://example.com or even www.example.net then you need to force the Certificate Signing Request (CSR) to use the **subjectAltName** certificate attribute. To do this, edit the openssl.cnf file as shown below (only changed lines shown):

```
# first find the [CA_Default] section
[CA_Default]
....
# uncomment or add
copy_extensions = copy
# this causes the ca function to copy the extension fields
# from the CSR and should be done on the host which handles
# the CSRs to create the certificates
```

```
# now find [v3_req] section
[v3_req]
...
# add the following line with host names modified as required
subjectAltName = "DNS:www.example.com, DNS:example.com,DNS:example.net"
# this should be done on the host that generates the CSR
```

When you run any CSR request add **-reqexts "v3_req"** to the arguments only when you need to include the additional **subjectAltName** fields. While the example above shows 3 names being added in practice it can be any number. Each entry has the format DNS:hostname.domain.name, multiple entries are comma separated and the whole lot is enclosed in quotes. The line following shows the CSR request in Step 3 of Method 3 with the additional argument:

```
openssl req -nodes -new -newkey rsa:2048 \
-keyout ca/private/cert1key.pem -out ca/certs/cert1csr.pem -reqexts "v3_req"
```

If the **-reqexts** argument is not added to the CSR a normal single server name certificate is created. This process only works with method 3 above and provides the most flexible approach.

Note: If you are going to produce a number of certificates each of which has different multiple server names then a better strategy may be to simply copy and rename the standard configuration file (openssl.cnf) to something sensible and then use the **-config filename** argument to pick up each file when required. You can have any number of such config files - as long as you remember their names and functionality.

If you want to create multi-server certificates with Method 1 or 2 then in addition to the above edits, make a further edit to openssl.cnf:

```
# find the [req] section
[req]
....
# add or uncomment
req_extensions = v3_req
```

This modification has the effect of adding the **subjectAltName** certificate attribute unconditionally to every CSR.

SSL Related File Format Notes

There are a confusing number of file formats with sometimes (in)appropriate file suffixes used for certificates, keys and other data used within X.509/SSL. This is an overview that may help before you dive into the quagmire:

1. All SSL related objects (Certificates, keys etc.) use native DER encoding. DER is a binary (8 bit) encoding which means that it cannot be sent by, among others, email. PEM (Privacy Enhanced Mail) is a method of simply encoding the native DER formats, using

base64, into something that can be sent by email or other communications systems.

2. Some PKCS#X standards are containers (encoded in DER format) - notably PKCS#7 (and its IETF CMS equivalent RFC 5652), PKCS12 and PKCS#8 - used to identify multiple objects within the same file. If a file contains a single object, such as a certificate or a CRL, then this object does not require - though it, optionally, may be, enclosed in - a container.

3. On its face, a single key looks as if it should be a single object and thus not require a container. However, the key use algorithm (as well as other parameters) is also needed and hence a single key does require a container (in this case PPKCS#8) to identify its constituent objects. Conversely, an X.509v3 certificate contains lots of information and therefore looks like it should need a container. However, X.509v3 is itself a container. Thus, where a single certificate appears in a file it does not need a container (for instance, .cer/.crt files). However, when a certificate and a private key both appear in a single file (.p12/.pkx) then in this case there are at least two objects which must be identified and therefore there must be a container (in this case PKCS#12).

4. Many of the PKCS#X standards are containers (encoded in DER) that will implicitly be send by a communications system of some sort, for example, a CSR, and thus are always, irrespective, of the file suffix finally encoded as PEM.

5. In many cases it is context that will determine the content of the file not the suffix. Thus, if you are expecting a single certificate (without a private key) then it may reasonably have the suffix .pem or .crt or .cer.

6. As a simple test open any file, irrespective of its file suffix, in a text editor. If its gobbledegook it's DER encoded. If you can see '-----BEGIN' it's PEM.

PEM Format

OpenSSL supports Privacy Enhanced Mail (PEM) as its default (native) format. An X.509 certificate's intrinsic format is ASN.1 DER (a binary format) as indeed are all other SSL related objects. PEM encodes binary DER in base 64 (RFC 3548) creating a text (ASCII/IA5 subset) version that may be sent by, among other things, mail systems. Objects encoded by PEM include header lines and trailer lines each starting and finishing with precisely 5 dashes to encapsulate the base64 material and provide a human readable indication of its content. PEM files look something like that shown below:

```
-----BEGIN CERTIFICATE-----
MIIDHDCCAoWgAwIBAgIJALt8VJ...
...
Cfh/ea7F1El1Ym1Zj2v3wLhRl1...
NH5lEmZybl+m2frlkjUv9KAvxc...
```

```
IFgovdU8YPMDds=
-----END CERTIFICATE-----

BLAH BLAH BLAH

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,6EF6203EF1A9533A

r7LMq15wr1OOmMsD84KyNo+5yY...
El3/msvQ98BkaMihajEn5f2UxO...
...
f6uoSk8HBZLItWTxqRuBRVb8jq...
hdp9hvvdja9XIrAPGQJ0u2QVw==
-----END RSA PRIVATE KEY-----
```

Note Lines have been truncated and omitted (replaced with ... in both cases) above since the deleted material does nothing to aid understanding of the process.

The text BEGIN CERTIFICATE and END CERTIFICATE in the above may take different values that describe the functionality and format of the material contained within. There may be more than one item in any PEM file (each de-limited by the -----BEGIN -----END sequence) and the file may have other information before, after and between, but not within the de-limiters. PEM is defined in RFC 1421 for use by S/MIME (RFC 3850).

PEM BEGIN Keywords

RFC 7468 identifies a number of keywords (or labels) that may appear in PEM encoded files in -----BEGIN and -----END markers (but, perhaps surprisingly, does not establish a IANA repository for these keywords). In addition the header file pem.h from the Openssl package (version 1.0.2d) contains other keywords (some of which are explicitly deprecated by RFC 7468, some of which may be implicitly deprecated - see notes below). The following table contains an amalgam of the two sources with appropropriate decriptions and notes added:

Note: The Keywords must appear in both the BEGIN and END labels in a PEM encoded file. For brevity neither the -----BEGIN or ----END string is shown. Thus, in the table the keyword CERTIFICATE appears. In a PEM encoded file this would appear twice as -----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----.

Keyword	Source	Usage	Notes
CERTIFICATE	RFC 7468	Contains single DER encoded X.509v3 certificate as defined by RFC 5280 Section 4	RFC 7468 deprecates X509 CERTIFICATE and X.509 CERTIFICATE
X509 CRL	RFC 7468	Contains single DER encoded X.509 certificate list as	RFC 7468 deprecates CRL

		defined by RFC 5280 Section 5	
CERTIFICATE REQUEST	RFC 7468	Contains single DER encoded PKCS#10 certificate request (aka CSR) as defined by RFC 2986 updated by RFC 5967	RFC 7468 deprecates NEW CERTIFICATE REQUEST
PKCS7	RFC 7468	Contains DER encoded PKCS#7 (CMS) container (which may include multiple certificates) as defined by RFC 2315	RFC 7468 deprecates CERTIFICATE CHAIN and implicitly discourages the use of multiple certificates in a PKCS#7 structure (RFC 7468 encourages replacement of PKCS#7 by IETF CMS RFC 5652 - see below).
CMS	RFC 7468	Contains DER encoded CMS container (which may include multiple certificates) as defined by RFC 5652	Mostly backward compatible with PKCS#7 above
PRIVATE KEY	RFC 7468	Contains unencrypted DER encoded PKCS#8 container with a single key as defined by RFC 5958 Section 2	RFC 5958 renames PrivateKeyInfo (from RFC 5208) to OneAsymmetricKey which allows for both public and private keys within the container. However, RFC 7468 does NOT support this format for PEM encoded public keys (see PUBLIC KEY below). Since the PKCS#8 container identifies the key algorithm it implicity deprecates (but does not do so explicity) RSA PRIVATE KEY, DSA PRIVATE KEY, EC PRIVATE KEY or ANY PRIVATE KEY, DSA PARAMETERS, EC PARAMETERS, DH PARAMETERS all of which may be generated by OpenSSL.
ENCRYPTED PRIVATE KEY	RFC 7468	Contains an encrypted DER encoded PKCS#8 container with a single key as defined by RFC 5958 Section 3	
PUBLIC KEY	RFC 7468	Contains an DER encoded SubjectPublicKeyInfo structure (a mini-container) with a single public key as	The SubjectPublicKeyInfo structure describes the key algorithm and therefore RFC 7468 implicitly deprecates (but does not do so explicitly) DSA PUBLIC KEY, RSA

		defined by RFC 5280 Section 4.1.2.7	PUBLIC KEY and ECSDA PUBLIC KEY all of which can be generated by OpenSSL.
ATTRIBUTE CERTIFICATE	RFC 7468	Contains an DER encoded Attribute certificate as defined by RFC 5755	Attribute Certificates are DER encoded X.509 which do NOT contain public keys. They are used primarily for authorization (not authentication) purposes. OpenSSL (1.0.2d) has no valid PEM keyword in pem.h for this certificate type.
CERTIFICATE PAIR	pem.h	??	Defined in OpenSSL pem.h header file but not addressed by RFC 7468. Content target currently unknown.
TRUSTED CERTIFICATE	pem.h	??	Defined in OpenSSL pem.h header file but not addressed by RFC 7468. Content target currently unknown but suspected to be a certificate with basicConstraints, cA TRUE..
PKCS #7 SIGNED DATA	pem.h	??	Defined in OpenSSL pem.h header file but not addressed by RFC 7468. PKCS#7 allows a number of 'content types', one of which is signed data - this feature is not widely implemented.
SSL SESSION PARAMETERS	pem.h	??	Defined in OpenSSL pem.h header file but not addressed by RFC 7468. Content target currently unknown.
X9.42 DH PARAMETERS	pem.h	??	Defined in OpenSSL pem.h header file but not addressed by RFC 7468. Content target currently unknown.

File Names

In many cases the file suffix is relatively meaningless as a means of identifying the content. If you know the context (what the file should contain) then the following information may help:

Use	Suffix	Private Key	Format	Notes
Key	.pem	yes	PEM	PEM encoded DER key in PKCS#8/RFC 5958 container, may be encrypted or unencrypted (PEM keyword will differentiate), though conventionally unencrypted due to its normal usage. PEM label PRIVATE KEY or ENCRYPTED PRIVATE KEY.
	.der	yes	DER	Not widely used. DER key in PKCS#8/RFC 5958 container, may be encrypted or unencrypted, though conventionally unencrypted due to its normal usage.

	.key	yes	PEM	Suffix used on many *nix systems to identify private key. DER key in PKCS#8/RFC 5958 container, may be encrypted or unencrypted, though conventionally unencrypted due to its normal usage. PEM label PRIVATE KEY or ENCRYPTED PRIVATE KEY.
Certificate	.crt	No	PEM or DER	Normally in PEM format (RFC 7468 suggest suffix always denotes PEM). Contains an X.509v3 certificate only. No container. Format accepted by MSIE, Firefox and Chrome browsers.
	.cer	No	PEM or DER	Normally in DER format (RFC 7468 suggest suffix always denotes DER). Contains an X.509v3 certificate only. No container. Format accepted by MSIE, Firefox and Chrome browsers.
	.pem	May	PEM	The suffix is not an indicator of the file content. May contain almost anything, from a single key to multiple certificates in a ca-bundle, encapsulated in a PKCS#7 (CMS) or even PKCS#10 certificate request. PEM label will describe contents. In some contexts this file is assumed to be the exact equivalent of .crt. Firefox uniquely accepts this format.
	.der	May	DER	The suffix is not an indicator of the file content. May contain almost anything, from a single key to multiple certificates in a ca-bundle, encapsulated in a PKCS#7 (CMS) or even PKCS#12 container or not. In some contexts this file is assumed to be the exact equivalent of .cer. Firefox uniquely accepts this format.
	.p12	May	PKCS#12 (RFC 7292)	PKCS#12 (RFC 7292) is a generic DER encoded container format. May (typically) contain one or more X.509v3 certificate and may (typically) contain a DER encoded private key as well as other types of data. Format accepted by MSIE and Chrome browsers.
	.pfx	Yes	RFC 7292	PKCS#12 (RFC 7292) is a generic DER encoded container format. Same as .p12 but typically used on Microsoft systems and by convention has one (or more) DER X.509v3 certificate(s) and a DER private key. Format accepted by MSIE and Chrome browsers.

	.p7b	No	PKCS#7 (or RFC 5652)	PKCS#7 (or RFC 5652) CMS DER container encoded as PEM. May contain one or more certificates as well as other objects. Format accepted by MSIE, Firefox and Chrome browsers. May or may not be PEM encoded.
Miscellaneous	.csr	No	PEM/PKCS#10	May also use the suffix .pem. Certificate Request (typically known as a CSR). Contains a PEM encoded PKCS#10 (RFC 7292) DER format container consisting of the users public key, algorithm type and required attributes to be added to the certifcate.
	.crl	No	PKCS#7 or certificate list structure	Certificate Revokation List (CRL) is a DER encoded certificate list structure. May be contained in a PKCS#7 (RFC 2315 container or more typically a single DER encoded certificate list structure defined by RFC 5280 Section 5. Normally PEM encoded. Format accepted by MSIE and Chrome browsers.

Certificate Bundles

Nominally a client is responsible for validating any end-entity certificate it receives from a server. This increasingly involves Intermediate and/or Cross certificates. Thus, what historically used to be a single certificate distribution is increasingly becoming a multi-certificate distribution. Such multi-certificate distributions are typically called certificate bundles or ca-bundles or certificate chains. Updating millions of clients in a timely fashion presents serious logistics problems so it is becoming increasing popular to distribute any new Intermediate or Cross certificates (even root certificates) via the server. There are three broad methods by which multi-certificate bundles can be created:

1. Using a PKCS#7 (or RFC 5652) structure - usually with a file of suffix .p7b (supported by MSIE and Chrome for multi-certificate import). This file suffix will never have a private key.

2. Using a PKCS#12 (RFC 7292) structure (which is a super container for PKCS#7 and PKCS#8) - may have file suffix .p12 or .pkx (supported by MSIE and Chrome for multi-certificate import). By convention .pkx has both a certificate (PKCS#7) and a private key (PKCS#8), .p12 may or may not have a private key. .pfx is the suffix required by IIS Web Servers (though .p12 is also supported).

3. Concatenated PEM encoded certificates in a particular order. Since PEM certificate files (PEM keyword CERTIFICATE) are text files they can be concatenated manually using a text editor or using a unix command like:

```
cat intermediate2.crt intermediate1.crt root.crt > ca.pem
# the order in which these files appear reflects the
# validation sequence used by the client
# from server cert thru intermediate/cross certs to root cert
# resulting file (ca.pem in this case) would use
# SSLCACertificateFile Apache 2  directive
```

This PEM format kluge is widely adopted because of its support by Apache. These certificate bundles will never have a private key.

OpenSSL Conversion, Extraction and Manipulation

The following command show a number of manipulations using the OpenSSL package. OpenSSL uses a native PEM format.

```
# conversion PKCS12 > Extract PEM Certificate
openssl pkcs12 -clcerts -nokeys -in cert.p12 -out usercert.pem
openssl pkcs12 -clcerts -nokeys -in cert.p12 -out usercert.crt
# extracts certificate only

openssl pkcs12 -nocerts -in cert.p12 -out userkey.pem
openssl pkcs12 -nocerts -in cert.p12 -out userkey.key
# extracts key only

# conversion PEM > PKCS12 (.p12 or .pkx)
# The result .p12 or .pfx file is in DER (binary format)
openssl pkcs12 -export -out cert.p12 -inkey ./userkey.pem -in ./usercert.pem
openssl pkcs12 -export -out cert.pkx -inkey ./userkey.pem -in ./usercert.pem
#NOTE: in both the above cases a passphrase will be requested, to suppress
just hit enter
# at the request for password and its verification or use the following
command
openssl pkcs12 -export -out cert.p12 -inkey ./userkey.pem -in ./usercert.pem -
nodes -passout pass:
# converts PEM encoded cert and key to a DER encoded PKCS#12 format
```

PKCS#X to RFC Table

The following table cross-references commonly used PKCS standards to their RFC equivalent:

PKCS No.	RFC(s)	Notes
PKCS#1	RFC 8017	Container for RSA alogorithm covering cryptographic primitives, encryption schemes and signature schemes.
PKCS#5	RFC 8018	Method for password protection of encrypted data (used in PCKS#8, PKCS#7 and PKCS#12).
PKCS#7	RFC 2315	Cryptographic Message Syntax (CMS) container. Usually PEM encoded. Used for, among other entities, one or more CRLs (may be used with file suffix .crl though rarely), and one or more (ExtendedCertificatesAndCertificates) certificates (.p7b). A separate IETF CMS standard (broadly, but not perfectly, compatible with PKCS#7) is defined by RFC 5652.
PKCS#8	RFC	RFC replaces the PKCS#8 specification. Key container allows for both

	5958	private and public keys but predominantly associated with private keys. Optionally may be encrypted. If PEM encoded RFC recommends use of file suffix .pem, if DER encoded use of file suffix .p8 (not widely supported).
PKCS#9	RFC 2985 and RFC 7894	Extended Attributes that may be used in PKCS#7, PKCS#8 and PKCS#10.
PKCS#10	RFC 2986 updated by RFC 5967	DER encoded Certificate Request container. Contians a number of attributes describing the public key algorithm and attributes that will be incorportated into the final certificate. Amost universally PEM format.
PKCS#12	RFC 7292	Generic container for Personal Information. Container consists of PKCS#7 and PKCS#8 containers inside a secure structure. File suffixes of .p12 and .pkx. DER encoded, never PEM.

Handling Certificates In Common Browsers

Certificates may be imported into a number of systems using the procedures defined below. This information changes from time to time. The browser version number is included to indicate the, then current, import method. This method describes importing Root or Intermediate certificates (in the cases of Chrome and MSIE the appropriate tab (Intermediate or Trusted Root) must be selected. Only Firefox will accept a .pem suffix directly. For both MSIE and Chrome any .pem root certificate produced by methods 1, 2 3 or 3A can be simply renamed ca/cacert.pem -> ca/cacert.cer or ca/cacert.crt as you choose.

MSIE (11): MSIE will accept a .cer, .crt, .p7b, .pfx, .p12 suffix (among others). MSIE->Tools->Internet Options->Content Tab->click view certificates button->select appropriate certificate store->click Import and follow the wizard prompts.

For Windows 7+ systems an alternative method is to use the Microsoft Management Console (MMC) with the Certificate Snap-in installed and navigate to the appropriate certificate store, click the Actions menu->All tasks->import.. then follow the wizard prompts (will accept a .cer, .crt, .p7b, .pfx, .p12 suffix (among others)).

In an AD environment the certificate can also be propagated to all users via Group Policy Object (GPO). Load Administrative Tools->Group Policy Manager->Expand domains->Right click on Default Domain Policy and select Edit->Expand Computer configuration-> Expand Windows Settings->Expand Security Settings->Expand Public Key Settings->Right click Trusted Root Certificate Authorities->select Import and follow wizard Prompts.

Firefox (41.x.x) Firefox will accept either a .pem, .cer, .crt, .der or .p7b suffix. Tools Menu->Options->Advanced->Certificates tab->View Certificates->click import button and select file.

Chrome (46.x.x.x) Chrome will accept either a .cer, .crt, .p7b, .pfx, .p12 suffix (among others). Settings->Enable Advanced Settings->Scroll to HTTPS/SSL->Manage Certificates->Select Appropriate Tab->click import button and follow the wizard prompts.

Relevant RFCs

List of RFCs relevant to TLS, X.509 certificates and PKI. Most are referred to in the text above. Not exhaustive, but we now update it whenever we update the text or whenever a relevant RFC is published. It should eventually become exhaustive.

RFC 2315	PKCS #7: Cryptographic Message Syntax Version 1.5. B. Kaliski. March 1998. (Format: TXT=69679 bytes) (Status: INFORMATIONAL) (DOI: 10.17487/RFC2315) (see also RFC 5652 for related CMS)
RFC 2585	Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP. R. Housley, P. Hoffman. May 1999. (Format: TXT=14813 bytes) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC2585) (defines list of file suffixes and expected contents)
RFC 2986	PKCS #10: Certification Request Syntax Specification Version 1.7. M. Nystrom, B. Kaliski. November 2000. (Format: TXT=27794 bytes) (Obsoletes RFC2314) (Updated by RFC5967) (Status: INFORMATIONAL)
RFC 4210	Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP). C. Adams, S. Farrell, T. Kause, T. Mononen. September 2005. (Format: TXT=212013 bytes) (Obsoletes RFC2510) (Updated by RFC6712) (Status: PROPOSED STANDARD)
RFC 4211	Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF). J. Schaad. September 2005. (Format: TXT=86136 bytes) (Obsoletes RFC2511) (Status: PROPOSED STANDARD)
RFC 5019	The Lightweight Online Certificate Status Protocol (OCSP) Profile for High-Volume Environments. A. Deacon, R. Hurst. September 2007. (Format: TXT=46371 bytes) (Status: PROPOSED STANDARD)
RFC 5246	The Transport Layer Security (TLS) Protocol Version 1.2. T. Dierks, E. Rescorla. August 2008. (Format: TXT=222395 bytes) (Obsoletes RFC3268, RFC4346, RFC4366) (Updates RFC4492) (Updated by RFC5746, RFC5878, RFC6176) (Status: PROPOSED STANDARD)
RFC 5272	Certificate Management over CMS (CMC). J. Schaad, M.

	Myers. June 2008. (Format: TXT=167138 bytes) (Obsoletes RFC2797) (Updated by RFC6402) (Status: PROPOSED STANDARD)
RFC 5273	Certificate Management over CMS (CMC): Transport Protocols. J. Schaad, M. Myers. June 2008. (Format: TXT=14030 bytes) (Updated by RFC6402) (Status: PROPOSED STANDARD)
RFC 5274	Certificate Management Messages over CMS (CMC): Compliance Requirements. J. Schaad, M. Myers. June 2008. (Format: TXT=27380 bytes) (Updated by RFC6402) (Status: PROPOSED STANDARD)
RFC 5280	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk. May 2008. (Format: TXT=352580 bytes) (Obsoletes RFC3280, RFC4325, RFC4630) (Updated by RFC6818) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC5280)
RFC 5652	Cryptographic Message Syntax (CMS). R. Housley. September 2009. (Format: TXT=126813 bytes) (Obsoletes RFC3852) (Also STD0070) (Status: INTERNET STANDARD) (DOI: 10.17487/RFC5652) (almost backward compatible with PKCS#7 RFC 2315)
RFC 5746	Transport Layer Security (TLS) Renegotiation Indication Extension. E. Rescorla, M. Ray, S. Dispensa, N. Oskov. February 2010. (Format: TXT=33790 bytes) (Updates RFC5246, RFC4366, RFC4347, RFC4346, RFC2246) (Status: PROPOSED STANDARD)
RFC 5878	Transport Layer Security (TLS) Authorization Extensions. M. Brown, R. Housley. May 2010. (Format: TXT=44594 bytes) (Updates RFC5246) (Status: EXPERIMENTAL)
RFC 5912	New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX). P. Hoffman, J. Schaad. June 2010. (Format: TXT=216154 bytes) (Updated by RFC6960) (Status: INFORMATIONAL)
RFC 5914	Trust Anchor Format. R. Housley, S. Ashmore, C. Wallace. June 2010. (Format: TXT=28393 bytes) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC5914)
RFC 5937	Using Trust Anchor Constraints during Certification Path Processing. S. Ashmore, C. Wallace. August 2010. (Format: TXT=16900 bytes) (Status: INFORMATIONAL) (DOI: 10.17487/RFC5937)
RFC 5958	Asymmetric Key Packages. S. Turner. August 2010. (Format: TXT=26671 bytes) (Obsoletes RFC5208) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC5958) (replaces PKCS#8)
RFC 5967	The application/pkcs10 Media Type. S. Turner. August 2010. (Format: TXT=10928 bytes) (Updates RFC2986)

	(Status: INFORMATIONAL)
RFC 6066	Transport Layer Security (TLS) Extensions: Extension Definitions. D. Eastlake 3rd. January 2011. (Format: TXT=55079 bytes) (Obsoletes RFC4366) (Status: PROPOSED STANDARD)
RFC 6125	Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS). P. Saint-Andre, J. Hodges. March 2011. (Format: TXT=136507 bytes) (Status: PROPOSED STANDARD)
RFC 6347	Datagram Transport Layer Security Version 1.2. E. Rescorla, N. Modadugu. January 2012. (Format: TXT=73546 bytes) (Obsoletes RFC4347) (Updated by RFC7507) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC6347)
RFC 6176	Prohibiting Secure Sockets Layer (SSL) Version 2.0. S. Turner, T. Polk. March 2011. (Format: TXT=7642 bytes) (Updates RFC2246, RFC4346, RFC5246) (Status: PROPOSED STANDARD)
RFC 6402	Certificate Management over CMS (CMC) Updates. J. Schaad. November 2011. (Format: TXT=66722 bytes) (Updates RFC5272, RFC5273, RFC5274) (Status: PROPOSED STANDARD)
RFC 6712	Internet X.509 Public Key Infrastructure -- HTTP Transfer for the Certificate Management Protocol (CMP). T. Kause, M. Peylo. September 2012. (Format: TXT=21308 bytes) (Updates RFC4210) (Status: PROPOSED STANDARD)
RFC 6818	Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revokation List (CRL) Profile. P Yee. January 2013. (Format: TXT=17439 bytes) (Updates RFC5280) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC6818)
RFC 6960	X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams. June 2013. (Format: TXT=82037 bytes) (Obsoletes RFC2560, RFC6277) (Updates RFC5912) (Status: PROPOSED STANDARD)
RFC 6961	The Transport Layer Security (TLS) Multiple Certificate Status Request Extension. Y. Pettersen. June 2013. (Format: TXT=21473 bytes) (Status: PROPOSED STANDARD)
RFC 6962	Certificate Transparency. B. Laurie, A. Langley, E. Kasper. June 2013. (Format: TXT=55048 bytes) (Status: EXPERIMENTAL) (DOI: 10.17487/RFC6962)
RFC 7027	Elliptic Curve Cryptography (ECC) Brainpool Curves for

	Transport Layer Security (TLS). J. Merkle, M. Lochter. October 2013. (Format: TXT=16366 bytes) (Updates RFC4492) (Status: INFORMATIONAL)
RFC 7030	Enrollment over Secure Transport. M. Pritikin, Ed., P. Yee, Ed., D. Harkins, Ed.. October 2013. (Format: TXT=123989 bytes) (Status: PROPOSED STANDARD)
RFC 7091	GOST R 34.10-2012: Digital Signature Algorithm. V. Dolmatov, Ed., A. Degtyarev. December 2013. (Format: TXT=39924 bytes) (Updates RFC5832) (Status: INFORMATIONAL)
RFC 7093	Additional Methods for Generating Key Identifiers Values. S. Turner, S. Kent, J. Manger. December 2013. (Format: TXT=7622 bytes) (Status: INFORMATIONAL)
RFC 7250	Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). P. Wouters, Ed., H. Tschofenig, Ed., J. Gilmore, S. Weiler, T. Kivinen. June 2014. (Format: TXT=38040 bytes) (Status: PROPOSED STANDARD)
RFC 7251	AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS. D. McGrew, D. Bailey, M. Campagna, R. Dugal. June 2014. (Format: TXT=18851 bytes) (Status: INFORMATIONAL)
RFC 7292	PKCS #12: Personal Information Exchange Syntax v1.1. K. Moriarty, Ed., M. Nystrom, S. Parkinson, A. Rusch, M. Scott. July 2014. (Format: TXT=58991 bytes) (Status: INFORMATIONAL) (DOI: 10.17487/RFC7292)
RFC 7457	Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). Y. Sheffer, R. Holz, P. Saint-Andre. February 2015. (Format: TXT=28614 bytes) (Status: INFORMATIONAL)
RFC 7468	Textual Encodings of PKIX, PKCS, and CMS Structures. S. Josefsson, S. Leonard. April 2015. (Format: TXT=41594 bytes) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7468)
RFC 7507	TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks. B. Moeller, A. Langley. April 2015. (Format: TXT=17165 bytes) (Updates RFC2246, RFC4346, RFC4347, RFC5246, RFC6347) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7507)
RFC 7525	Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). Y. Sheffer, R. Holz, P. Saint-Andre. May 2015. (Format: TXT=60283 bytes) (Also BCP0195) (Status: BEST CURRENT PRACTICE) (DOI: 10.17487/RFC7525)
RFC 7568	Deprecating Secure Sockets Layer Version 3.0. R. Barnes, M. Thomson, A. Pironti, A. Langley. June 2015. (Format:

	TXT=13489 bytes) (Updates RFC5246) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7568)
RFC 7627	Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension. K. Bhargavan, Ed., A. Delignat-Lavaud, A. Pironti, A. Langley, M. Ray. September 2015. (Format: TXT=34788 bytes) (Updates RFC5246) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7627)
RFC 7633	X.509v3 Transport Layer Security (TLS) Feature Extension. P. Hallam-Baker. October 2015. (Format: TXT=21839 bytes) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7633)
RFC 7670	Generic Raw Public-Key Support for IKEv2. T. Kivinen, P. Wouters, H. Tschofenig. January 2016. (Format: TXT=21350 bytes) (Updates RFC7296) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7670)
RFC 7685	A Transport Layer Security (TLS) ClientHello Padding Extension. A. Langley. October 2015. (Format: TXT=7034 bytes) (Updates RFC5246) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7685)
RFC 7711	PKIX over Secure HTTP (POSH). M. Miller, P. Saint-Andre. November 2015. (Format: TXT=41302 bytes) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7711)
RFC 7773	Authentication Context Certificate Extension. S. Santesson. March 2016. (Format: TXT=33338 bytes) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7773)
RFC 7804	Salted Challenge Response HTTP Authentication Mechanism. A. Melnikov. March 2016. (Format: TXT=39440 bytes) (Status: EXPERIMENTAL) (DOI: 10.17487/RFC7804)
RFC 7817	Updated Transport Layer Security (TLS) Server Identity Check Procedure for Email-Related Protocols. A. Melnikov. March 2016. (Format: TXT=29855 bytes) (Updates RFC2595, RFC3207, RFC3501, RFC5804) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7817)
RFC 7894	Alternative Challenge Password Attributes for Enrollment over Secure Transport. M. Pritikin, C. Wallace. June 2016. (Format: TXT=19712 bytes) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7894)
RFC 7906	NSA's Cryptographic Message Syntax (CMS) Key Management Attributes. P. Timmel, R. Housley, S. Turner. June 2016. (Format: TXT=145888 bytes) (Status: INFORMATIONAL) (DOI: 10.17487/RFC7906)
RFC 7918	Transport Layer Security (TLS) False Start. A. Langley, N. Modadugu, B. Moeller. August 2016. (Format: TXT=23825 bytes) (Status: INFORMATIONAL) (DOI: 10.17487/RFC7918)

RFC 7919	Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS). D. Gillmor. August 2016. (Format: TXT=61937 bytes) (Updates RFC2246, RFC4346, RFC4492, RFC5246) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7919)
RFC 7924	Transport Layer Security (TLS) Cached Information Extension. S. Santesson, H. Tschofenig. July 2016. (Format: TXT=35144 bytes) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7924)
RFC 7925	Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things. H. Tschofenig, Ed., T. Fossati. July 2016. (Format: TXT=141911 bytes) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7925)
RFC 7935	The Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure. G. Huston, G. Michaelson, Ed.. August 2016. (Format: TXT=17952 bytes) (Obsoletes RFC6485) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7935)
RFC 8017	PKCS #1: RSA Cryptography Specifications Version 2.2. K. Moriarty, Ed., B. Kaliski, J. Jonsson, A. Rusch. November 2016. (Format: TXT=154696 bytes) (Obsoletes RFC3447) (Status: INFORMATIONAL) (DOI: 10.17487/RFC8017)
RFC 8018	PKCS #5: Password-Based Cryptography Specification Version 2.1. K. Moriarty, Ed., B. Kaliski, A. Rusch. January 2017. (Format: TXT=80887 bytes) (Obsoletes RFC2898) (Status: INFORMATIONAL) (DOI: 10.17487/RFC8018)

Change Log

The **Page modified** date at the foot of this page is always correct.

25th January 2017: Update PKCS#5 RFC Reference. Addition of PKCS#1 with RFC reference. Addition of RFCs 8017 and 8018.
16th September 2016: Typo in CRMF abbreviation. Note of RFC7918 'false start' whereby message transmission from the client may start after sending 'Finished' and prior to receipt of 'Finished' from server. Addition of RFCs 7918, 7918 and 7935.
28th July 2016: Updated RFCs and new section on use of subject and subjectAltName in certificates.
13th July 2016: Missing hyperlinks.
13th February 2016: Updated RFCs, incorrect hyperlink. New section on use of certificates in hosting providers (multi-tenant) environments.
2nd November 2015: Additional notes on TLS/SSL/Cert file names, formats and PKCS containers. Updated information about importing certificates for Chrome, Windows, MSIE and Firefox.

28th October 2015: RFCs Updated. Addition of End-Entity certificate definition.
23rd October 2015: RFCs Updated. Padding extension in ClientHello noted. RFC 7633 Pivate Internet X.509 extension allows cross referencing to TLS features to assist in attack prevention. Addition of OIDs to differentiate between Standard X.509 extensions (under 2.5.29) and Private Internet extensions (under 1.3.6.1.5.5.7.1). Some reformatting for smaller (mobile) screens.
29th September 2015: RFCs Updated. Extended Master Secret - note added to TLS/SSL description.
14th July 2015: RFCs Updated. Note added over deprecation of SSL v3.0.
10th July 2015: RFCs Updated. Note added about Data Transmission Content Protection (DTCP) certificates and their usage in the TLS handshake protocol as defined in RFC 7562.
30th May 2015: RFCs Updated. Note added about TLS_FALLBACK_SCSV defined in RFC 7507.
15th March 2015: RFCs Updated.
5th January 2015: Fixed incorrect href values for x509-chaining, changed incorrect reference to subjectKeyInfo to subjectPublicKeyInfo. Fixed error in subjectAltName to correctly identify use of dNSName attribute.
4th July 2014: Note in ClientHello, ServerHello, X.509 amd SubjectPublicKeyInfo about the vestigal certificate format defined in RFC 7250. RFCs updated.
21st January 2014: Note in ClientHello about Server Name Indication (SNI). RFCs updated.
22nd Dec 2013: RFCs updated.
18th Dec 2013: RFCs updated.
Nov 2013: Page converted to HTML5
Nov 2013: Reversed order on the page in all cases from SSL/TLS to TLS/SSL to reflect increasing reality.
Nov 2013: Updated Figure 2
Oct 2013: RFC update RFC 7027 & 7030. Updated page text covering ECCs

问题，意见，建议，更正（包括断开的链接）或要添加的东西？请把忙碌的生活时间从"邮件我们"（在屏幕顶部），网站管理员（下面）或zytrax的信息支持。你将有一个温暖的内心发光在一天的剩余时间。

网站由zytrax

web-master at zytrax
页面修改：2017年1月26日。