# 如何利用openssl查看.pem文件：

**∟"OpenSSL" Viewing Certificates in DER and PEM**

|  |  |
| --- | --- |
| *This section provides a tutorial example on how to use 'OpenSSL' to view certificates in DER and PEM formats generated by the 'keytool -exportcert' command.* |  |

One way to verify if "keytool" did export my certificate using DER and PEM formats correctly or not is to use "OpenSSL" to view those certificate files. To do this, I used the "openssl x509" command to view keytool\_crt.der and keytool\_crt.pem:

>openssl x509 -in keytool\_crt.pem -inform pem -noout -text

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 1185636568 (0x46ab60d8)

Signature Algorithm: dsaWithSHA1

Issuer: C=CA, ST=Herong State, L=Herong City, ...

...

>openssl x509 -in keytool\_crt.der -inform der -noout -text

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 1185636568 (0x46ab60d8)

Signature Algorithm: dsaWithSHA1

Issuer: C=CA, ST=Herong State, L=Herong City, ...

O=Herong Company, OU=Heron

...

Cool. "OpenSSL" can read certificates in DER and PEM formats generated by "keytool". What I learned so far:

* "keytool" can generate self-signed X5.09 version 3 certificates.
* "keytool" can export certificates with DER and PEM formats.
* "OpenSSL" can read certificates generated by "keytool" in both DER and PEM formats.

# 如何生成一个自签证书并安装：

https://www.sslshopper.com/article-how-to-create-and-install-an-apache-self-signed-certificate.html

SSL is an essential part of creating a secure Apache site. SSL certificates allow you encrypt all the traffic sent to and from your Apache web site to prevent others from viewing all of the traffic. It uses public key cryptography to establish a secure connection. This means that anything encrypted with a public key (the SSL certificate) can only be decrypted with the private key (stored only on the server) and vice versa.

**When to Use a Self Signed Certificate**

You should never use a self signed certificate on an e-commerce site or any site that transfers valuable personal information like credit cards, social security numbers, etc.

An SSL certificate is necessary for more than just distributing the public key: if it is signed by a trusted third-party, it verifies the identity of the server so clients know they aren’t sending their information (encrypted or not) to the wrong person. So what is a [self signed certificate](http://hello.k7mm.com/5i5.php?u=Y3pvdkwzZDNkeTV6YzJ4emFHOXdjR1Z5TG1OdmJTOWhjblJwWTJ4bExXaHZkeTEwYnkxamNtVmhkR1V0WVMxelpXeG1MWE5wWjI1bFpDMWpaWEowYVdacFkyRjBaUzVvZEcxcw%3D%3D&b=13)? It is a certificate that is signed by itself rather than a trusted third party. Isn’t that bad? In most cases, yes. You will almost never want to use a self signed certificate on a public Apache server that requires anonymous visitors to connect to your site because they could easily become a victim of a man-in-the-middle attack. However, self signed certificates have their place:

* **Self signed certificates can be used on an Apache development server**. There is no need to spend extra cash buying a trusted certificate when you are just developing or testing an application.
* **Self signed certificates can be used on an intranet**. When clients only have to go through a local intranet to get to the server, there is virtually no chance of a man-in-the-middle attack.
* **Self signed certificates can be used on personal sites with few visitors**. If you have a small personal site that transfers non-critical information, there is very little incentive for someone to attack the connection.

Just keep in mind that visitors will see a warning in their browsers (like the one below) when connecting to an Apache site that uses a self signed certificate until it is permanently stored in their certificate store. **You should never use a self signed certificate on an e-commerce site or any site that transfers valuable personal information like credit cards, social security numbers, etc.** Just lay down a few dollars on a trusted [cheap SSL certificate](http://hello.k7mm.com/5i5.php?u=Y3pvdkwzZDNkeTV6YzJ4emFHOXdjR1Z5TG1OdmJTOWphR1ZoY0dWemRDMXpjMnd0WTJWeWRHbG1hV05oZEdWekxtaDBiV3c9&b=13) or a [free SSL certificate](http://hello.k7mm.com/5i5.php?u=Y3pvdkwzZDNkeTV6YzJ4emFHOXdjR1Z5TG1OdmJTOWhjblJwWTJ4bExXWnlaV1V0YzNOc0xXTmxjblJwWm1sallYUmxjeTFtY205dExXRXRabkpsWlMxalpYSjBhV1pwWTJGMFpTMWhkWFJvYjNKcGRIa3VhSFJ0YkE9PQ%3D%3D&b=13).

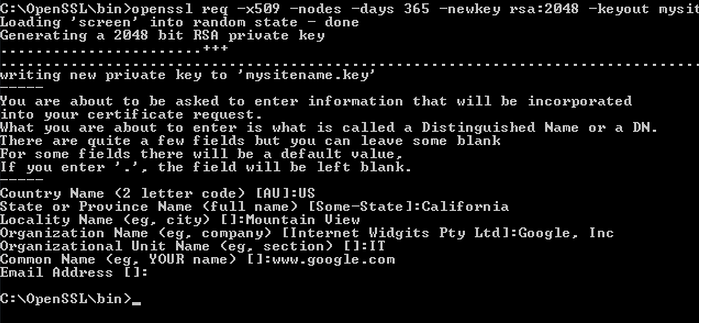


**Generate Your Apache Self Signed Certificate**

Great! So now you know when to use an **Apache self signed certificate** and when not to. Now, let’s create one: First, we need to make sure OpenSSL is installed. If you are installing the self signed certificates on Windows, grab the [Windows version of OpenSSL](http://hello.k7mm.com/5i5.php?u=T2k4dmQzZDNMbk5zY0hKdmQyVmlMbU52YlM5d2NtOWtkV04wY3k5WGFXNHpNazl3Wlc1VFUwd3VhSFJ0YkE9PQ%3D%3D&b=13) (If you get an error when you run the installer, you may need to download the Visual C++ 2008 Redistributables listed on that page first). If you are on another type of server, try running “openssl” on the command line to see if OpenSSL is already installed. If it is not, you will need to download a package or compile it from [its source](http://hello.k7mm.com/5i5.php?u=T2k4dmQzZDNMbTl3Wlc1emMyd3ViM0puTHc9PQ%3D%3D&b=13).

Once you have OpenSSL installed, just run this one command to create an Apache self signed certificate（这一步浓缩了很多步，下一个问题会细细分析）:

openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mysitename.key -out mysitename.crt



You will be prompted to enter your organizational information and a common name. The common name should be the fully qualified domain name for the site you are securing (www.mydomain.com). You can leave the email address, challenge password, and optional company name blank. When the command is finished running, it will create two files: a mysitename.key file and a mysitename.crt self signed certificate file valid for 365 days.

**Install Your Self Signed Certificate**

Now, you just need to configure your Apache virtual host to use the SSL certificate. If you only have one Apache virtual host to secure and you have an ssl.conf file being loaded, you can just edit that file. Otherwise, you will need to make a copy of the existing non-secure virtual host, paste it below, and change the port from port 80 to 443.

1. Open your Apache configuration file in a text editor. Depending on your operating system and Apache version, it will be located in different places but you will usually find it at /etc/httpd/httpd.conf. On a Windows machine, you will usually find it at C:\Program Files\Apache\Apache2\conf\httpd.conf
2. In most cases, you will find the <VirtualHost> blocks in a separate file in a directory like /etc/httpd/vhosts.d/ or /etc/httpd/sites/. Add the lines in bold below. <VirtualHost 192.168.0.1:**443**>  
   DocumentRoot /var/www/website  
   ServerName www.yourdomain.com **SSLEngine on  
   SSLCertificateFile /etc/ssl/crt/mysitename.crt  
   SSLCertificateKeyFile /etc/ssl/crt/mysitename.key**  
   </VirtualHost>
3. Change the names of the files and paths to match your certificate files. Save the changes and exit the text editor.
4. Restart your Apache web server using one of the following commands: /usr/local/apache/bin/apachectl startssl  
   /usr/local/apache/bin/apachectl restart

Learn more about [installing a certificate in Apache](http://hello.k7mm.com/5i5.php?u=Y3pvdkwzZDNkeTV6YzJ4emFHOXdjR1Z5TG1OdmJTOWhjR0ZqYUdVdGMyVnlkbVZ5TFhOemJDMXBibk4wWVd4c1lYUnBiMjR0YVc1emRISjFZM1JwYjI1ekxtaDBiV3c9&b=13).

**Check the Apache Self Signed Certificate Installation**

If the Apache site is public, you can use our [SSL Checker](http://hello.k7mm.com/5i5.php?u=Y3pvdkwzZDNkeTV6YzJ4emFHOXdjR1Z5TG1OdmJTOXpjMnd0WTJobFkydGxjaTVvZEcxcw%3D%3D&b=13) to verify that it is installed correctly (ignoring the warning that it is not trusted because it is self signed). Otherwise, just go to the website in your web browser using https in the address bar (https://www.mysitename.com) and verify that the certificate is being given out by the server by clicking the certificate icon (after clicking through the warnings).

# 生成自签证书并安装的详细版本，如何利用根证书对子证书签名

x509证书一般会用到三类文，key，csr，crt。

Key 是私用密钥openssl格，通常是rsa算法。

Csr 是证书请求文件，用于申请证书。在制作csr文件的时，必须使用自己的私钥来签署申，还可以设定一个密钥。

crt是CA认证后的证书文，（windows下面的，其实是crt），签署人用自己的key给你签署的凭证。

1.key的生成

openssl genrsa -des3 -out server.key 2048

这样是生成rsa私钥，des3算法，openssl格式，2048位强度。server.key是密钥文件名。为了生成这样的密钥，需要一个至少四位的密码。可以通过以下方法生成没有密码的key:

openssl rsa -in server.key -out server.key

server.key就是没有密码的版本了。

2. 生成CA的crt

openssl req -new -x509 -key server.key -out ca.crt -days 3650

生成的ca.crt文件是用来签署下面的server.csr文件。

3. csr的生成方法

openssl req -new -key server.key -out server.csr

需要依次输入国家，地区，组织，email。最重要的是有一个common name，可以写你的名字或者域名。如果为了https申请，这个必须和域名吻合，否则会引发浏览器警报。生成的csr文件交给CA签名后形成服务端自己的证书。

4. crt生成方法

CSR文件必须有CA的签名才可形成证书，可将此文件发送到verisign等地方由它验证，要交一大笔钱，何不自己做CA呢。

openssl x509 -req -days 3650 -in server.csr -CA ca.crt -CAkey server.key -CAcreateserial -out server.crt

输入key的密钥后，完成证书生成。-CA选项指明用于被签名的csr证书，-CAkey选项指明用于签名的密钥，-CAserial指明序列号文件，而-CAcreateserial指明文件不存在时自动生成。

最后生成了私用密钥：server.key和自己认证的SSL证书：server.crt

证书合并：

cat server.key server.crt > server.pem

# 如何利用openssl的私钥文件对一个明文文件进行加解密：

首先，生成一对rsa密钥对（包含公钥和私钥），当然私钥经过了des3加密，密码是helixiao，

openssl genrsa -des3 -out server.key 2048（会提示输入加密密钥的密码，我们用helixiao）

需要注意的是这个server.key文件包含了**公钥和密钥两部分**，也就是说这个文件即可用来加密也可以用来解密。后面的1024是生成密钥的长度。

**openssl可以将这个文件中的公钥提取出来：**

openssl rsa -in server.key -pubout -out server\_pub.key

-in指定输入文件，-out指定提取生成公钥的文件名。至此，我们手上就有了一个公钥，一个私钥（包含公钥）。现在可以将用公钥来加密文件了。

我在目录中创建一个hello的文本文件，然后利用此前**生成的公钥加密文件**：

openssl rsautl -encrypt -in hello -inkey server\_pub.key -pubin -out hello.enc

-in指定要加密的文件，-inkey指定密钥，-pubin表明是用纯公钥文件加密，-out为加密后的文件。

**解密文件：**

openssl rsautl -decrypt -in hello.en -inkey server.key -out hello.dec

-in指定被加密的文件，-inkey指定私钥文件，-out为解密后的文件。

至此，一次加密解密的过程告终。在实际使用中还可能包括证书，这个以后有机会再说~