SSL 证书原理及格式

SSL 数字证书是在我们上网时用来验证服务器的真实身份的。SSL 是传输层的安全协议，后来改进为 TLS，不过它用的数字证书我们仍然称为 SSL 证书。

SSL 协议的简单原理如下：

客户端发送（自己支持的 ssl/tls 协议及相关参数）给服务端
服务端（选择其中一个协议版本及相关参数）发给客户端
服务端发送（自己的 ssl 证书）给客户端
客户端（在本地查找可靠的 ca 证书去验证服务端的证书是否可信）
若服务端证书可信
客户端发送（用服务端证书里的公钥去加密密钥交换的信息）给服务端
服务端发送（用服务端自己的私钥加密的密钥交换信息）给客户端
双方计算交换后的密钥（为对称的加密算法）
客户端用交换后的密钥发一些验证信息给服务端，服务端正确响应，则 SSL 验证通过。
接下来：
双方 使用对称的加密算法去加密正式的数据
当 ssl 会话到期后，再重新交换新的密钥

ssl 用到了非对称的密钥体制中的 RSA 算法，以及 DH 密钥交换算法（当然，这个密钥交换可能升级了，不再是原来的 DH 算法了，可能是 ECDHE），双方交换的是什么密钥？是对称加密算法的密钥，具体是哪个对称加密算法是由 ssl 握手时协商的，可能是 DES，也可能是 AES。

RSA 算法：

利用大整数因子分解的困难性
先随机生成 2 个大素数 p 和 q，再计算 n=p*q
$\psi(n)=(p-1)*(q-1)$
随机数 e $(0<e<\psi(n))$满足 $gcd(e,\psi(n))=1$，即 e 与 $\psi(n)$互素
计算 $d=e^{-1} \bmod \psi(n)$
然后就得到 公钥 pub-key 为 e,n
私钥 pri-key 为 d,n

加密算法：密文=$明文^{e} \bmod n$

解密算法：明文=$密文^{d} \bmod n$

加密和解密是相对的，互逆的，当用 e 去加密时，就得用 d 去解密，当用 d 去加密时，就得用 e 去解密。要想通过 e 得知 d 是不可能的，要想通过 d 得知 e 也是不可能的，所以服务端生成了密钥对 ed(n)后，把其中的公钥给客户端，自己保留私钥，这样就可以互相加密解密了，

服务端用私钥加密的信息，所有的客户端都可以看到，因为客户端很轻松就能获得服务端的证书里的公钥，但客户端发的加密消息（用公钥加密），只有服务端能看到，因为私钥只存在服务端手里。

所以 RSA 算法不是用来加密消息的，因为客户端都能看到服务端的消息，它只是用来做数字签名的，顺便交换密钥。

、

DH 密钥交换算法:

利用离散对数问题的难解性
q 为一个素数，a 为 q 的一个本原元，要交换密钥的双方 已知 a,q

$Y = a^{x} \mod q$

给出 x 容易算出 Y，给出得知 Y 时，反而不易（很难）算出 X

密钥交换过程如下：
用户 A 和用户 B 协商 q 和 a
用户 A 产生一个随机数 X(A)<q　　用户 B 产生一个随机数 X(B)<q

用户 A 计算 $Y(A)=a^{x(A)} \mod q$ 再把 Y(A)发给用户 B

用户 B 计算 $Y(B)=a^{x(B)} \mod q$ 再把 Y(B)发给用户 A

用户 A 根据 B 发来的 Y(B)计算出 $k=(Y(B))^{X(A)} \mod q$

用户 B 根据 A 发来的 Y(A)计算出 $k=(Y(A))^{X(B)} \mod q$

然后，用户 A 和用户 B 算出的 k 是一样的，这个 k 就是交换后的密钥
当其他人截获了 AB 的信息，得知了 q,a,Y(A),Y(B)也无法计算出 k
因为其他人不知道 X(A)和 X(B)，前面讲过，给出 Y，是不易算出 X 的。

密钥交换过程中就算使用明文，其他人也无法得知 AB 双方交换后的 k，比较安全，但单纯使用密钥交换算法是无法应对中间人攻击的，所以得用上 RSA 算法，其中服务端使用私钥去加密要交换的 Y(A)，客户端用公钥去加密要交换的 Y(B)，这样就可以应对中间人攻击，接下来要信任的就是服务端了，因为其他人也可以宣称自己就是这个服务端，它做了流量的劫持，不再做中间人了，自己替代了目标服务器。所以我们需要一个共同信任的机构，由这个机构去对所有的其他服务器的公钥及相关信息做一个签名，生成一个证书，没错，就是 ssl 证书。

（这个交换的密钥是什么？ 当然是对称的加密算法的密钥了，ssl 握手完成后，通信双方就使用对称的加密算法去加密 正式的数据了，因为对称加密算法执行速度比非对称的快）

签名：

可信的CA（验证机构）使用它的私钥把 其他服务端的（公钥及相关信息）加密（这就叫签名），生成一个ssl证书，服务端收到CA发的证书后，部署到服务器上，当有客户请求ssl连接时，服务端再把这个证书给客户端，客户端用可信CA的公钥去解密此证书，解密后的内容再和此证书上所宣称的信息一比对，就可得知此证书的真伪了。
可信CA的公钥存放在它的CA证书里，顶级CA证书是由它自己来签名的（中间CA的证书是由上层CA来签名的）。可信的CA一般在操作系统安装时就放在操作系统里了。

话说我们凭什么就要相信这些CA呢？因为这个验证机制是人家最早提出并得到了广泛使用的，不用它，还能用谁呢？

ssl证书里都有哪些内容呢？

ssl证书文件里有2部分内容，一部分是明文的，含服务端的相关身份信息及RSA的公钥还有表明由哪个CA给它签名的，以及密文的("服务端的相关身份信息及RSA公钥"的hash值）这个密文是由（明文里表明的）可信CA去加密的。

这样我们就可以从操作系统里找到此证书所说的那个CA的公钥，用这个CA的公钥去解密此证书的密文部分，得出密文里的hash信息再和 根据此证书的明文部分计算出的hash比对，如果吻合则说明此证书可信。这样服务端的身份就可信了吗？其实没那么快就相信服务端的身份，因为真实服务端的证书我们所有客户端都可以获得的，所以也有可能是其他人用此证书去充当服务端，不过没关系，它没有服务端的私钥，所以我们在ssl验证时，在交换完密钥后，会发一些验证消息，以验证服务端的真实性。因为如果伪造的服务端没有真实服务端的私钥，它发过来的交换信息在客户端这边用真实服务端的公钥解密时，就不能正确解密，所以完成密钥交换后，双方得到的key是不一致的。这样完成密钥交换后，再发一些验证消息，这个伪造的服务端就被识破了。

当客户端找不到证书里表明的CA时，就会提示连接是不安全的，无法验证此网站的身份。

证书链：

可信的CA只是少数的，后来申请ssl证书的人越来越多，于是这些顶级的CA又把签名权下放到其他的商家那里，由其他商家充当中间CA，我们现在申请的ssl证书一般都是中间CA签名的，中间CA的证书是由顶层CA签名的。而我们操作系统里可能没有预装中间CA的证书，导致我们无法验证由中间CA签名的ssl证书，这时就需要网站服务端提供证书链，从证书链中获取中间CA的证书，再从中间CA证书去找更上层直至顶层CA的证书。这样就完成了最下层ssl证书的验证。

有时候浏览器会自作聪明地 自己去找中间CA的证书再去找顶级CA，就是说它不看网站服务端给的证书链了，它自己去找合适的证书链。这也就是 有时我们网站上的证书链明明是4层的，结果在浏览器里只有3层 的原因了。（火狐浏览器可以导出它自己寻到的证书链）

ssl 证书的格式

ssl 数字证书文件的内容可以是文本格式，也可以是二进制格式。
常用的证书标准及文件格式如下：

编码格式	证书文件后缀	说明
x.509	.cer .crt .der .pem	证书里含 RSA 的公钥，相关身份信息
pkcs#7	.p7b	含 RSA 的公钥，可含 CRL 信息
pkcs#12	.p12 .pfx	可含 RSA 的私钥，有密码保护
jks	.jks	java 特有的，可含私钥

怎么申请自己网站的 ssl 证书呢，

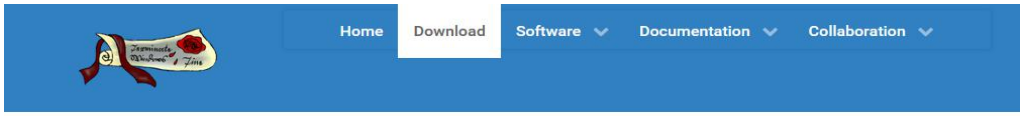
1. 先在本地生成.key 密钥对文件，这个 xx.key 文件叫作密钥对文件（有些人称它为私钥文件是不正确的），因为这个.key 文件是 RSA 算法生成的公钥及私钥文件，公钥和私钥必需是成对生成的，不可能先生成一个私钥，再随后生成公钥。
2. 把 xx.key 密钥对文件里的公钥取出来，加上一些网站的身份信息，合成一个.csr 文件，.csr 文件叫作证书请求文件，里面的中文信息用 utf-8 编码。
3. 我们把.csr 文件提交给某个可信 CA，由此 CA 去签名，生成一个证书文件，CA 服务商一般会提供多种格式的证书文件以及它们的证书链文件。
4. 我们再根据自己 web 服务器的要求去转换成相应的格式，比如 tomcat 的.jks 格式和 IIS 的.pfx 格式。

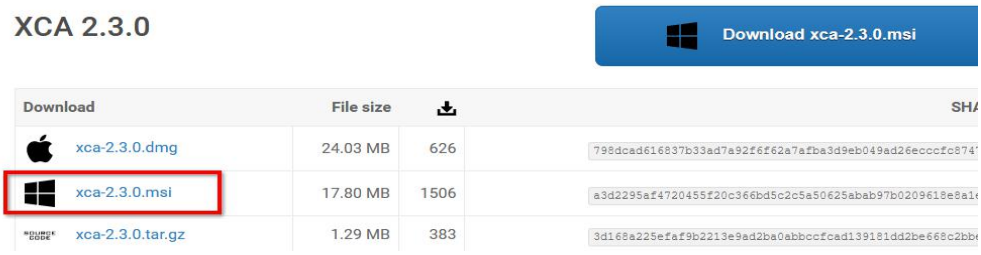
pkcs#7 证书可以为二进制文件，也可为 base64 的文本文件
pkcs#12 的证书不论后缀为何，都是二进制文件，
x.509 编码的话，.cer 和.crt 一般都是使用 PEM（base64）编码的文本文件，.der 为二进制文件
.pem 后缀证书可以是二进制的，也可以是 base64 文本的，具体的得打开看看才知道，还有证书链也有以.pem 为后缀的。证书链文件就是由多个相关证书的内容放在一个文件里生成的。

ssl 证书格式怎么相互转换呢，可以使用 XCA 证书管理工具，也可以使用 openssl 工具，这两者都是开源免费的。

XCA 证书管理工具的使用

到官网 https://hohnstaedt.de/xca/index.php/download 去下载新版的 xca 工具并安装





下载 xca-x.x.x.msi 这个安装包，安装后，打开 xca 软件

在主界面上，点击左上角菜单栏的“文件”→“新建数据库”
选择保存到某个目录下，名为 test.xdb
确定后，要求输入数据库的密码

点击 OK 后，就行了，接下来导入我们从 CA 证书商家那获得的 ssl 证书，

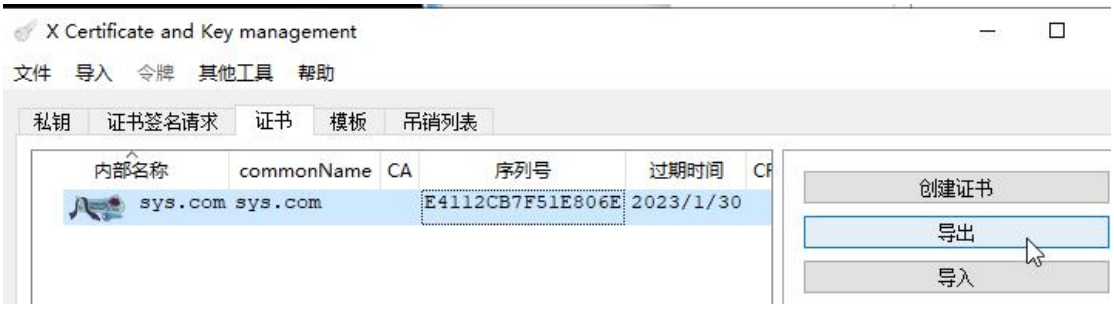
点击“私钥”，“导入”，选择目标.key/.pem/.der/.p8/.pvk/.pub 文件，例如 sys.com.key
其实导入的不止是私钥，因为.key 文件是密钥对文件，在 xca 里只显示其中的私钥，和私钥配对的公钥也是包含在此文件里的。

导入成功后，就会出现一个 key 列表信息，如上图
再导入证书文件，点击“证书”，“导入”

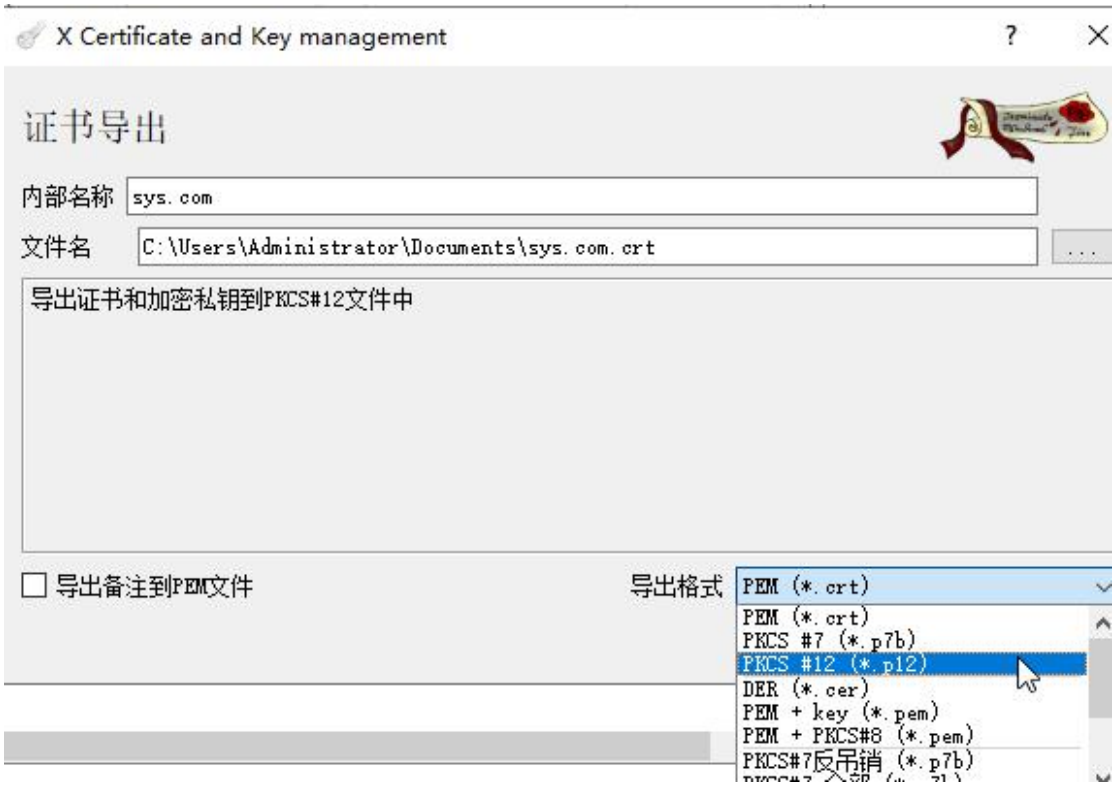
默认的导入只能导入 x.509 格式的.pem/.cer/.crt/.der
也可以点击下面的“导入 pkcs#12”（.pfx/.p12）和“导入 pkcs#7”（.p7b）

导入成功后，也会多出一个证书文件列表，如下图

在 xca 管理工具里，导入证书及密钥后，我们可以查看证书或密钥的详细信息，也可导出为其他格式的文件，比如导出为.p12 文件，（现在的.p12 文件和.pfx 文件是一样的，可以直接把.p12 后缀改为.pfx，就能在 iis 上使用了）



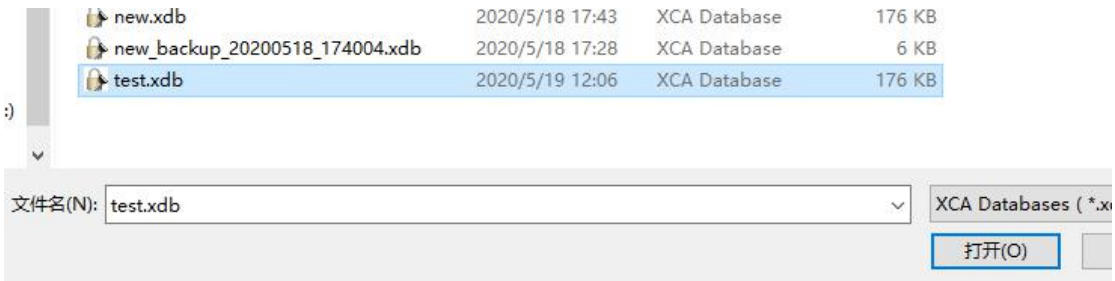
选中目标证书，点击“导出”



然后选择导出后的目录，以及导出的格式，如 pkcs12 的.p12 后缀格式，它可以直接改名为.pfx

XCA 除了可以转换证书格式，也可以生成 RSA 密钥对，并创建证书请求文件

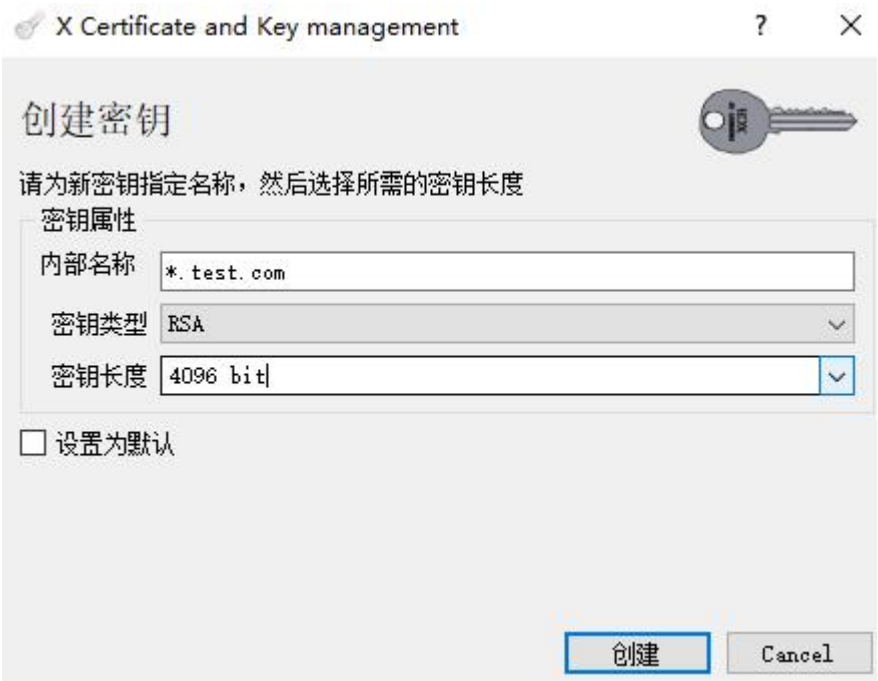
xca 在打开时，默认不会打开之前创建的数据库，要我们手动去打开，点击“文件”，“打开数据库”



找到目标数据库，点击“打开”

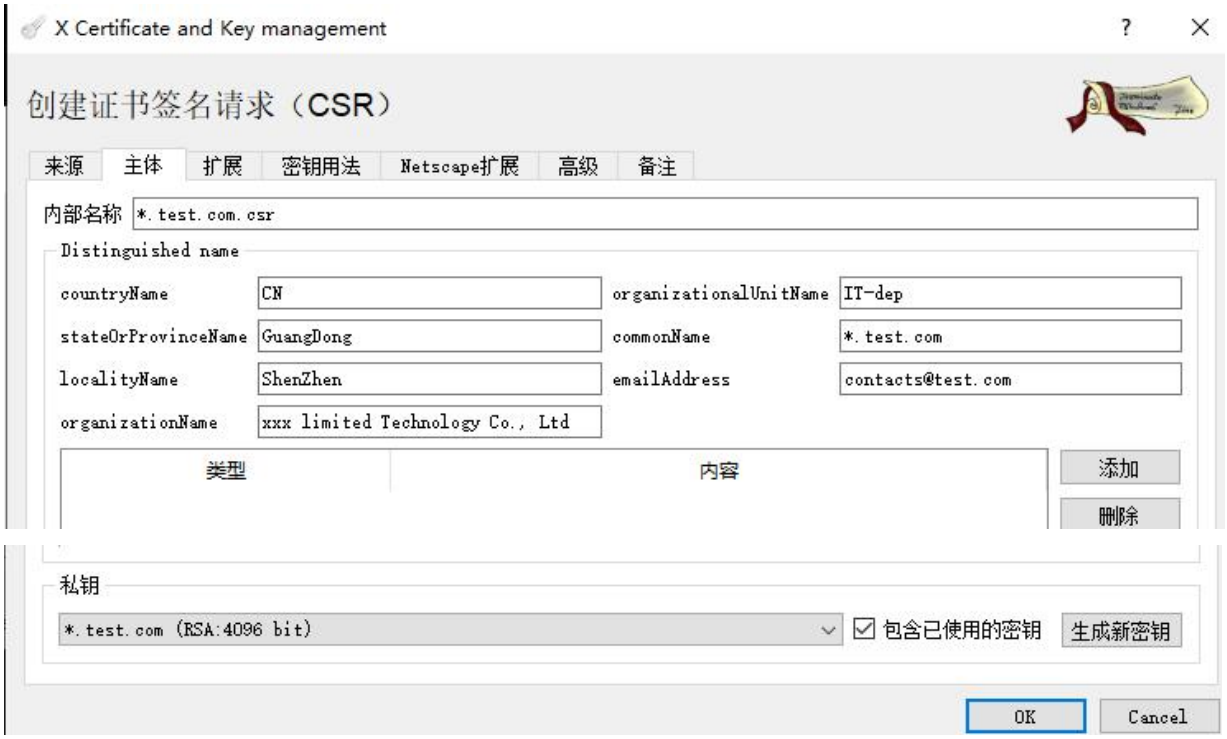
输入数据库的密码，点击“OK”

在“私钥”界面，点击“创建密钥”



名称随便写，比如*.test.com，一般是写网站域名或带*通配符的域名，密钥长度不低于 2048 位，现在推荐用 4096 位

然后在“证书签名请求”页，点击“创建请求”



如上图，证书签名请求文件里，要填写的不多，一般只写主体部分，其他的会写的就写，不会的就不写。内部名称就是请求文件在 xca 数据库里的文件名称，可以随便写，重要的是下面的 7 个空以及最下行的“私钥”，（这里得再说一下，key 文件不叫私钥文件，就叫密钥或密钥对文件，含 RSA 的公钥及私钥）中文是用 utf-8 编码

countryName 填国家代码，只能写 2 个英文字母，中国写 CN

stateOrProvicnceName 填写省级行政单位名，如 GuangDong，支持中文

localityName 填写地级行政单位名，如 ShenZhen，支持中文

organizationName 填写组织名称，一般写公司名，如 xxx xxx co., ltd 支持中文

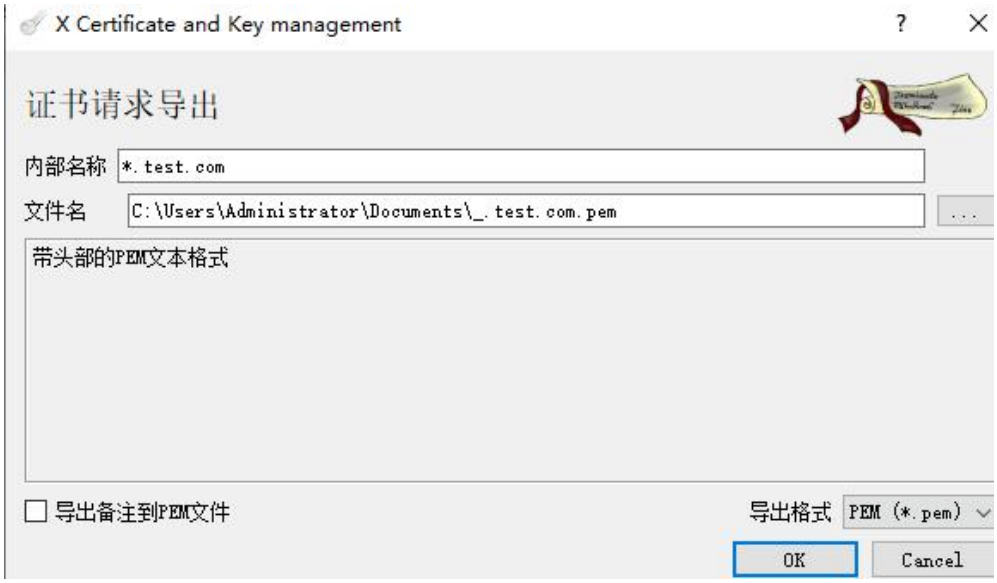
organizationlUnitName 组织单位名，一般写公司的某个部门，如 IT-dep，支持中文

commonName 通用名称，一般写域名，可写带*.的通配符域名

emailAddress 可写可不写

最下行的“私钥”是指密钥文件，表示 xca 管理界面里的“私钥”一栏里的密钥，在证书签名申请文件里，只会导入密钥中的公钥

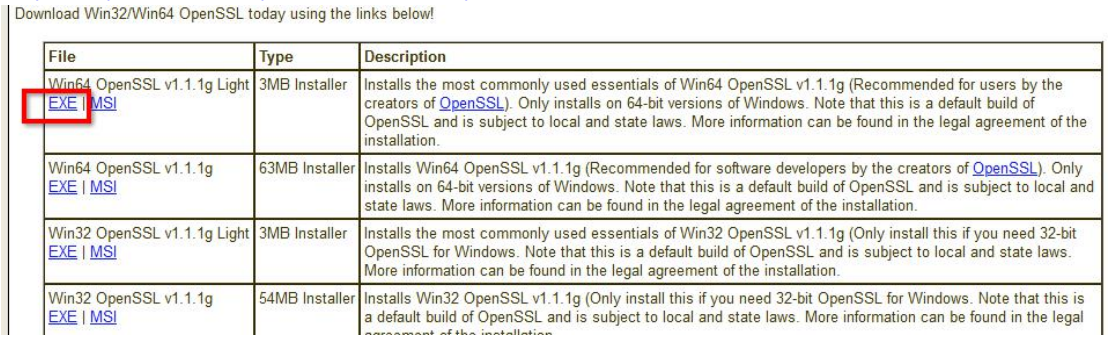
生成证书签名请求文件后，点击目标文件，点击“导出”



导出 csr 请求文件后（它的后缀不一定非得是.csr，也可以是.pem）就可以提交给 ssl 证书签发机构了。证书签发机构审核通过后，就会发给我们证书文件

OpenSSL 工具的使用

Linux 系统可以用 apt-get install openssl 或 yum install openssl 安装，windows 可以到 https://slproweb.com/products/Win32OpenSSL.html 网址下载，选择下载 exe 文件



Download Win32/Win64 OpenSSL today using the links below!

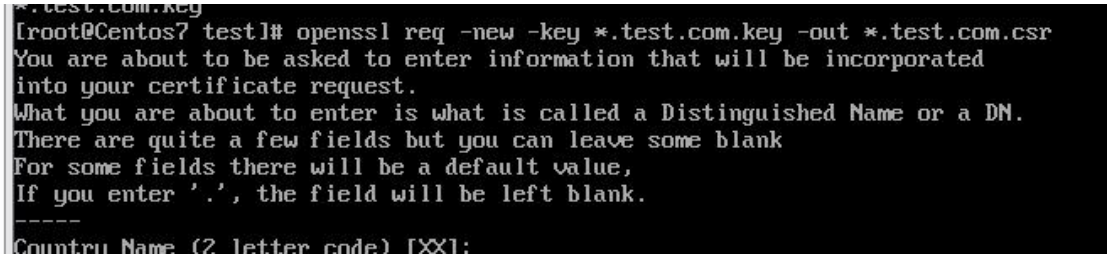
File	Type	Description
Win64 OpenSSL v1.1.1g Light EXE | MSI	3MB Installer	Installs the most commonly used essentials of Win64 OpenSSL v1.1.1g (Recommended for users by the creators of OpenSSL). Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win64 OpenSSL v1.1.1g EXE | MSI	63MB Installer	Installs Win64 OpenSSL v1.1.1g (Recommended for software developers by the creators of OpenSSL). Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win32 OpenSSL v1.1.1g Light EXE | MSI	3MB Installer	Installs the most commonly used essentials of Win32 OpenSSL v1.1.1g (Only install this if you need 32-bit OpenSSL for Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win32 OpenSSL v1.1.1g EXE | MSI	54MB Installer	Installs Win32 OpenSSL v1.1.1g (Only install this if you need 32-bit OpenSSL for Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal

windows 版安装后，在 cmd 里进入安装目录，再运行 openssl.exe 命令

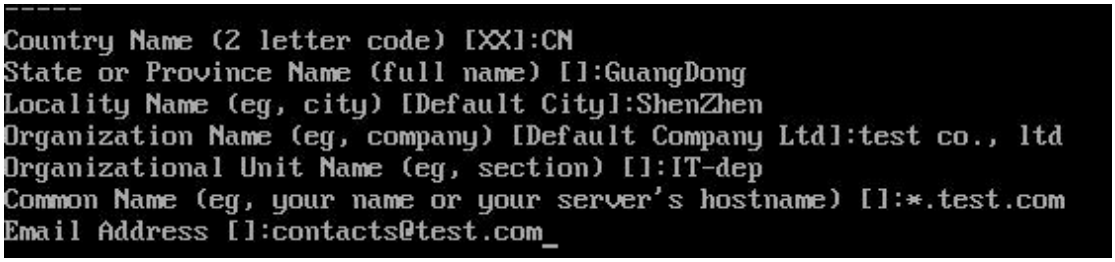
openssl genrsa -out *.test.com.key 生成.key 密钥对文件，默认是 2048 位

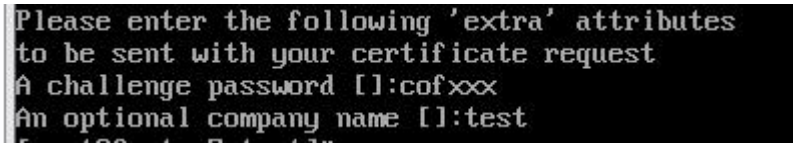
#openssl genrsa -out *.test.com.key 4096 指定 RSA 密钥长度

#openssl req -new -key *.test.com.key -out *.test.com.csr 创建证书签名申请文件



然后根据提示输入相关信息，





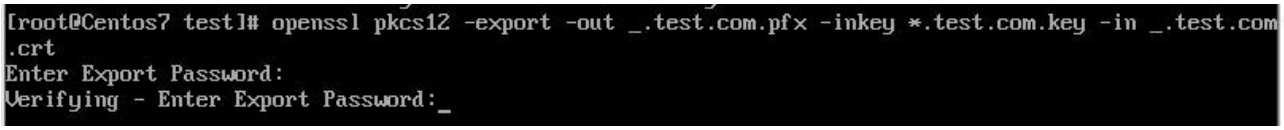
还要输入额外的信息，随便写，也可不写

然后把*.test.com.csr 请求文件提交给 CA 去签名，通过后得到_.test.com.crt 或.cer 之类的证书文件。

格式的转换：

把 x.509 编码转成 pkcs12

openssl pkcs12 -export -out _.test.com.pfx -inkey *.test.com.key -in _.test.com.crt



.pfx 文件要求输入保护密码

把.p7b 转为 x.509 的.crt

先转成 PEM 文本的文件，再转输出为 certs 格式的

openssl pkcs7 -inform der -in _.test.com.p7b -out _.test.com.pem.p7b

openssl pkcs7 -print_certs -in _test.com.pem.p7b -out _.test.com.crt

其他的不多说了，我也不会了，哈，自行网上搜索吧。

作者：Cof-Lee

2020 年 5 月 19 日