深入浅出 SSL 证书

网络安全是一个永恒的话题，保护网络安全有很多方法，其中应用最为广泛的就是使用 SSL 证书来保护 C/S 或者 B/S 的通信安全。但是其实很多时候，我们并不知道如何管理和配置 SSL 证书。因此，笔者在本场 Chat 中将会和大家分享下面的话题。

- 什么是 SSL 证书
- 为什么要使用 SSL 证书
- 什么是对称加密和非对称加密
- Window 操作系统下如何管理 SSL 证书？
- 如何从 HTTPS 网站导出 CA 证书
- 如何开发一个 Eclipse 插件把证书导入到 Java 的 keystore。
- 自签名证书 VS CA 证书
- 如何通过可视化工具生成和管理公钥和私钥

1. 什么是 SSL 证书？

根据百度百科对SSL证书的定义，SSL证书是数字证书的一种，类似于驾驶证、护照和营业执照的电子副本。因为配置在服务器上，也称为SSL服务器证书。SSL 证书就是遵守SSL协议，由受信任的数字证书颁发机构CA（在有的测试环境也可能 是自签名的证书，也就是自己给自己颁发的证书），在验证服务器身份后颁发，具有服务器身份验证和数据传输加密功能。

2. 为什么要使用 SSL 证书

```
<!DOCTYPE HTML><html lang="en"><head>
<meta charset="UTF-8">
<title>Hello SSL</title>
</head>
<body>
<p>
<h3>SSL Example</H3>
<p>大家好，我是冰尘！！！</p>

</body></html>
```

然后启动Tomcat9，在打开浏览器之前，请先下载一个名字叫做Fiddler的软件，百度百科对Fiddler的介绍如下：

Fiddler是一个http协议调试代理工具，它能够记录并检查所有你的电脑和互联网之间的http通讯，设置断点，查看所有的“进出”Fiddler的数据（指cookie,html,js,css等文件，这些都可以让你胡乱修改的意思）。 Fiddler 要比其他的网络调试器要更加简单，因为它不仅仅暴露http通讯还提供了一个用户友好的格式。

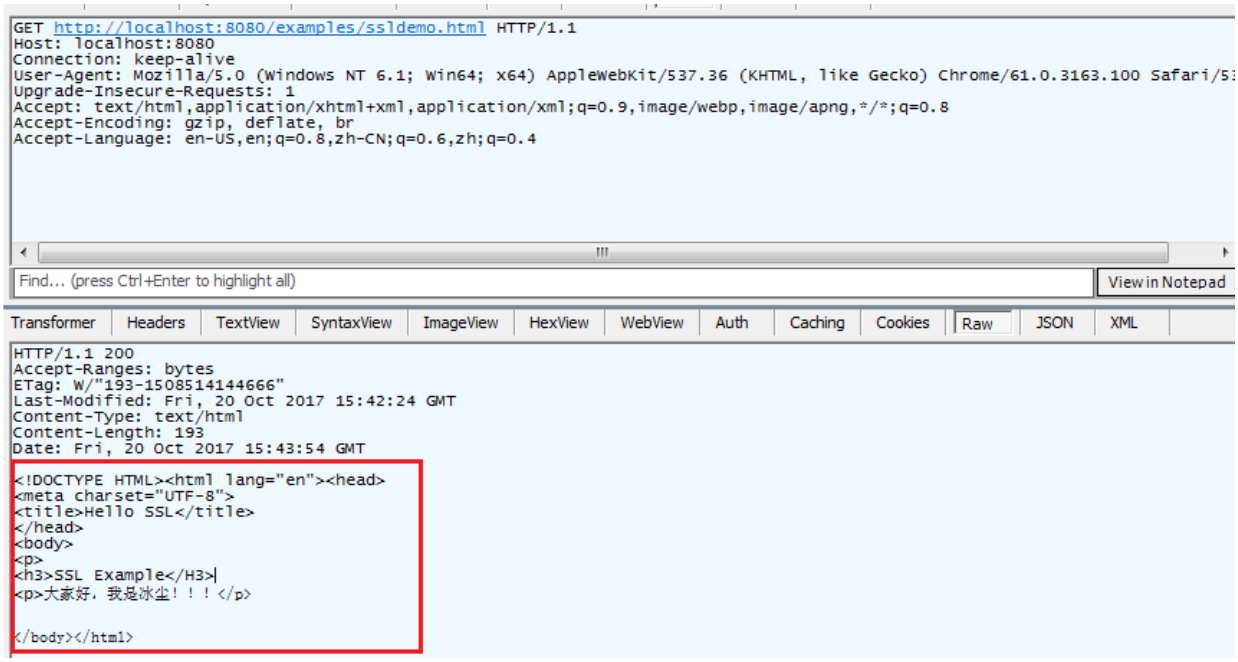
我们可以在https://www.telerik.com/fiddler这个地址下载这个免费的工具并安装。

假设我们已经安装并打开了Fiddler，在浏览器输入下面的地址，我们将看到下面的内容：http://localhost:8080/examples/ssldemo.html

SSL Example

大家好，我是冰尘！！！

这个时候Fiddler监测到的内容如下：



根据Fiddler截获的信息，我们知道在客户端和服务器传输的信息完全是明文，如果有不怀好意的人或者黑客，截取了浏览器和这个站点之间的所有的http请求的话，其上面传输的信息，所见即所得，被一览无余。在浏览器和Web服务器之间，如果传输只是一般的文本信息，比如CSDN的博客等，问题也不大。但是如果传输的是银行账号和密码呢？如果传输的是身份证信息或者信用卡信息呢？如果没有一种很好的机制去保护的话，如果被监听到了，后果将是灾难性的。那如何破这个局呢？其中一种很常见的方式，就是搭建一个基于SSL的Web服务器，让所有的信息通过https协议来传输。

下面笔者来配置一下Tomcat服务器，让其来支持https,因为本chat更偏向于实战，所以笔者列出了其具体步骤。

1. 创建一个SSL的证书

假设我们机器上已经安装了JDK并配置了JAVA Home，然后打开命令行，确保keytool命令能用，如果不用请到JDK的安装目录下bin子目录寻找。在命令行中输入下面的命令：

```
keytool -genkey -alias tomcat -keyalg RSA -keystore tomcat.jks
```

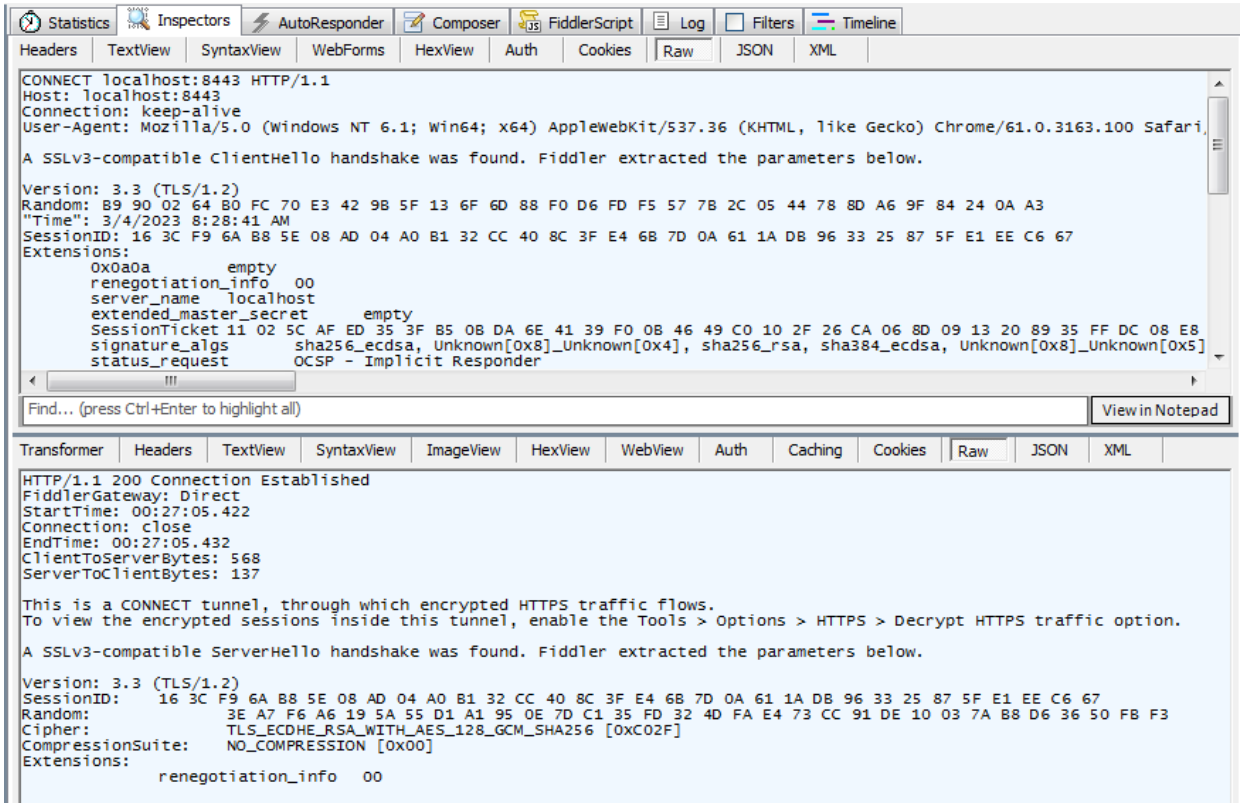
在Tomcat的安装目录下打开server.xml文件，笔者的电脑上是，D:\apache-tomcat-9.0.1\conf\server.xml文件，然后编辑这个文件。把已有的8080端口的Connector注释起来，然后加入下面新的8443端口的Connector。

文本形式的配置如下：

```
    <Connector
               protocol="org.apache.coyote.http11.Http11NioProtocol"
               port="8443" maxThreads="200"
               scheme="https" secure="true" SSLEnabled="true"
               keystoreFile="D:/apache-tomcat-9.0.1/conf/tomcat.jks"
keystorePass="changeit"
               clientAuth="false" sslProtocol="TLS"/>
```


4. 重启启动Tomcat服务器

在浏览器输入，https://localhost:8443/examples/ssldemo.html。这个时候，我们发现其走的https协议，也就是用SSL保护了的http协议。这个时候在回头来看Fiddler监听的信息的结果。



从上面的结果来看，我们的Fiddler监听到的已经是加过密的信息，看了这些信息，压根就不知道Web服务器向浏览器传输了什么信息，因为其是密文的，只有知道其秘钥的工具才能把其破解成明文。这样妈妈再也不同担心我们在网上传输个人的信息了。

3. 什么是对称加密和非对称加密？

说起SSL证书，就不得不提对称加密和非对称加密。下面分别阐述。

3.1 对称加密

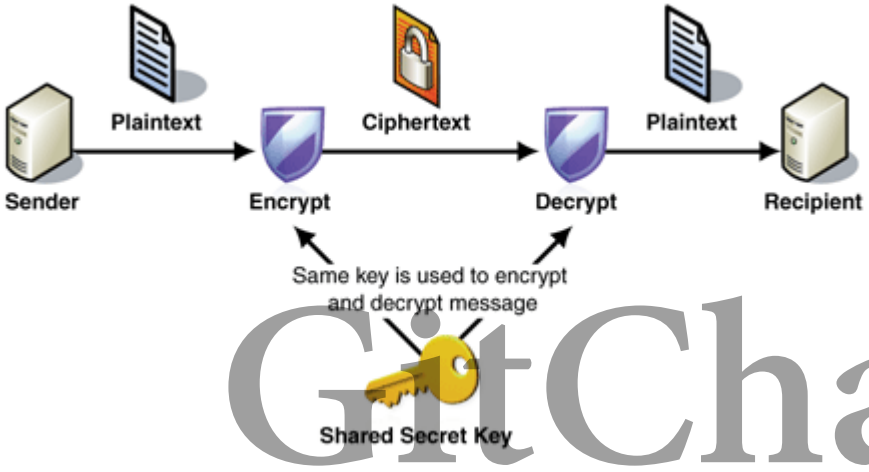
对称加密算法（Symmetric encryption）是一种计算机加密技术，使用加密密钥来伪装信息。它的数据转换使用了一个数学算法和一个私有密钥，这导致无法从加密后的消息中

的d，还原成a； e还原成b，f还原成c。 因为加密解密用的同一份秘钥，加密和解密通过秘钥完全对称的，所以叫做对称加密。

目前比较通用和常见的对称加密算法主要有下面的种类：

- AES
- Blowfish
- DES
- 三重 DES
- Serpent
- Twofish

千言万语抵不过一幅图，下面笔者从网上找了张经典的图，大家一看图就应该明白了什么时候对称加密算法。



3.2 非对称加密

既然对称加密算法这么强大，那为什么还要使用非对称加密算法呢？那什么又是非对称加密算法呢？

因为对称加密算法有一个最不安全的地方，就是秘钥的分发；如何保证秘钥能从A处共享给B的时候，特别是复杂和万能的互联网上，如何保证在交换秘钥的时候不被窃听到？

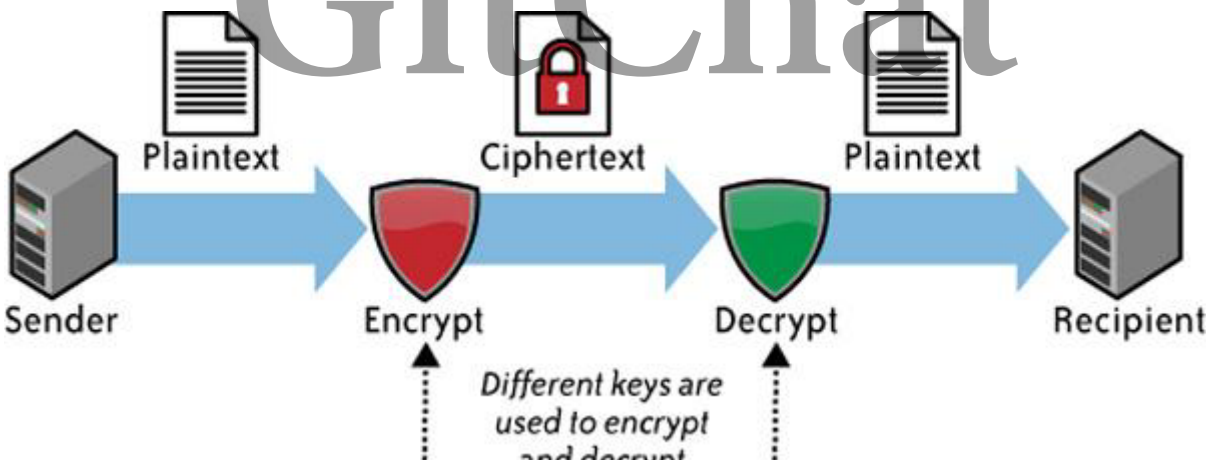
- 首先A把自己的公钥发送给B，B也把自己的公钥发送给A
- A把要发送给B的信息，用B的公钥进行加密，然后发送给B，因为只有B有B公钥所对应的私钥B，所以即使被传输的信息被第三方监听到，也没有关系，因为只要私钥B才能解密。
- B收到从A发送过来的用公钥B加密的信息后，用自己独一份的私钥B把信息解密；然后根据A的信息，知道某一个重要的秘密，为了对A进行回复， B就会用A给公钥把需要回复的信息加密发送给A
- 因为只有A有A公钥所对应的私钥A，所以即使被传输的信息被第三方监听到，也没有关系，因为只要私钥A才能解密。

是不是整个通信就非常安全了。因为任何第三方即使在公钥A和公钥B相互交换的过程中知道到了公钥A和公钥B，也没有用，因为私钥A和私钥B没有相互交换，只有各自自己知道。从而保证了通信双方通信的安全。

目前比较通用和常见的非对称加密算法主要有下面的种类：

- RSA
- EIGramal
- Rabin
- 椭圆曲线密码

千言万语抵不过一幅图，下面笔者从网上找了张经典的图，大家一看图就应该明白了什么时候非对称加密算法.

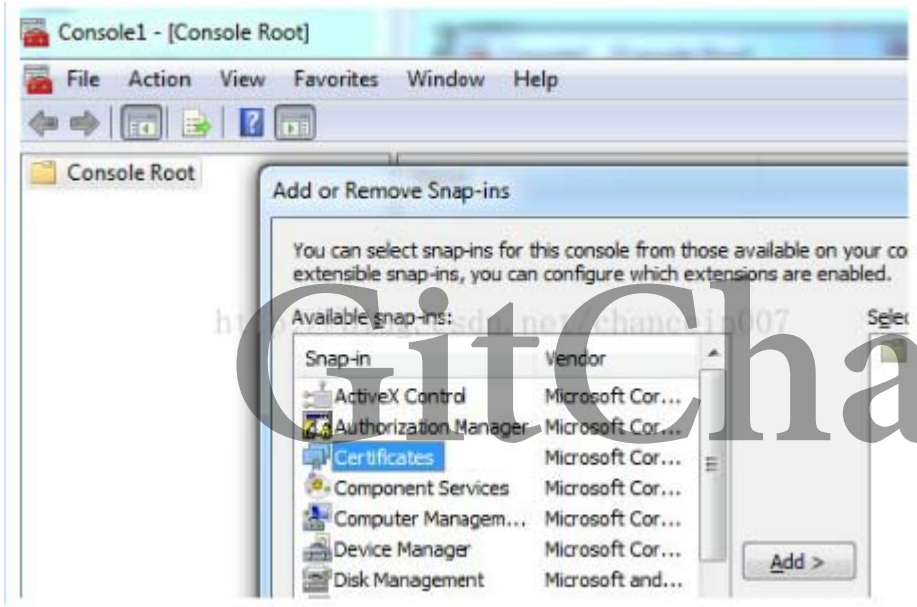


身也提供了一个存储证书和管理证书的工具，就是mmc工具。操作的简要步骤如下：

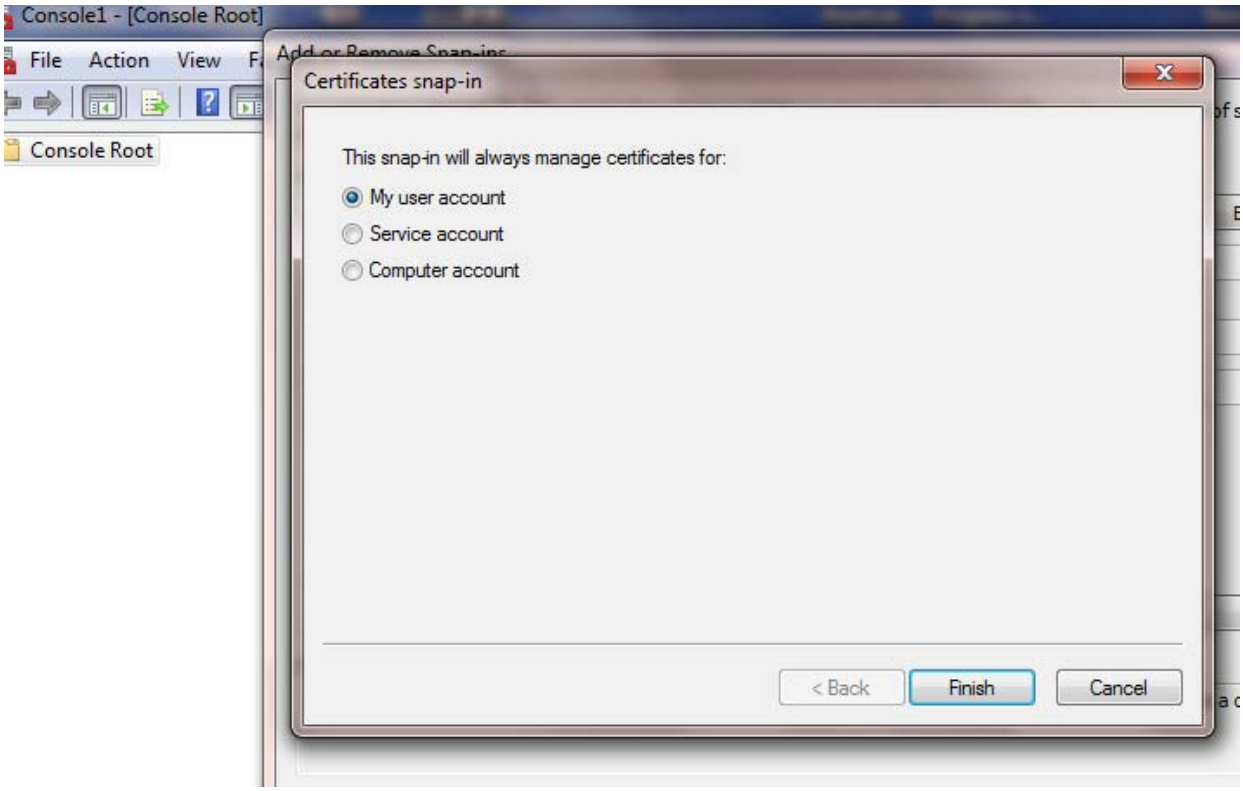
Step 1. 在run命令中输入mmc或者在cmd的命令行中输入mmc

Step 2 将会打开一个mmc的管理台

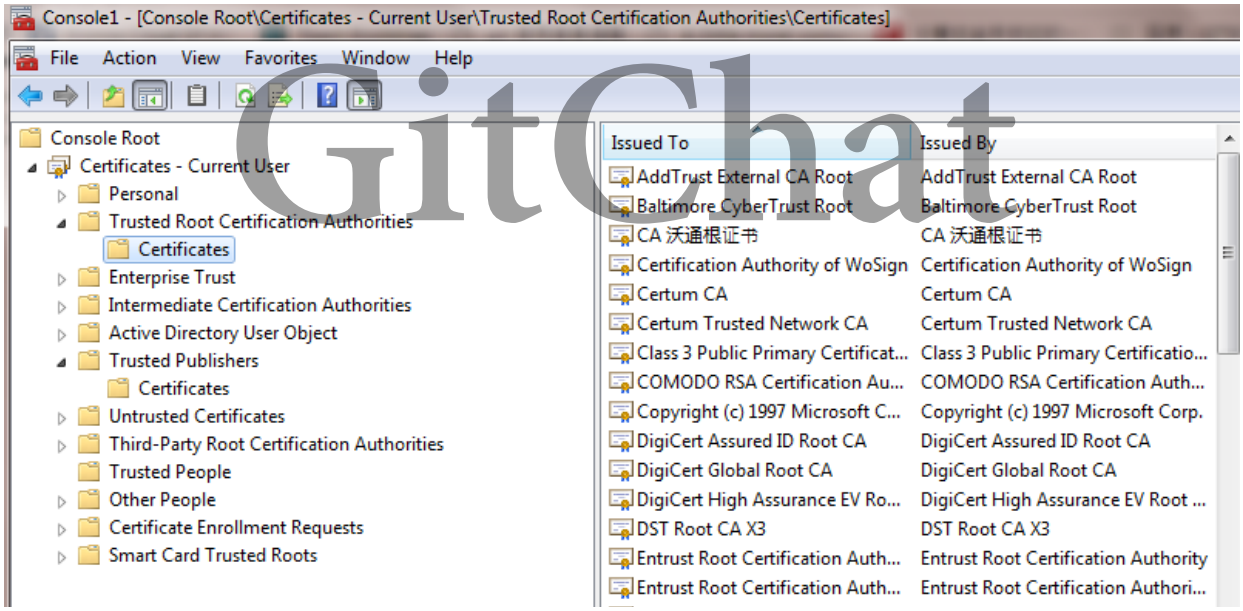
Step 3 在File菜单中选择File->Add or Remove Snap-ins选项，将会打开下面界面，我们选择并添加”Certificates”选项。

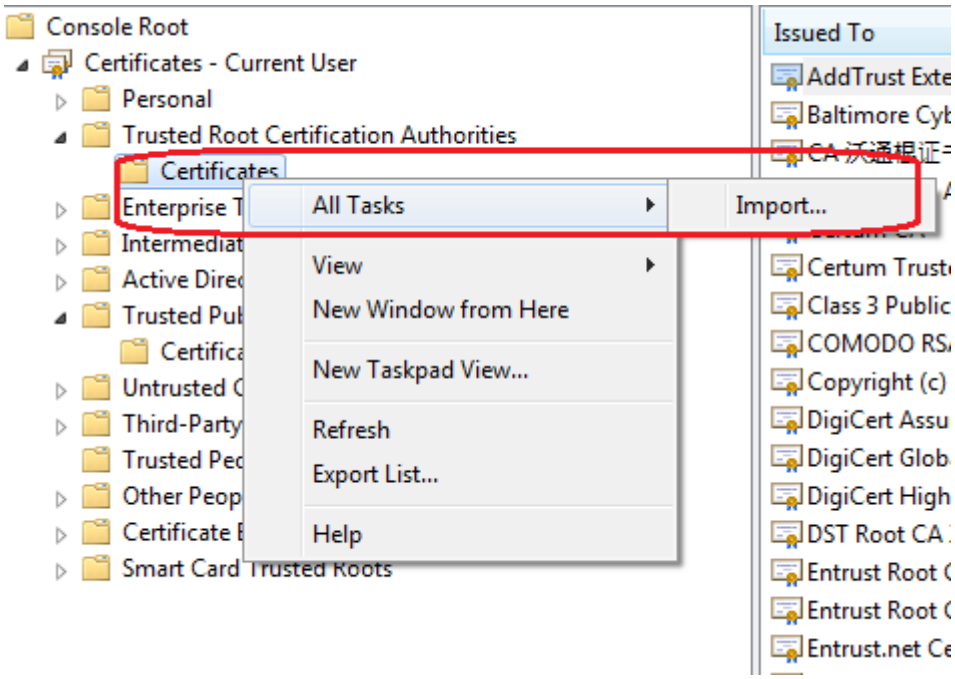


Step4 选择一个账号类型，一般选择My User account即可。

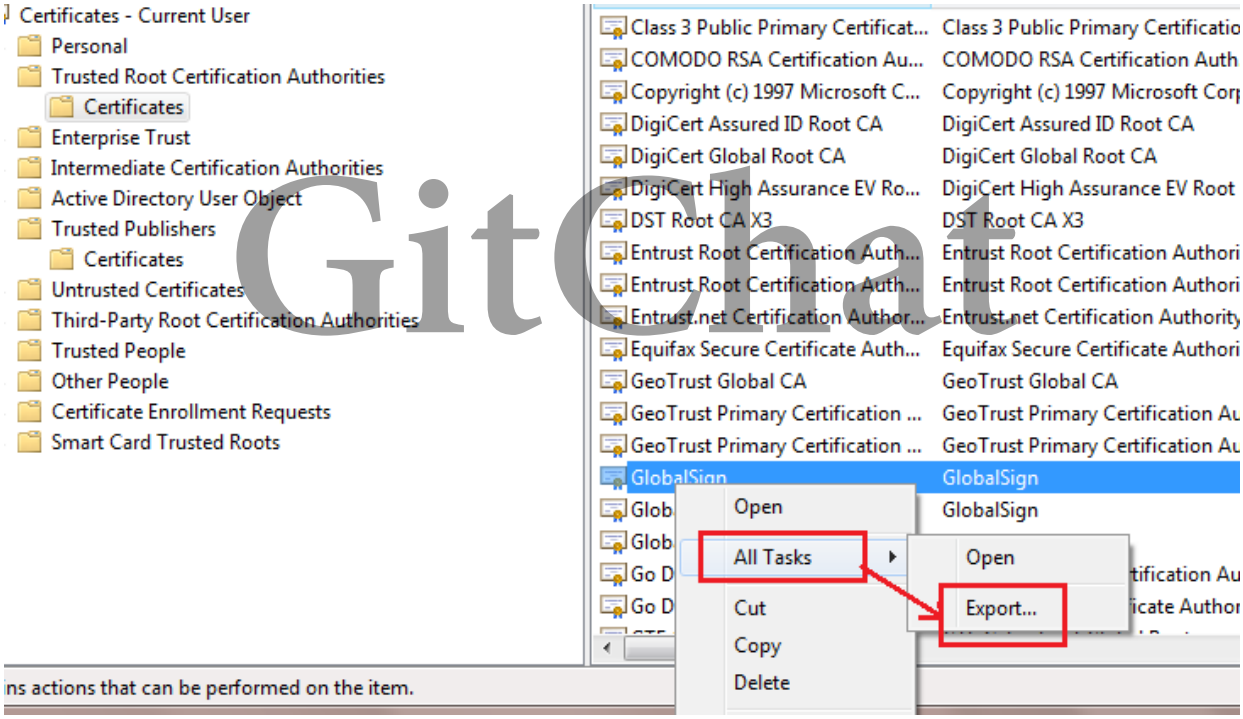


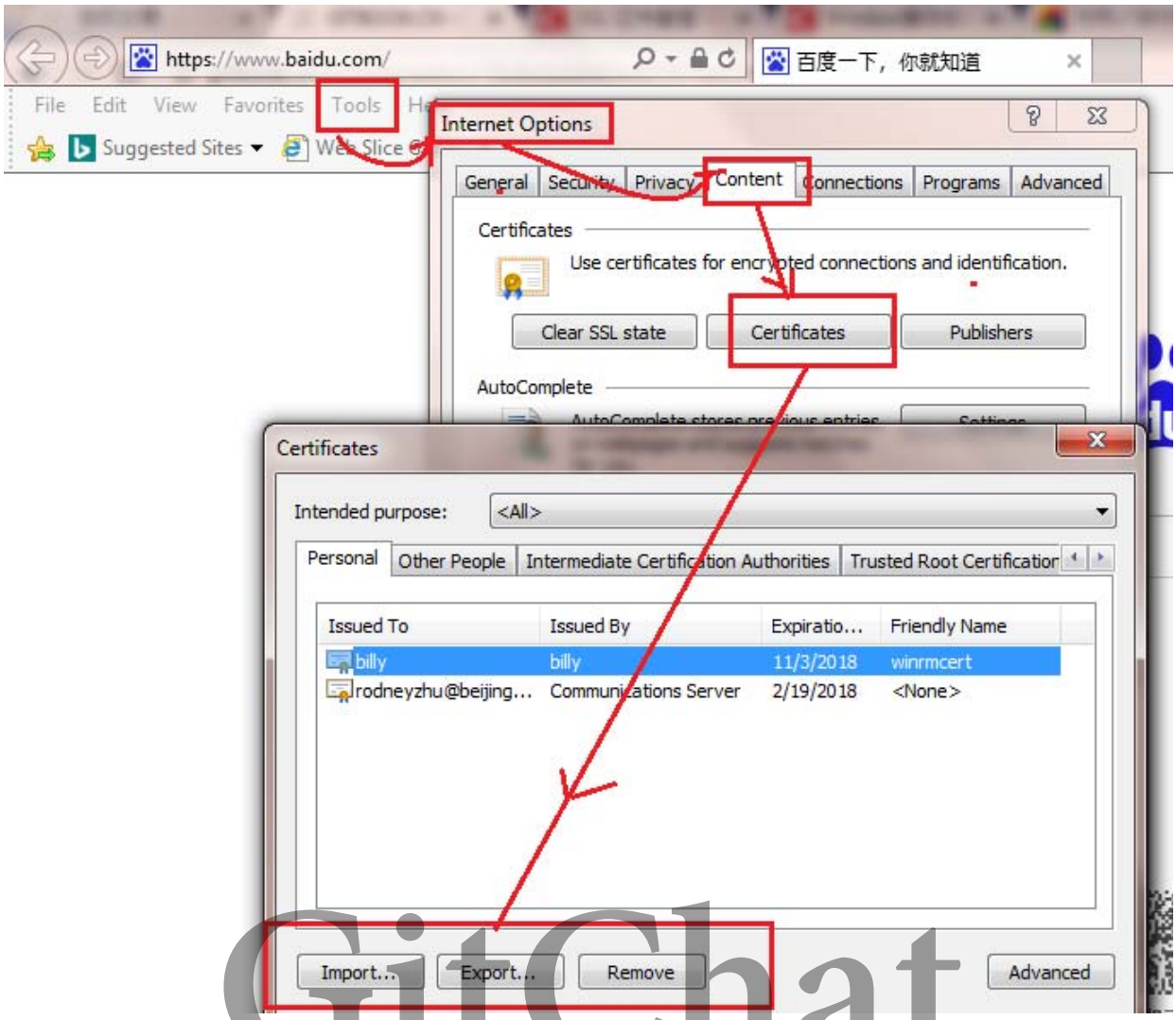
Step 5.这样就能把所属的账号类型下面的证书列举出来了。





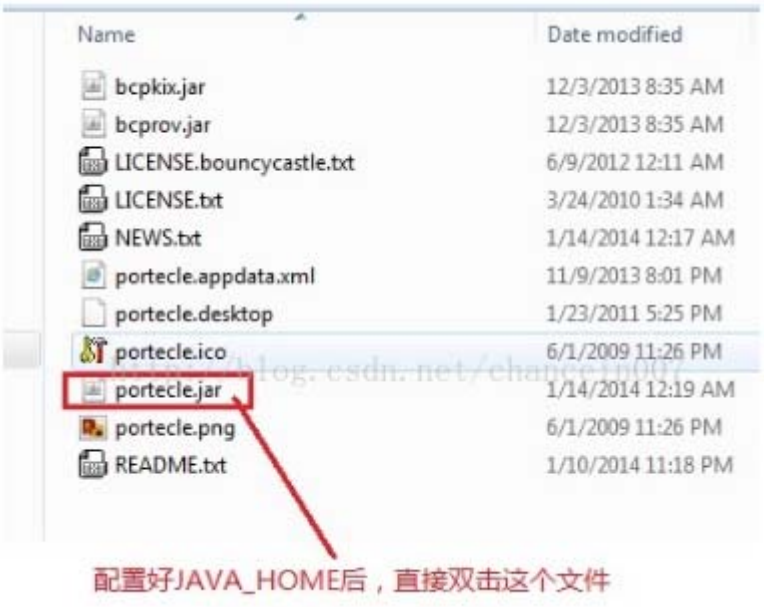
Step 7. 可以通过下图的方式导出证书（以根证书为例子）





5. 如何从 HTTPS 网站导出 CA 证书

我们在访问https的时候，对于直接用Java，.NET, Python等语言开发的程序的时候，我们需要调用并访问HTTPS网站的时候，需要提供访问网站的CA证书，这个时候客户端才能成功访问系统网站，那么有什么好的方法和工具能够帮助我们快速的从我们需要访问的网站上把CA证书导出来呢？

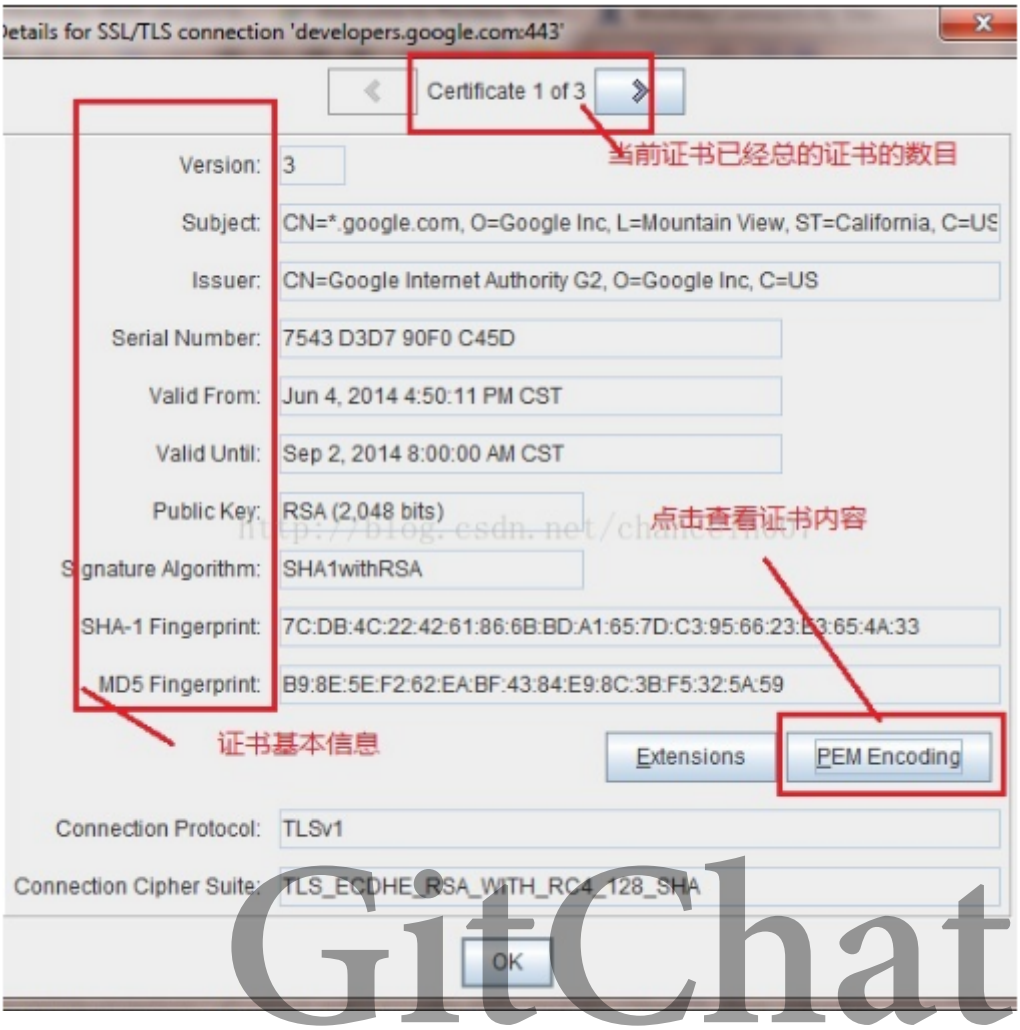


Step3. 安装JDK后，在环境变量里面配置JAVA_HOME（具体步骤，请在Google或者百度里面搜索“java home 设置”）因为不是本节的重点。

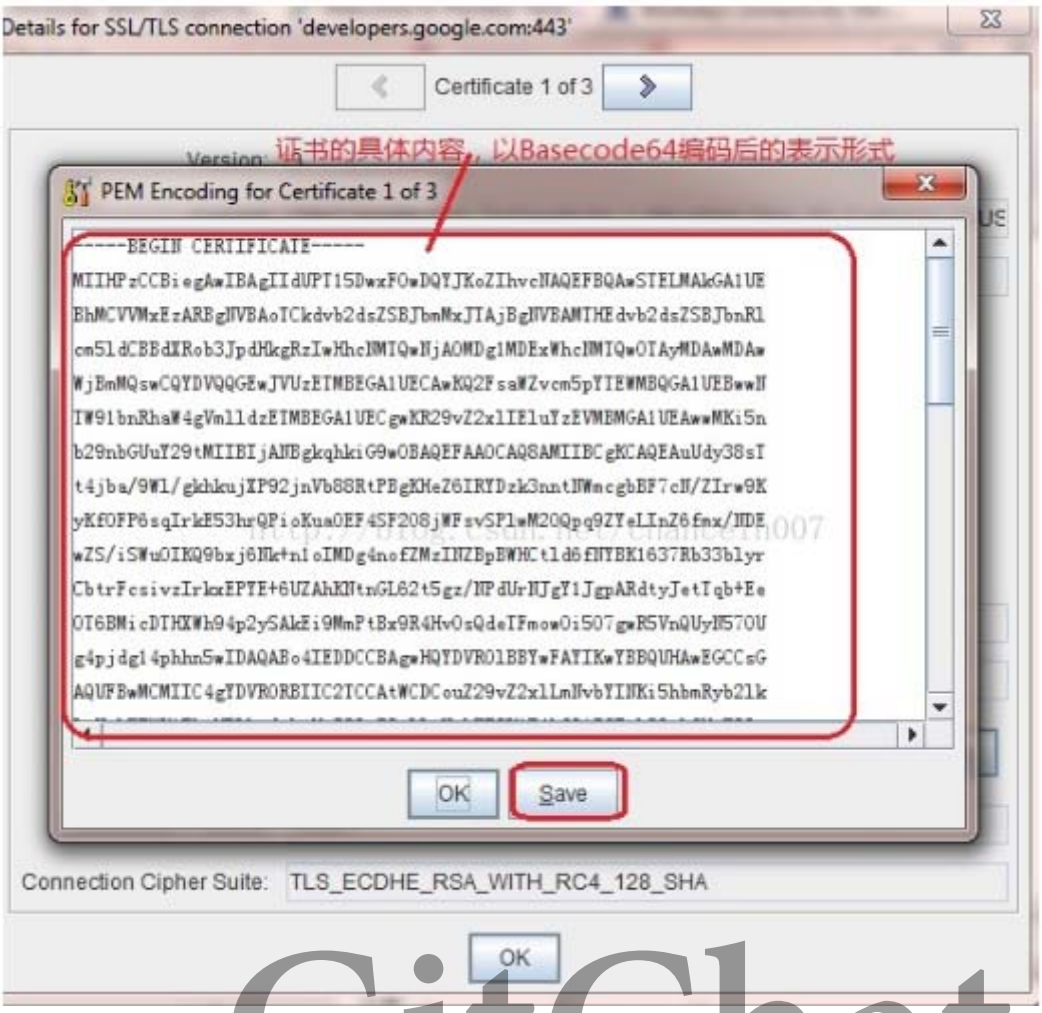
配置完后，双击portecle.jar 文件，如上图，将会弹出下面的界面。

Step4. 点击下面的菜单项，弹出“Examine SSL/TLS Connection” 窗体

Step6. 在弹出的证书详情的窗体中，我们可以看到下面的界面，见下图



Step 7 . 在Step6 附图中的“PEM Encoding”按钮，将会弹出下面的窗体。



Step 8. 点击“Save”按钮，保存证书。

Step 9. 点击Step6页面顶端的右向箭头（->），重复步骤七到八，把剩下的两个证书也导出来。至此，恭喜各位看官，你的HTTPS的SSL的CA证书导出成功了。

6. 开发一个 Eclipse 插件把证书导入到 Java 的 keystore

我们在开发或者使用SSL的过程中，很多的软件需要我们提供java的keystore，特别是一些基于Java的中间件产品。我们常规的做法是JDK自带的工具命令（keytool）去做，比如，下面的例子，

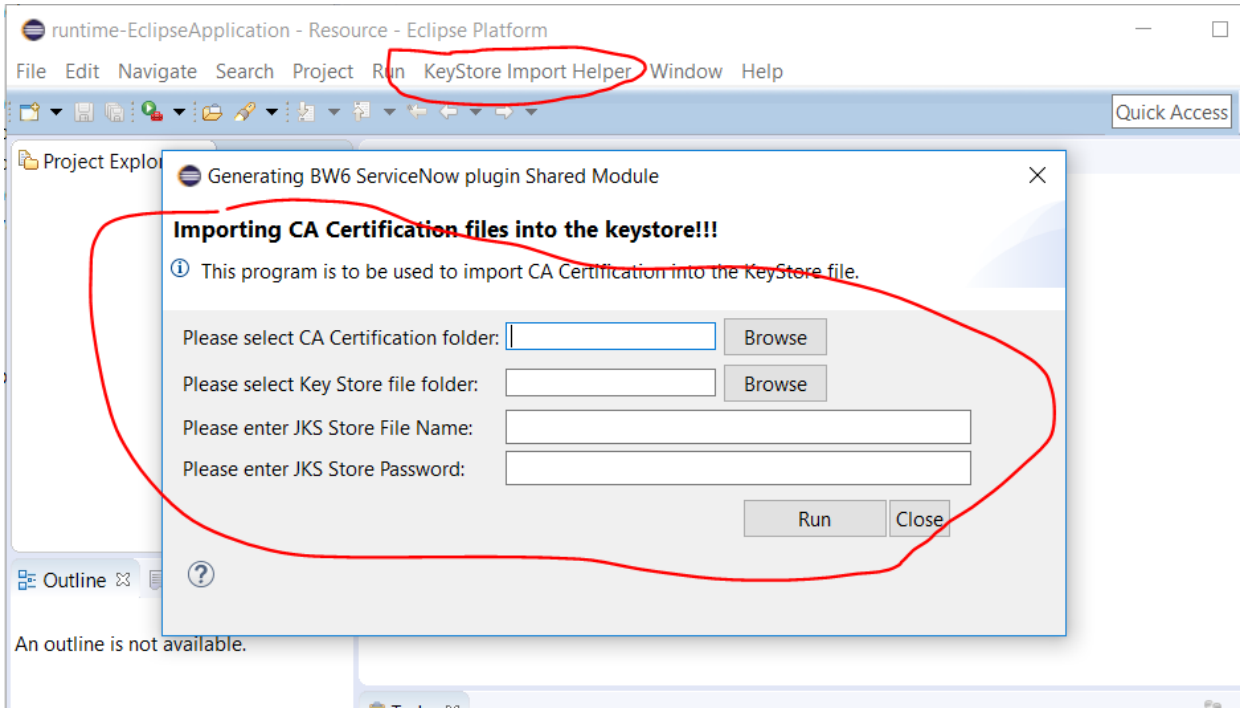
```
keytool -import -v -alias EnTrust2048 -file
D:\certs\EnTrust2048.cer -keystore D:\certs\test.jks
keytool -import -v -alias EntrustCertificationAuthorityL1C -file
D:\certs\EntrustCertificationAuthorityL1C.cer -keystore
D:\certs\test.jks
keytool -import -v -alias test.com -file D:\certs\Service-
now.com.cer -keystore D:\certs\test.jks
```

但是这种方式比较繁琐，假设我们一个文件夹下面有100个SSL的证书，那么我们就要输入100个类似于上面的命令。如果是文件夹里面套文件夹里面还有证书，就更麻烦。那么有没有好的办法呢？笔者就跟大家分享一下如何用java的程序代码去实现。

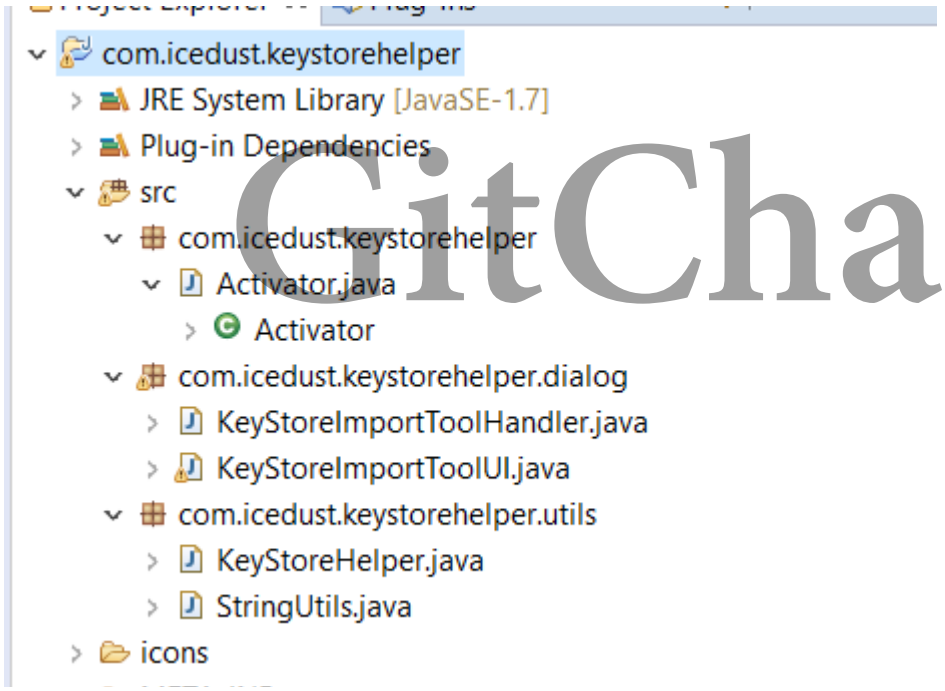

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.security.KeyStore;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.List;
import javax.naming.ldap.LdapName;
import javax.naming.ldap.Rdn;
```

```
File(originalTrustFolder);
                File[] certs = trustedFolder.listFiles();
                if (certs != null) {
                for (File cert : certs) {
                CertificateFactory factory =
CertificateFactory.getInstance("X.509");
                try {
                X509Certificate certificate = (X509Certificate)
factory.generateCertificate(new FileInputStream(cert));
                X500Principal principal =
certificate.getSubjectX500Principal();
                LdapName ldapDN = new
LdapName(principal.getName());
                List<Rdn> rdns = ldapDN.getRdns();
                    for (Rdn rdn : rdns) {
                    String type = rdn.getType();
                    if (type.equals("CN")) {
                      keystore.setCertificateEntry((String)
rdn.getValue(),certificate);
                                    break;
                    }
                                }
                } catch (Exception ex) {
                    continue;
                }
                }
                }
                FileOutputStream fos = new
FileOutputStream(jksTrustStoreLocation);
                keystore.store(fos, password.toCharArray());
                fos.close();
            } catch (Exception exp) {
            }
        }
    }

    /**
     * @param args
```



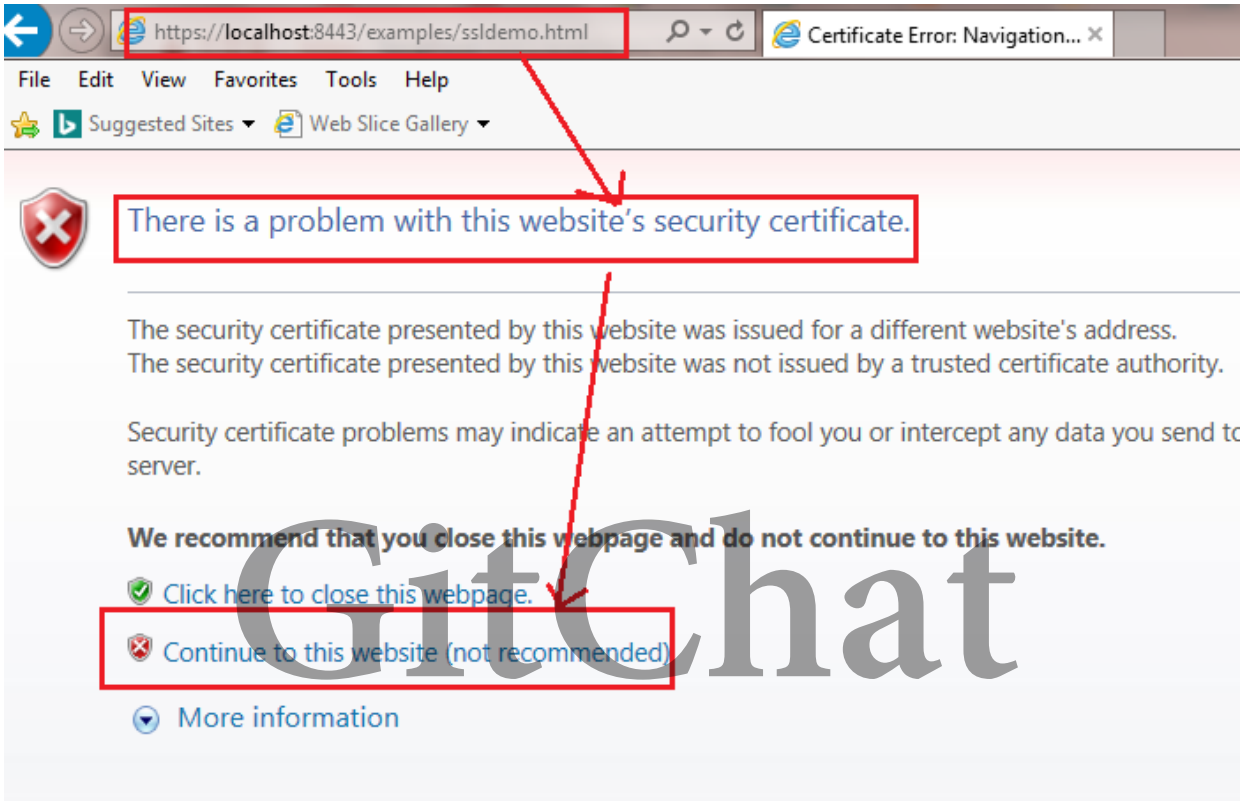
程序代码结构如下：



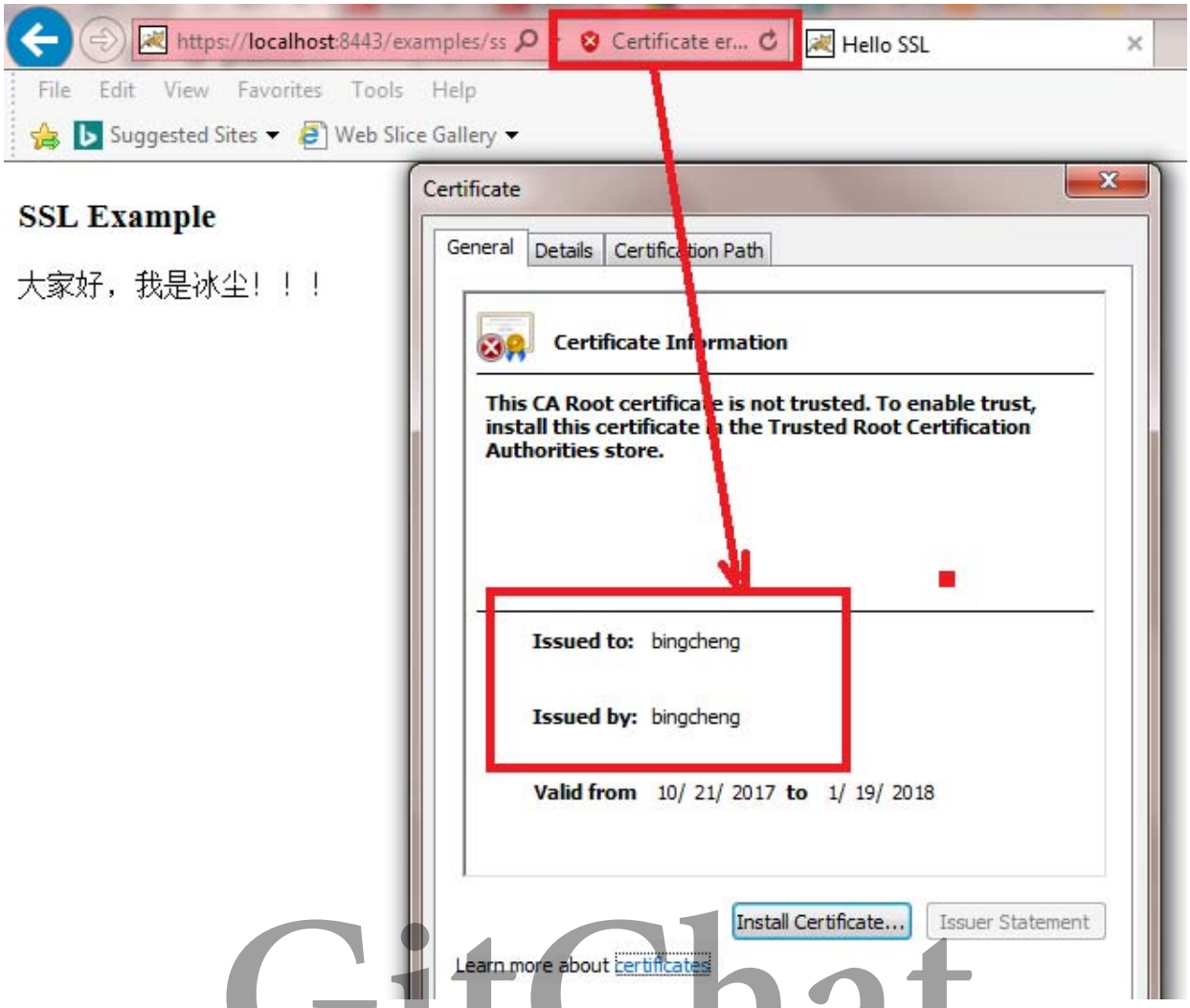
7. 自签名证书 VS CA 证书

所谓的自签名证书就是自己给自己签发证书；在本文的第2个章节，笔者曾经自己在Tomcat上搭建了一个https的网站，这个https的网站就是用的是自签名的证书，假如用IE浏览器打开上面的这个地址，我们将会看到其证书的情况：https://localhost:8443/examples/ssldemo.html。

第一次打开的时候，其会提醒我这是一个不安全的网站：

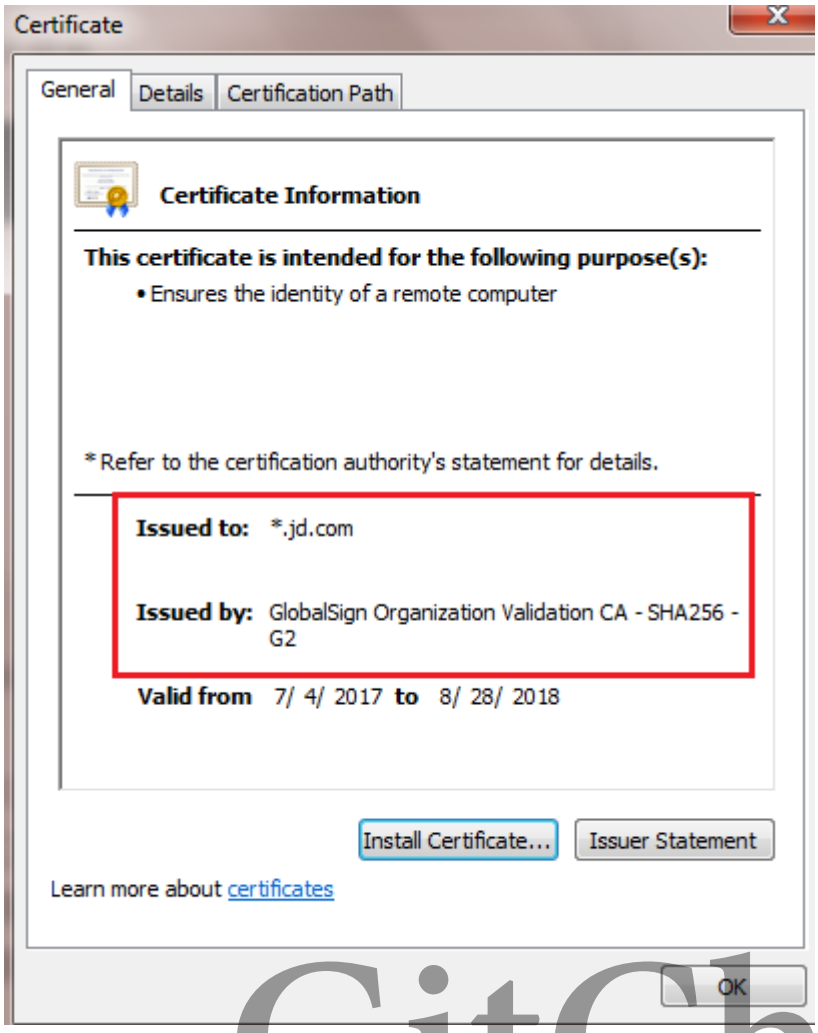


如果我们进一步点击”Continue to this website (not recommended). “, 然后浏览器才会真正的进入这个页面。接着在浏览器网址的上方，我们进一步点击查看证书，就会发现，这个证书是bingcheng老师自己颁发给自己的。



那为什么会有这个提示呢？因为上面网站的SSL证书，是我自己用工具生成的，没有被第三方的权威机构或者公司的CA签署，所以浏览器认为是不安全的网站，所以弹出这个页面让用户引起注意。那么什么是CA证书呢？

CA证书就是通过第三方有资质的权威的公司或者机构，先把自己的证书请求发送过去，然后第三方的权威CA就把发过来的证书请求，通过他们的CA的根证书或者二级三级等下级代理证书把发送过来的证书请求标注上颁发机构；当然在提交证书申请的时候，需要提供申请方的网站或者公司的相关证明资料，证明这个网站的证书就是给这个网站本身用的，而不是一个仿冒的公司。目前比较权威的第三方CA一般都自动集成在了浏览器里面，目前世界上最知名的8家SSL证书CA供应商如下：



从上图可以看出，京东的网站不是一个自签名的证书，而是有第三方的权威CA认证公司GlobalSign颁发和签署的。

那么我们在工作当中，有的时候老板会让我们去生成一个SSL证书的请求，然后让第三方的CA帮忙签署，那这个时候，我们应该如何去做呢？生成SSL证书请求有很多的方法，笔者是做JAVA开发出身的，就介绍一种最为简单的方法，就是直接利用JDK自带的工具去生成一个证书请求。假设我们要对一个www.51talkdocter.com 申请一个由权威CA颁发的SSL证，我们就需要执行下面的命令：

```
C:\Users\bingchang>keytool -genkey -alias 51talkdocter -keyalg
```

```
  [Unknown]:  cn
Is CN=www.51talkdocter.com, OU=51talkdocter, O=test, L=beijing,
ST=beijing, C=cn
 correct?
  [no]:  yes
```

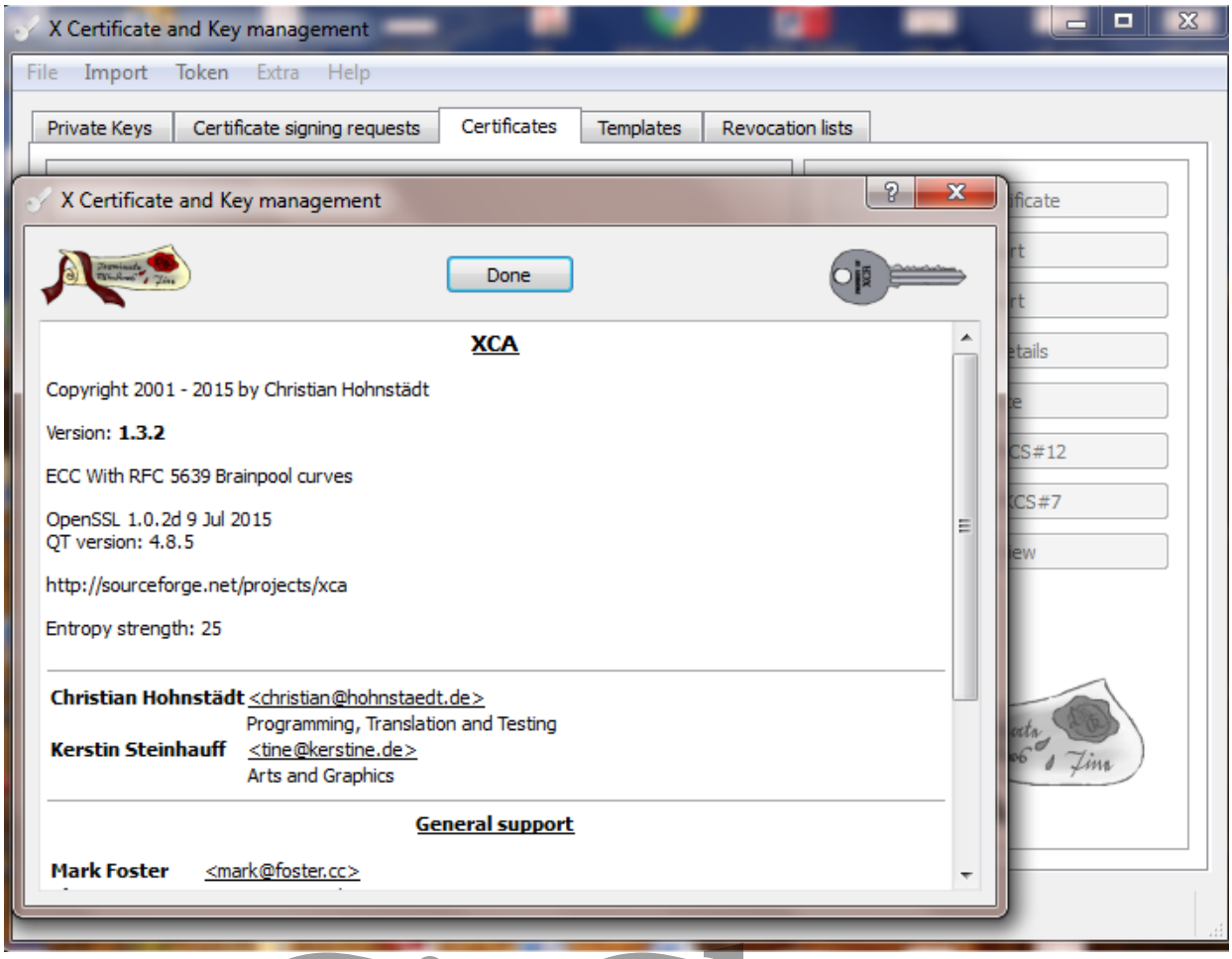
上面的命令会生成一个私钥，然后利用这个私钥去生成一个证书请求。命令如下并输入上个命令中曾经设置过的密码：

```
keytool -certreq -keyalg RSA -alias 51talkdocter -file
www.51talkdocter.com.csr -keystore www.51talkdocter.com.jks
```

执行完上面的命令后，就会生成 www.51talkdocter.com.csr 一个CA证书签署申请文件，然后把这个文件发送给上面提到任何一家权威的CA机构或者其他的CA机构去签署我们的申请文件，签署完成后，就能正常使用了。

8. 如何通过可视化工具生成和管理公钥和私钥

一般情况下，大家能想到管理和生成SSL证书的方法就是OpenSSL程序或者用JDK自带的keytool命令，但是这两种工具虽然功能强大，但是用户的可操作性并不好，需要用户记住并输入一些命令，而且也不是特别的直观。那么有没有一个好的证书管理工具，能帮我们方便的管理证书，方便的生成证书请求，方便的对证书进行自签名，方便的把证书以一种格式导入进来，然后以自己想要的另外一种格式导出去呢？答案就是，XCA（X Certificate and key management），XCA是一个开源的工具，底层还是基于openSSL的类库和API的。下面是其界面.

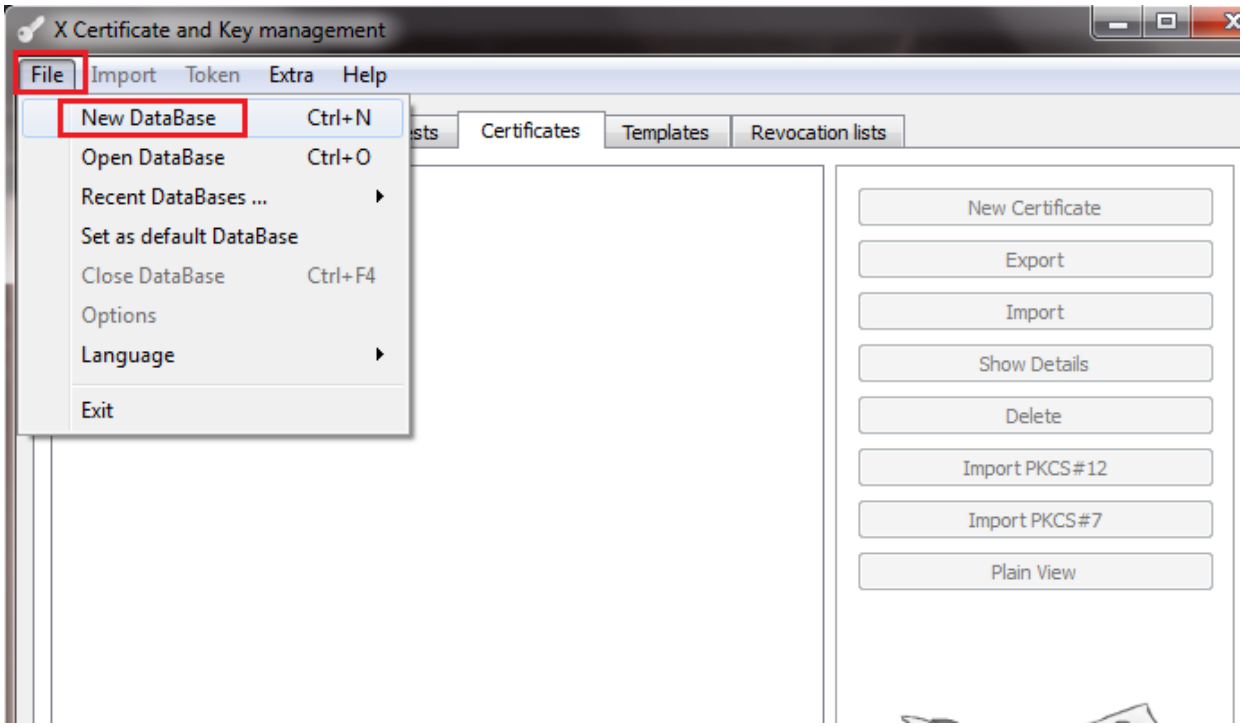


那么如何安装呢？

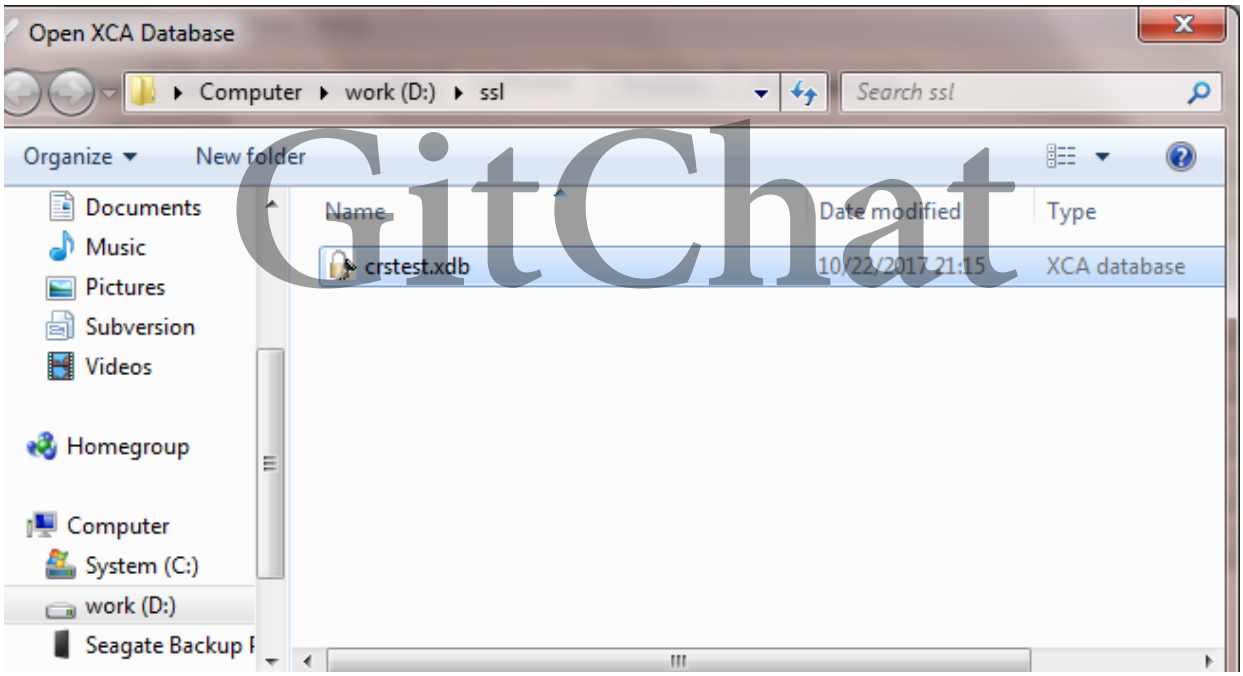
- Step1 . 打开浏览器，输入下载地址
- Step2 . 然后点击“direct link” 或者 “mirror”标签,下载最新的版本，最新的版本是：1.3.2
- Step3. 安装下载后的exe文件
- Step4. 这样在你的开始菜单中，就能找到xca的目录，打开xca的程序即可。

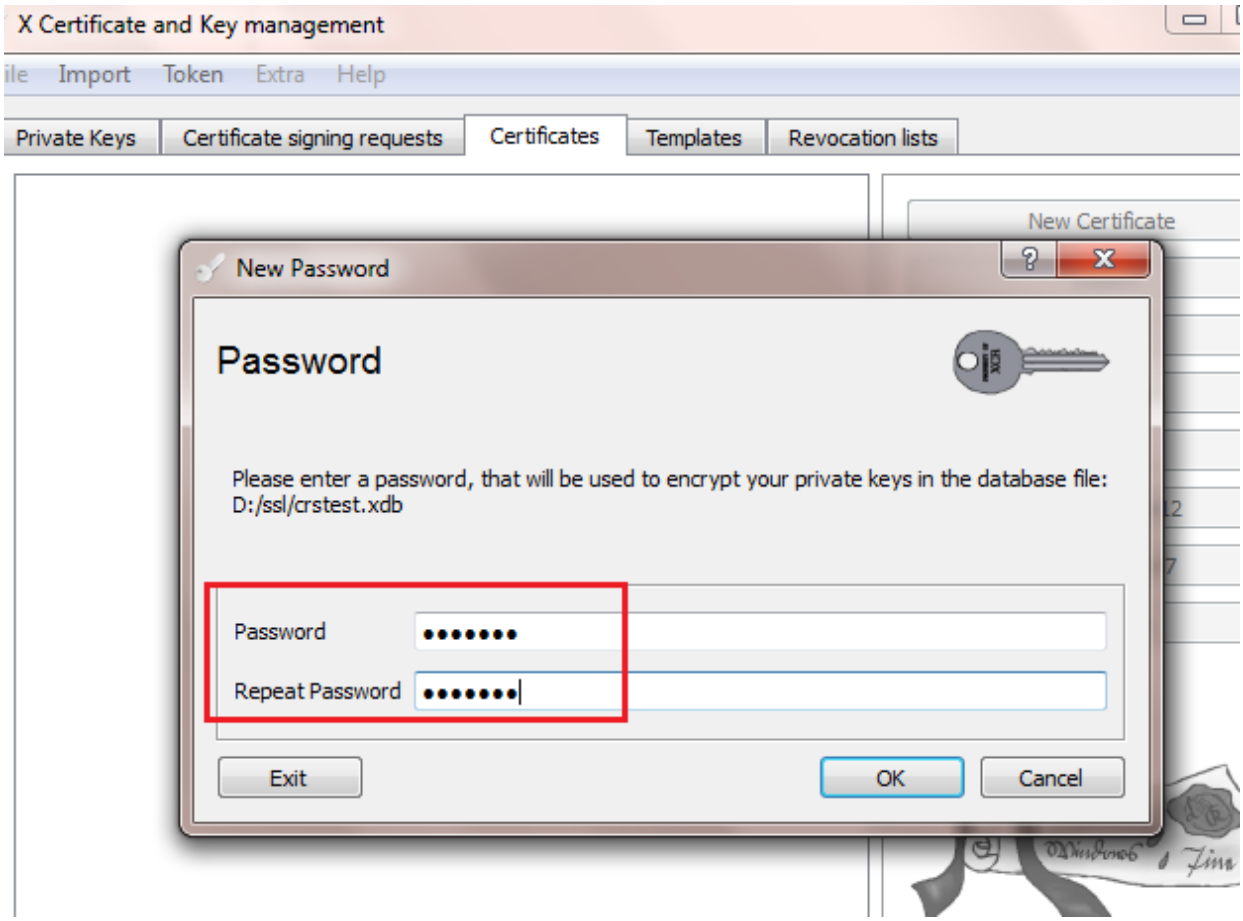
万事俱备了，那我们就开始入门使用了。我们就以上面的第7部分为例子，用XCAwww.51talkdocter.com创建证书请求，演示一下如何通过XCA也能达到同样的目的。

Step1. 打开XCA

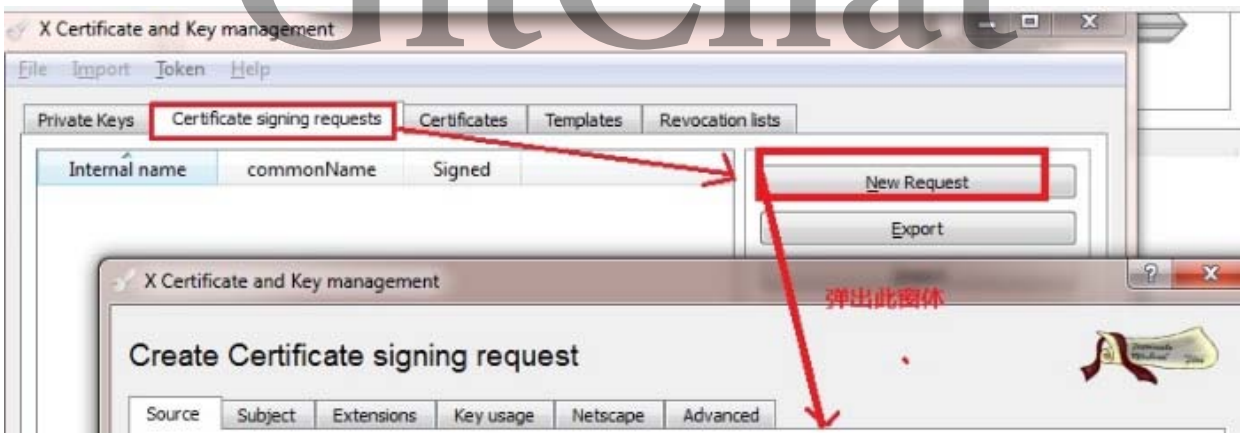


- Step3. 选择证书数据存储的位置并命名存储文件的名字。

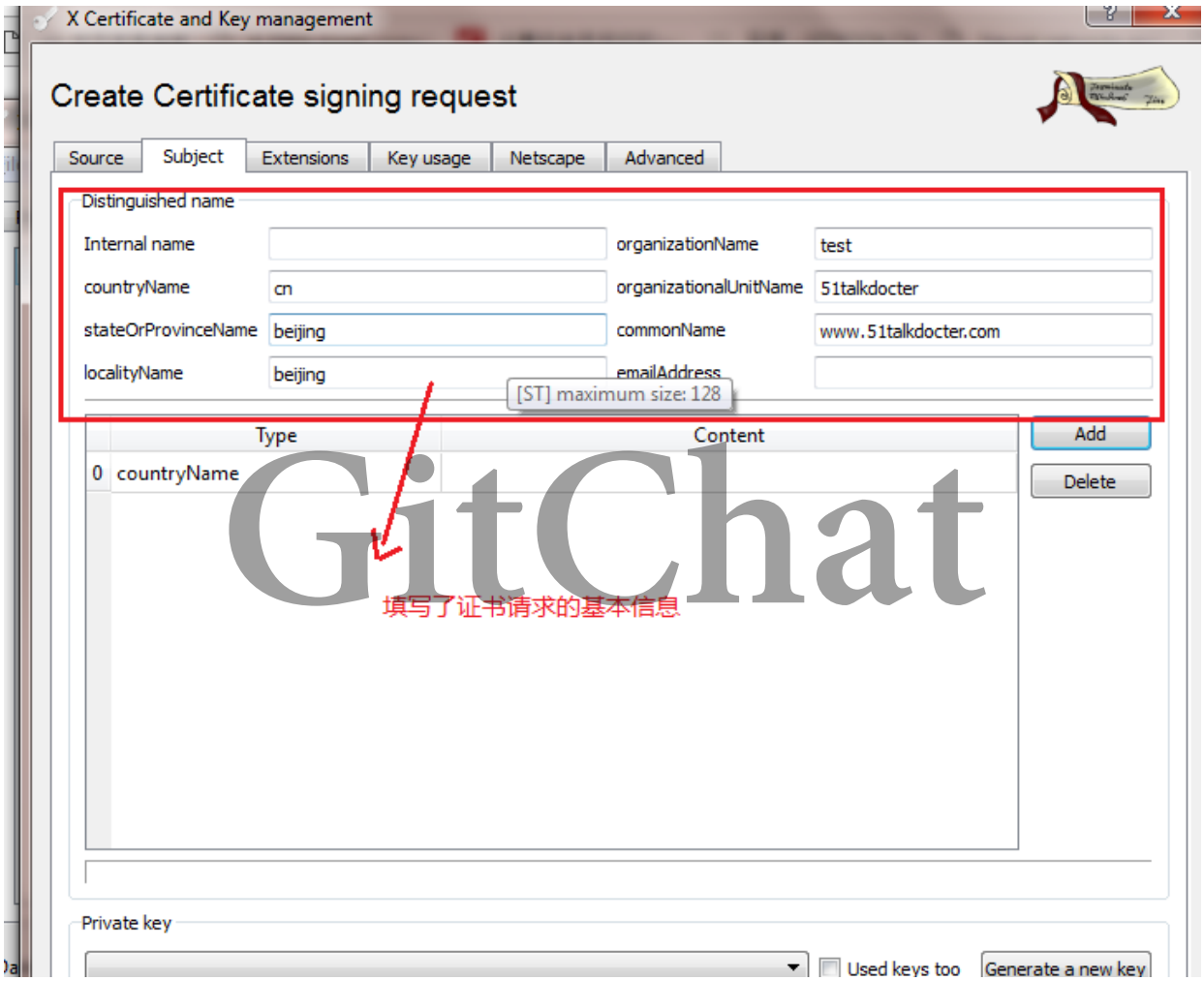




Step5： 鼠标点击到“Certificate Signing Request”分页，并点击右边的“New Request”，将弹出下面的窗体。



Step6 ：“Source”标签页保持不变，用默认的模板即可，点击“Subject”标签页，输入证书的信息，比如内部名，组织名，组织单元名字，国家名，省份以及通用名字。一般内部名和通用名字填一样，而且通用名一般都需要填写，因为这个是用来标示当前证书在你的组织内部的唯一性；一般组织名和组织单元名一般和你的公司和部门相吻合，比如，你属于ibm的信息部（IT部门），那么你的组织名（organizationName）可以填写成：ibm，为了和上面的用jks生成的保持一致，我输入的是test。你的organizationUnitName可以填写成IT，表示的是IT部门，我在这写成了51talkdocter，通用名（common name）和内部 名字取决你证书的用途，主要用来描述你证书的功能和用途的。我的这个例子中，主要是为www.51talkdocter.com域名申请的，所以我写成www.51talkdocter.com。具体信息如下：

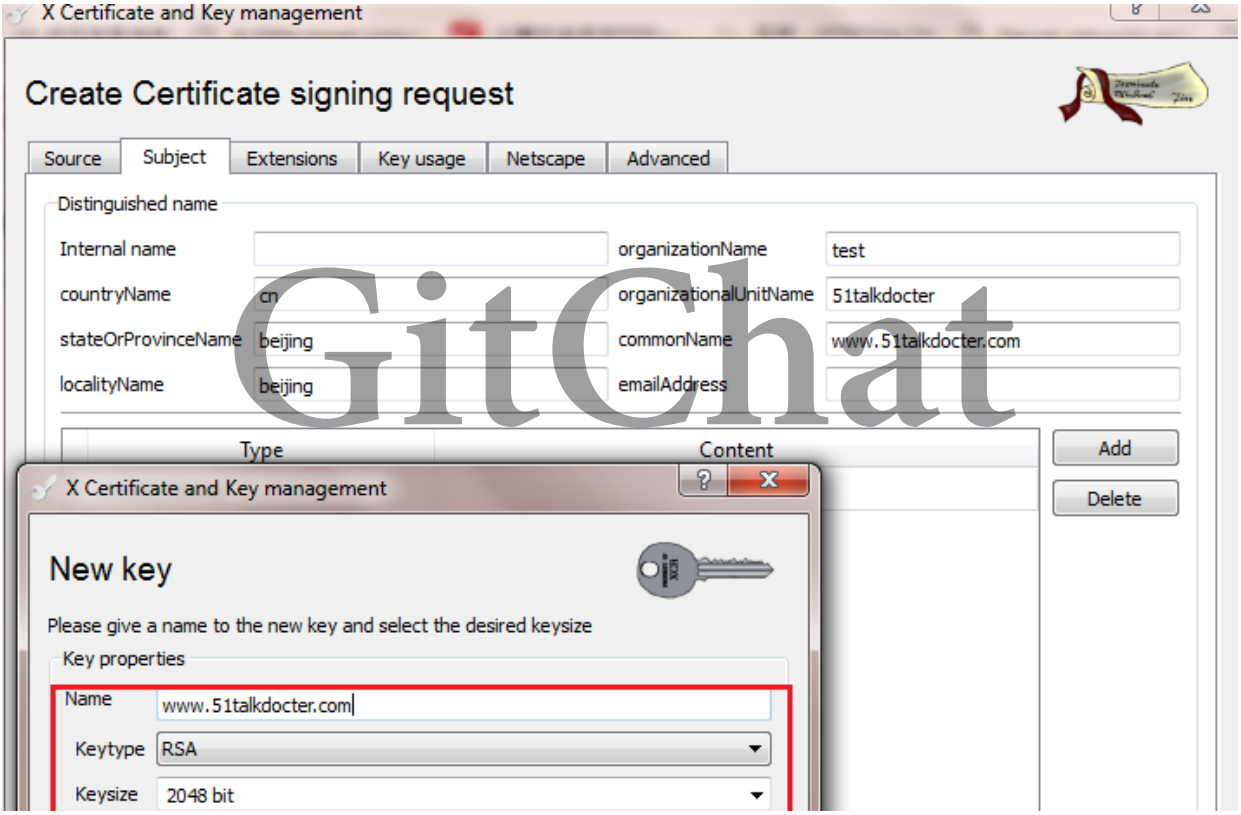


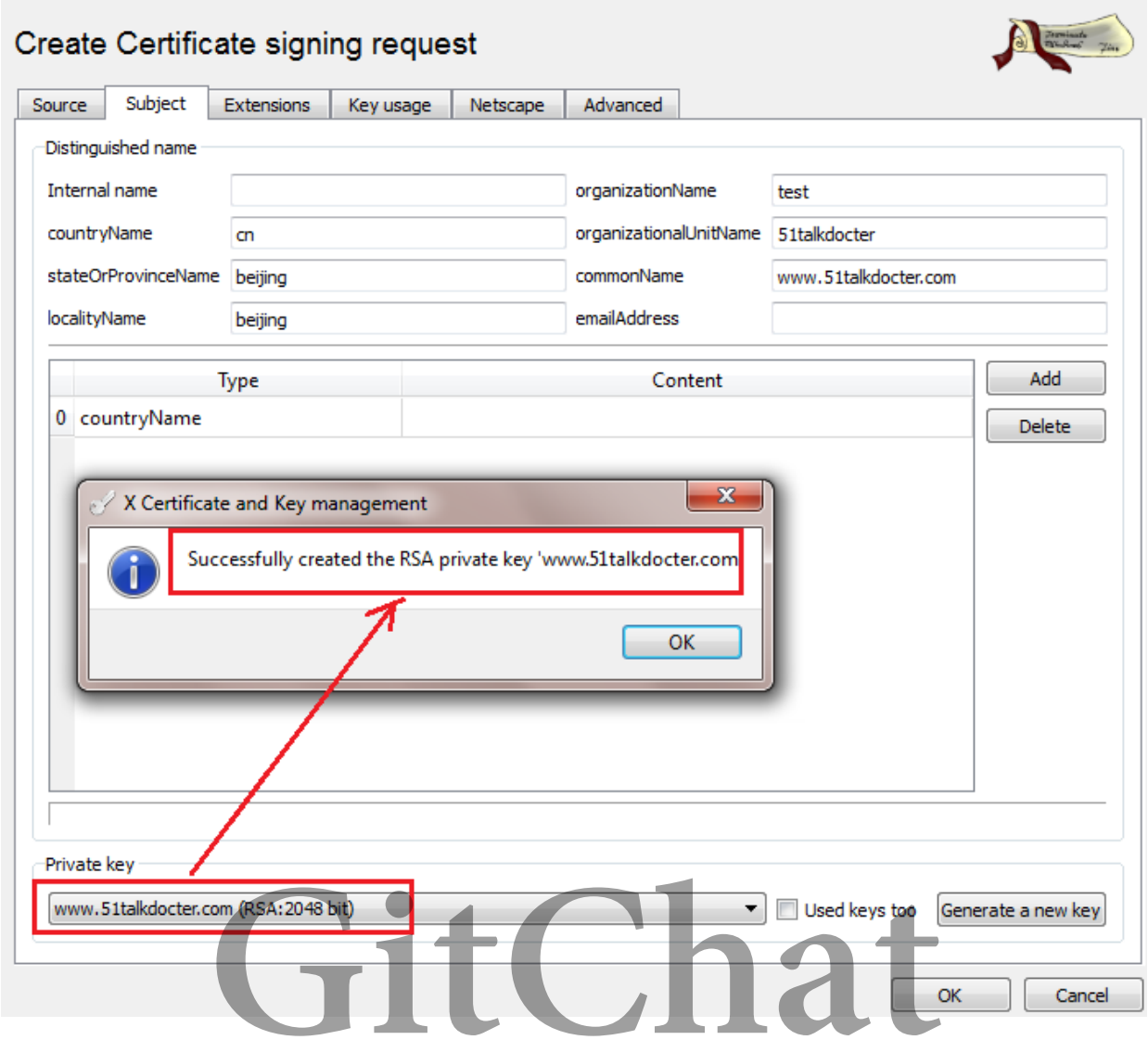
都承认的第三方权威机构办的，这时的公安局就相当于我们SSL证书的颁发机构，当前业界比较权威就是Thawte，Verisign公司等。

（2）如果这个证书只是企业或者组织内部使用，一般用自己组织内部或者个人认可的证书中心去签名即可，这就好比你那着你的劳动合同去公司申请工牌一样，你这个工牌只能在你组织或者公司内部使用，你不能拿你的工牌去做火车，道理一样的。

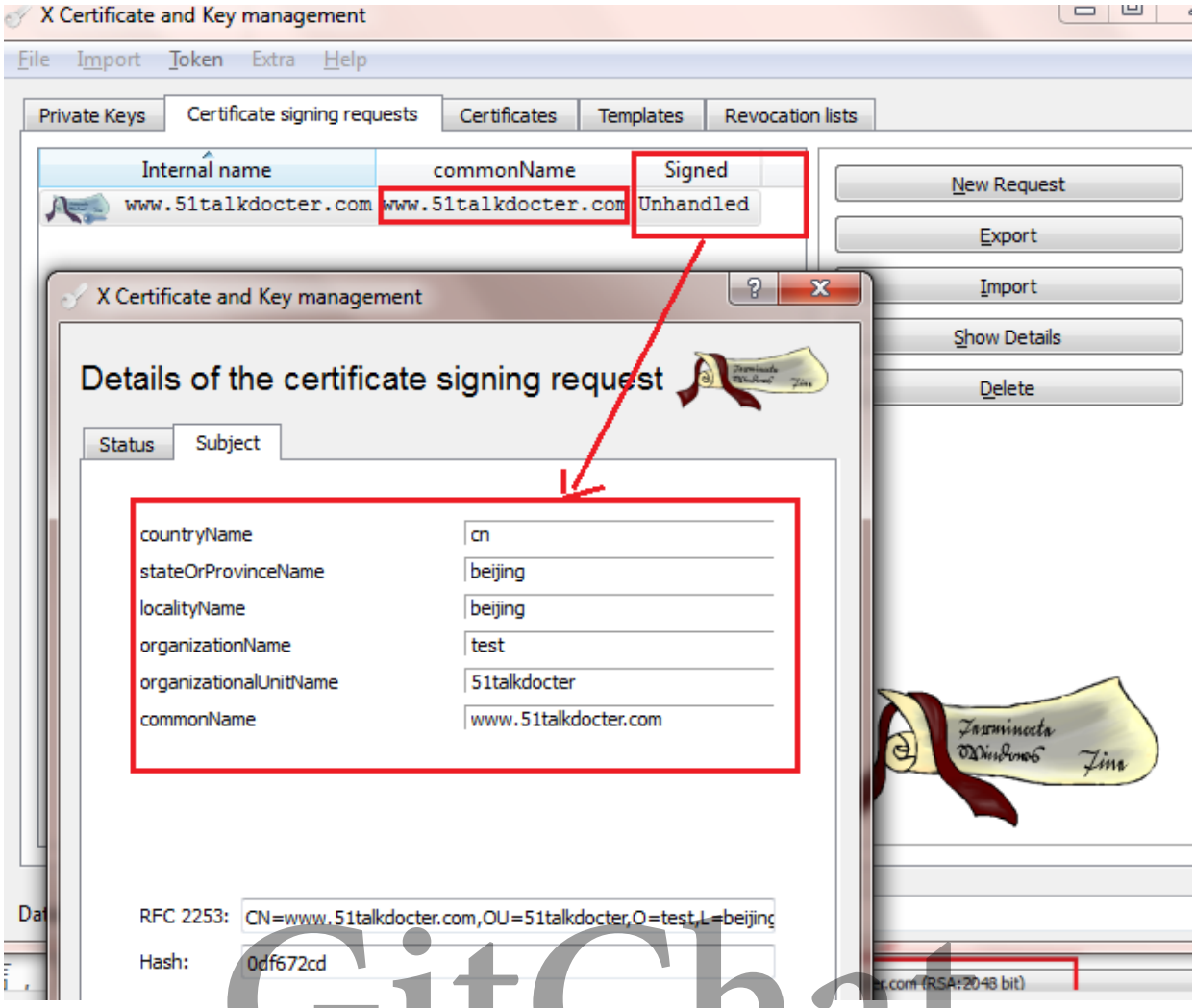
那么私钥是做什么的呢？私钥就是用来保护信息用的，前面我们提到了我们一般把公钥告诉别人，让别人用我们的公钥去加密数据，然后不告诉别人我们的私钥，这样别人用我们的公钥加密的数据，只有我们自己能解密开，因为默认情况下，私钥只有我们自己知道；所以这个私钥一定要保护好，而且必须有密码器保护，这就是为什么我们在Step4需要设置密码的原因了。

如果要生成证书请求的私钥，就需要点击当前弹出页面的”Generate a New Key” 按钮，就会弹出一个窗口来生成包含证书请求的私钥，见下图：



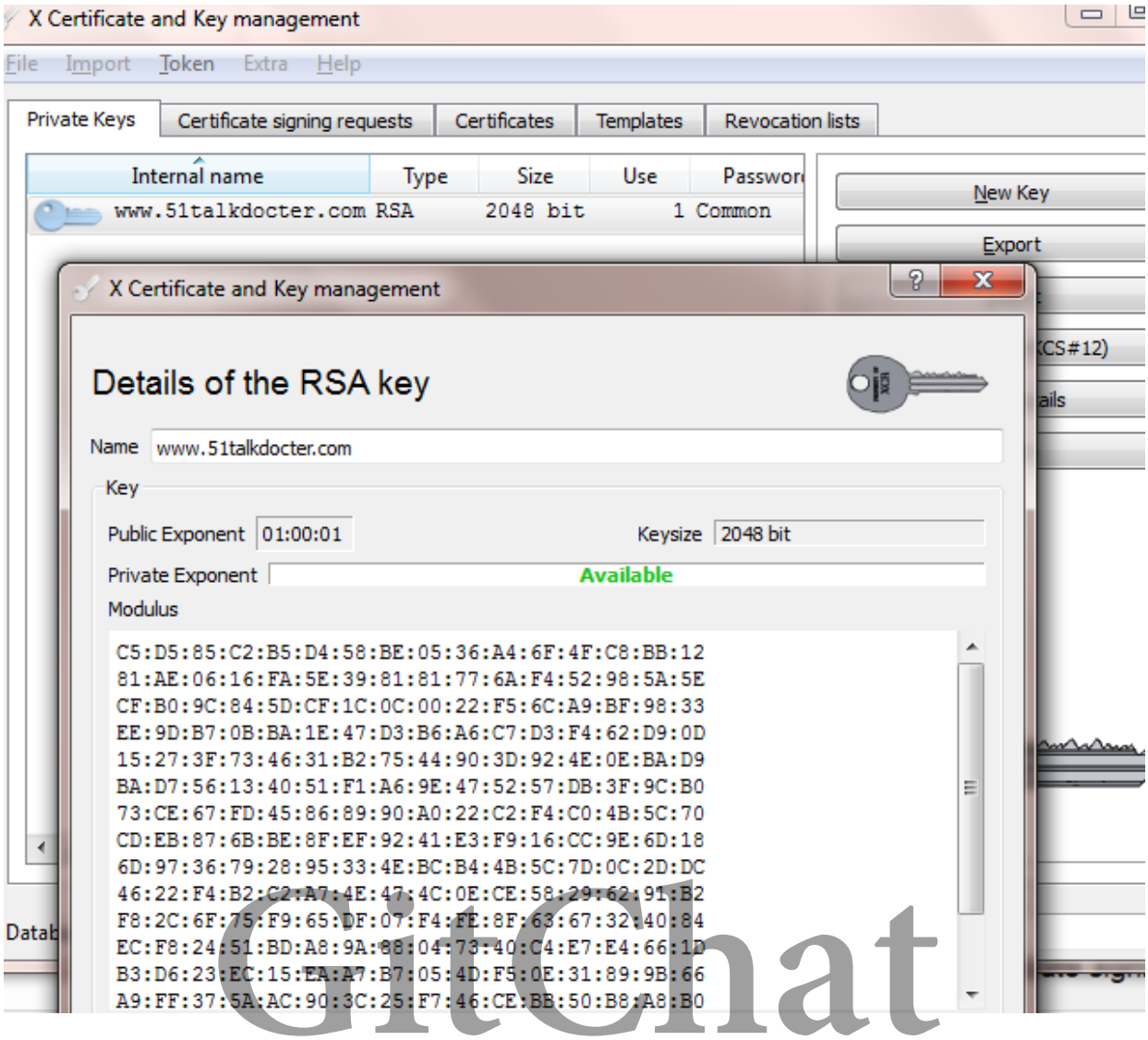


Step9. 查看证书信息： 在“Certificate Signing Request”标签页，选择你申请的证书，然后双击，将会弹出一个页面，点击“Subject”标签页，将会看到证书的基本信息。



从上图可以看出，生成了一个www.51talkdocter.com的证书请求，其Signed列的状态为”Unhandled”,表示其为一个等待被CA签署的证书请求。

Step10. 查看私钥基本信息：点击“Private keys” 标签页面，我们将会看到我们创建的私钥匙的信息。



一个生成www.51talkdocter.com的证书请求就这么愉快的被XCA工具给完成了，而且所见即所得，非常容易的反过头来查看证书的信息，而且对于我们理解SSL的公钥和私钥的概念以及其表达方式也特别的有用。上面只是XCA的例子而已，笔者希望能够通过这个例子让大家举一反三，其实其功能远远不止这些，大家可以在看完笔者的GitChat，如果有兴趣的话，可以继续探索，也可以参阅我的博客专栏。

9. 总结