Openssl与数字证书

（ubuntu15+openssl+https+tomcat7）

1. 术语
2. 加密算法

加密可以分为对称加密和非对称加密。

1. 对称加密，只有一个密钥，通信的双方就用这一个统一密钥进行加密和解密。对

称加密算法常用的有DES（Data Encryption Standard，数据加密标准）和AES（Advanced Encryption Standard，高级加密标准）。

1. 非对称加密，一般有两个密钥，这两个密钥称为密钥对，一个是私钥，密钥生成

者自己私密保管，另一个则是公钥，对外公开。用私钥加密的数据，只能用公钥解密，反之，用公钥加密的数据，自然也只能用私钥解密。常用的非对称加密算法有RSA（是Ron Rivest、Adi Shamir、Leonard Adleman这三个发明者的姓氏的首字母）。

1. 数字摘要

数字摘要就是采用单项Hash函数将需要加密的明文“摘要”成一串固定长度（比特位）的密文这一串密文又称为数字指纹，它有固定的长度，而且不同的明文摘要成密文，其结果总是不同的，而同样的明文其摘要必定一致。说白了，就是根据一定的算法，对明文进行抽样，内容稍有不同，抽样的结果就明显不同，但长度却是固定的。可以通过数字摘要，来鉴别明文的完整性。

常用的数字摘要算法有：MD5（128位）、SHA-1（160位、192位、256位）。

1. 数字签名

数字签名，就是“附加在数据单元上的一些数据，或是对数据单元所作的密码变换，这种数据和变换允许数据单元的接收者用以确认数据单元来源和数据单元的完整性，并保护数据，防止被人（例如接收者）进行伪造”。说的明白点，就是在数字时代，采用数字技术，将目标内容打上“签名者”的身份烙印，就像在生活中在纸质上用笔签字或画押一样，告诉人们，这东西确实是出自其手。

数字签名的常用做法，就是对数字摘要进行加密，即用私钥加密，用公钥解密。假设有这么个场景：小张要给小李通过网络发信息，但又担心发送的内容被他人获取并篡改，因为如果真被人获取并篡改的话，小李得到的信息就不是小张原来的信息，这会坏了大事，怎么办？于是他们就约定这么做：

1. 小李事先生成一组密钥对，私钥自己持有，公钥在发信息前告诉小张。
2. 小张用小李的公钥对信息进行加密，然后发给小李，这时因为信息用公钥过加密，别人即使获到，

没有私钥也无法解密，保证了信息的安全。

1. 信息到达小李之后，小李用私钥进行解密，从而得到小张发过来的信息。这时小李要给小张回信

息，他先把要回复的信息生成一份摘要，然后用自己的私钥对摘要进行加密，即数字签名，最后把信息连同数字签名一起发送给小张。

1. 小张得到回信后，首先辨认签名，即用小李的公钥对签名进行解密，如果解密成功，说明信息确

实是小回复过来的无疑（回信者身份得到确认）；接着验证信息的完整性，看有没有被他人篡改过（因为公钥是公开的），这时小张对回信进行数字摘要（他们事先约定好摘要算法），然后和解密的数字摘要进行对比，如果一致，则说明信息没有被篡改过，是小李发过来的真实信息；如果不一致，说明信息被篡改过，小张把该信息丢弃掉。

常用的数字加密算法有：RSA，非对称加密，加密速度慢，一般对短小的内容进行加密，如密码；DES/AES，对称加密，机密速度相对与RSA快，一般对大篇幅内容进行加密。通常的加密做法是，用DES/AES密钥对通信的内容进行加密，而DES/AES的密钥则用RSA私钥或公钥进行加密。

1. 数字证书

虽然彼此用私钥加密，用公钥解密，或用公钥加密用私钥解密，实现了对信息进行加密的同时，又验明了对方的“真身”。但是，这可是“隔空”的，彼此“不见面”，对方被冒充了怎么办？谁能证明小张拿到的公钥就是小李本人的，能保证不被掉包？于是想到了一个办法：数字证书。通过第三方权威机构（证书授权中心）给数字证书申请者颁发数字证书（其实是对数字证书申请者的公钥进行公证），从而确保通信双方身份的真实性。

数字证书是“一个经证书授权中心（即CA）[数字签名](http://baike.baidu.com/view/7626.htm)的包含[公开密钥](http://baike.baidu.com/view/1145160.htm)拥有者信息以及公开密钥的文件”。最简单的证书包含证书名称、一个申请者的公开密钥以及证书授权中心的数字签名。CA对数字证书签名，就是对证书的内容（如证书名称，申请者身份信息、申请者的公钥以及证书的实效性，当然还有摘要和加密的算法）先生成摘要，然后对摘要用CA自己的私钥对证书摘要进行加密。

有了数字证书，当双方通信过程是这样的：

1. 小张（他事先持有给小李颁发证书的CA的公钥）要发信息给小李，他先要求小

李先提供数字证书。

1. 小李提供数字证书给小张。
2. 小张拿到小李的数字证书后，先验证证书的真伪，即用CA的公钥对数字证书的

签名进行解密，如果解密成功，说明该数字证书确实是对应的CA颁发的；紧跟着就是验证数字证书的完整性，即用对签名解密后得到的摘要，与小张自己对数字数字证书的摘要进行对比，如果一致，说明数字证书内容是真实的，没有被篡改。于是，小张开始放心地给小李发信息了（通信握手成功）：用数字证书中小李的公钥加密信息，发给小李。

1. SSL/ TLS

SSL，Secure Sockets Layer的简称，即[安全套接层](http://baike.baidu.com/view/525499.htm)。TLS，Transport Layer Security简称，翻译成中文叫“[传输层安全](http://baike.baidu.com/view/831898.htm)”，是SSL的继任者。SSL/TLS是为[网络通信](http://baike.baidu.com/view/538641.htm)提供安全及[数据完整性](http://baike.baidu.com/view/702953.htm)的一种安全协议，它介于应用层与传输层之间，对网络连接进行加密。

1. HTTPS

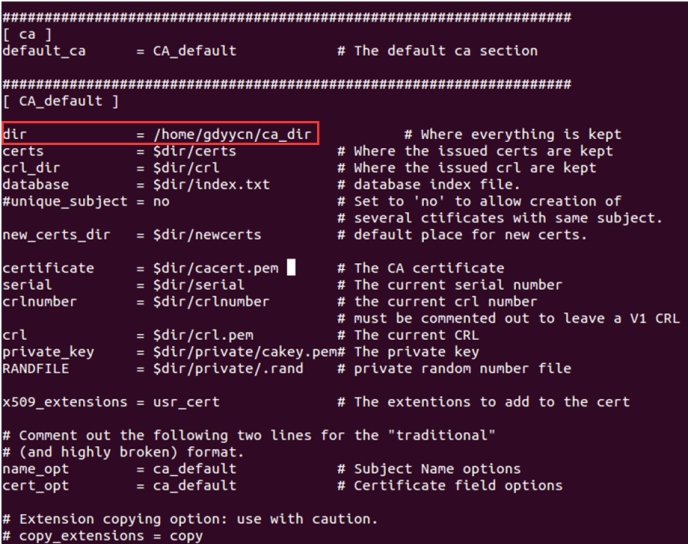
HTTPS，全称叫“Hypertext Transfer Protocol Secure”，翻译成中文叫“[安全超文本传输协议](http://baike.baidu.com/view/831896.htm)”，它建立在SSL/TLS的基础之上，它的目标是给HTTP提供安全的数据传输保证，即网络上传输的数据是加密的，而不是像HTTP那样进行明文传输。

1. 数字证书签发流程
2. 数字证书申请者生成密钥信息。
3. 导出签发申请（内容包括申请者信息、公钥等），提供给CA。
4. CA签发数字证书（内容包括申请者信息、申请者公钥、约定的加密算法、证书

有效期和CA的签名等）。

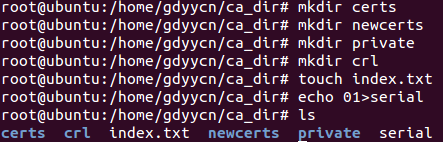
1. OpenSSL配置
2. 修改openssl的工作目录

找到/usr/local/ssl/ openssl.cnf配置文件，将工作目录修改为自己指定的某个目录，如“/home/gdyycn/ca\_dir”



1. 在设定的工作目录下，创建将要用到的相关子目录certs（证书申请文件和签发

的证书存放目录）、newcerts（新证书存放目录，如果需要的话）、private（随机数和密钥存放目录）、crl）和index.txt文件。



1. 创建和自签发根证书

这里的根证书，作为CA的签发其他证书的证书。

1. 创建随机数文件

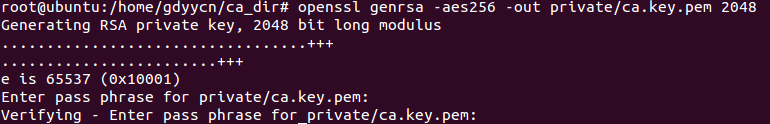
openssl rand -out private/.rand 1000



1. 创建根证书密钥

设置根证书密钥密码：88888888，密钥文件为ca.key.pem。

openssl genrsa -aes256 -out private/ca.key.pem 2048



genrsa表示生成RSA私钥

-aes256 表示用256位的AES密钥算法对RSA私钥进行加密

-out 私钥输出路径

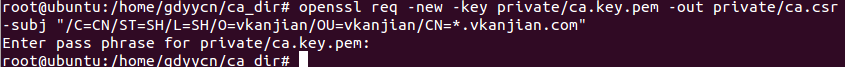
2048 表示RSA密钥长度



1. 创建根证书签发申请

通过根证书的密钥信息，创建根证书申请文件，文件名称为ca.csr，主体为“\*.vkanjian.com”，前面的“\*”表示对“vkanjian.com”的所有子级域名均有效。

openssl req -new -key private/ca.key.pem -out private/ca.csr -subj "/C=CN/ST=SH/L=SH/O=vkanjian/OU=vkanjian/CN=\*.vkanjian.com"





req 表示申请

-new 表示新请求

-key 刚生成的密钥

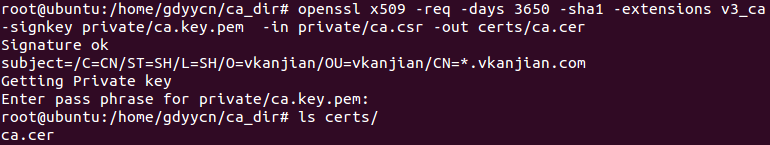
-out 签发申请输出路径

-subj 签发对象相关信息（即用户信息）

1. 自签发根证书

用根证书密钥，进行自签发，签发的文件为ca.cer。

openssl x509 -req -days 3650 -sha1 -extensions v3\_ca -signkey private/ca.key.pem -in private/ca.csr -out certs/ca.cer



X509 表示签发X.509格式证书

-req 表示证书输入请求

-days 有效天数

-sha1 表示SHA1摘要算法

-extensions 扩展项，值为“v3\_a”，表示按OpenSSL配置文件（openssl.cnf）v3\_a项来

配置

扩展项

-signkey 签名密钥

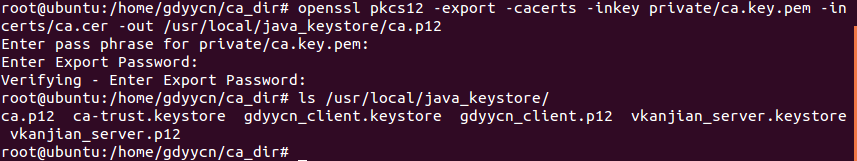
-in 前面生成的签发申请

-out 输出签发的证书（此处为ca.cer）

1. 导出PKCS#12格式根证书
2. 用openssl导出

执行如下命令，三次密码均输入：88888888，导出文件为ca.p12。

openssl pkcs12 -export -cacerts -inkey private/ca.key.pem -in certs/ca.cer -out /usr/local/java\_keystore/ca.p12



pkcs12 表示转换成PKCS#12格式文件

-export 导出

-cacerts 表示导出CA（证书认证机构）证书

-inkey 输入已经生成的CA密钥

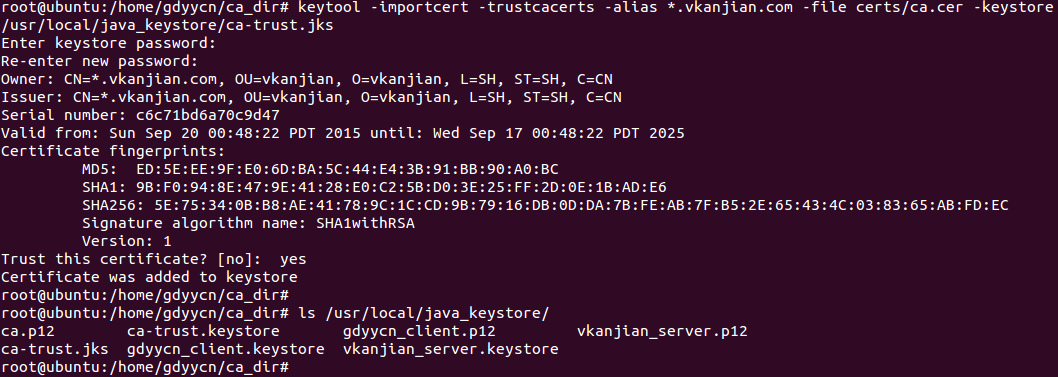
-in 输入自签发的根证书

-out 输出转换成PKCS#12格式的证书

1. 用JAVA自带的keytool工具导出

导出的格式则为JKS，文件名这里为ca-trust.jks：

keytool -importcert -trustcacerts -alias \*.vkanjian.com -file certs/ca.cer -keystore /usr/local/java\_keystore/ca-trust.jks



-importcert 表示导入证书

-trustcacerts 表示到信任库

-alias 信任库证书别名

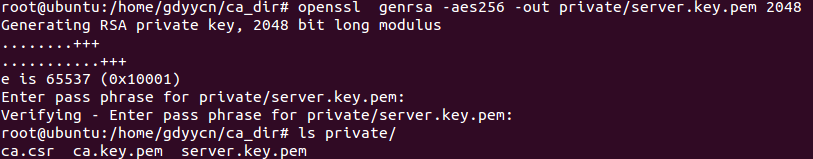
-file输入自签发的根证书

-keystore 输出的信任库文件路径

1. 签发服务器证书
2. 创建密钥

创建服务器证书密钥，密钥密码为：88888888，文件名为server.key.pem。

openssl genrsa -aes256 -out private/server.key.pem 2048



genrsa表示生成RSA私钥

-aes256 表示用256位的AES密钥算法对RSA私钥进行加密

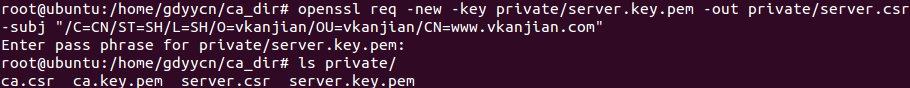
-out 私钥输出路径

2048 表示RSA密钥长度

1. 创建服务器证书签发申请

用服务器证书密钥，创建服务器证书签发申请，签发对象指定为“[www.vkanjian.com](http://www.vkanjian.com)”，申请文件为server.csr。

openssl req -new -key private/server.key.pem -out private/server.csr -subj "/C=CN/ST=SH/L=SH/O=vkanjian/OU=vkanjian/CN=www.vkanjian.com"



req 表示申请

-new 表示新请求

-key 刚生成的密钥

-out 签发申请输出路径

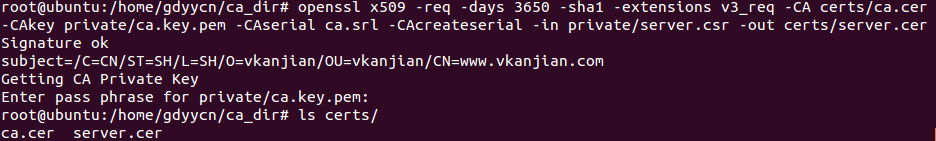
-subj 签发对象相关信息（即用户信息）

1. 用根证书签发服务器证书

用CA的证书、CA的密钥和服务器的签发申请来签发服务器端证书。签发的后的服务器

证书文件为server.cer。

openssl x509 -req -days 3650 -sha1 –extensions v3\_req -CA certs/ca.cer -CAkey private/ca.key.pem -CAserial ca.srl -CAcreateserial -in private/server.csr -out certs/server.cer

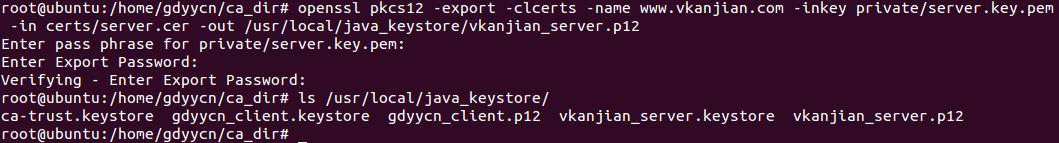


1. 导出PKCS#12格式服务器端证书

导出到java环境可用的PKCS#12格式的服务器端证书，导出文件为vkanjian\_server.p12。

执行如下命令，三次密码均输入：88888888。

openssl pkcs12 -export -clcerts -name www.vkanjian.com -inkey private/server.key.pem -in certs/server.cer -out /usr/local/java\_keystore/vkanjian\_server.p12



pkcs12 表示转换成PKCS#12格式文件

-export 导出

-clcerts 表示导出客户证书

-name 要导出的证书名称

-inkey 输入已经生成的服务器端密钥

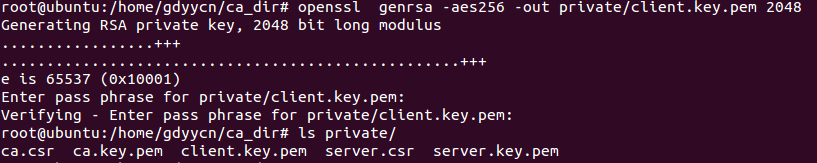
-in 输入签发的服务端证书

-out 输出转换成PKCS#12格式的证书

1. 签发客户端证书
2. 创建客户端证书密钥

执行如下命令，输入密钥密码“88888888”，创建密钥文件client.key.pem。

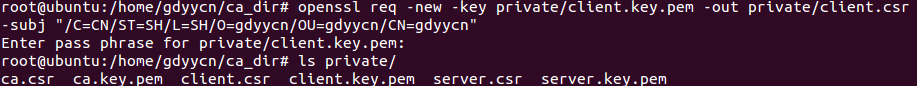
openssl genrsa -aes256 -out private/client.key.pem 2048



1. 创建客户端数字证书签发申请

用客户端证书密钥信息，创建客户端数字证书签发申，申请文件为client.csr。

openssl req -new -key private/client.key.pem -out private/client.csr -subj "/C=CN/ST=SH/L=SH/O=gdyycn/OU=gdyycn/CN=gdyycn"

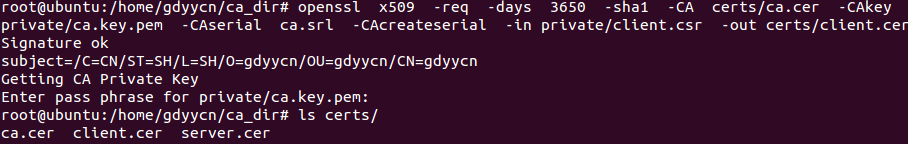


1. 用根证书签发客户端证书

用CA的证书、CA的密钥和客户端的签发申请来签发客户端证书。签发后的客户端证书

文件为client.cer。

openssl x509 -req -days 3650 -sha1 -CA certs/ca.cer -Cakey private/ca.key.pem -CAserial ca.srl -CAcreateserial -in private/client.csr -out certs/client.cer

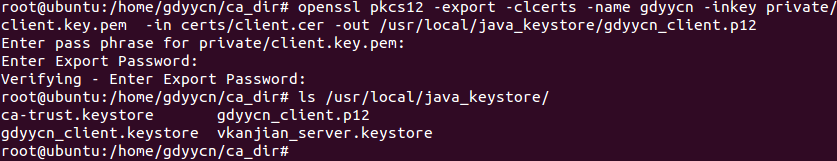


1. 导出PKCS#12格式客户端证书

从客户端证书client.cer中导出到java环境可用的PKCS#12格式的客户端证书文件gdyycn\_client.pk12。

执行如下命令，三次密码均输入：88888888

openssl pkcs12 -export -clcerts -name gdyycn -inkey private/client.key.pem -in certs/client.cer -out /usr/local/java\_keystore/gdyycn\_client.p12



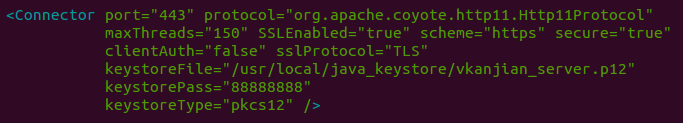
1. Tomcat单向认证（https）

只验证服务器端的数字证书。

1. 修改服务器文件

在tomcat安装目录下，找到${tomcat\_home}/conf/server.xml文件，将注释掉的端口为

8443的connector打开，修改成如下图的设置。



keystoreFile : 服务器PKCS#12格式证书库路径

keystorePass: 服务器端证书库密码

keystoreType: 证书库类型（为PKCS#12格式）

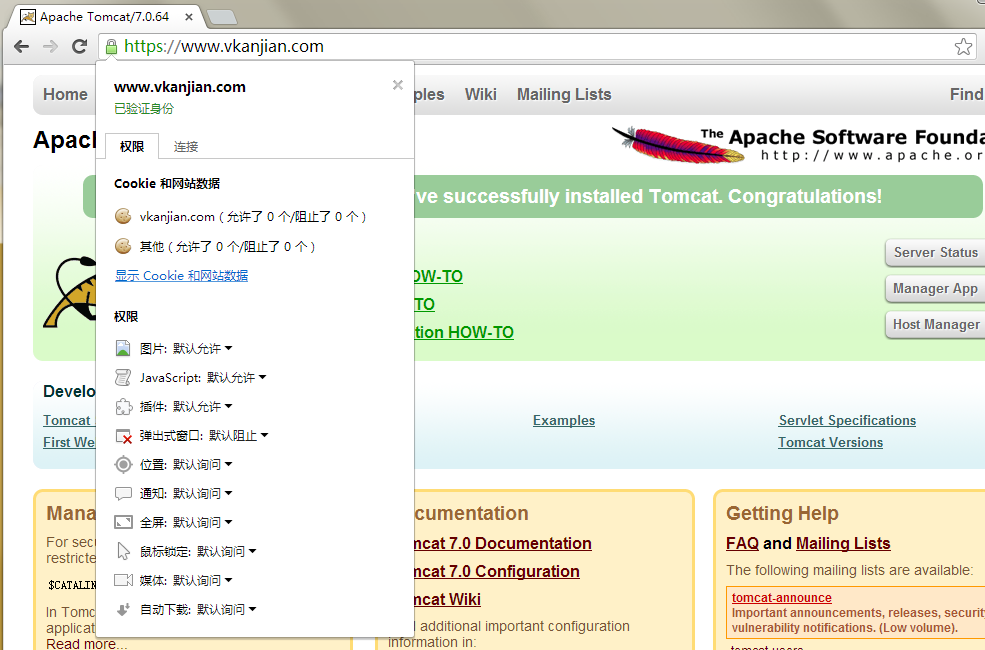
端口改成: 443

1. 访问

通过浏览器访问https://www.vkanjian.com/（该域名通过操作系统hosts文件进行了映射），浏览器报证书不受信任警告。



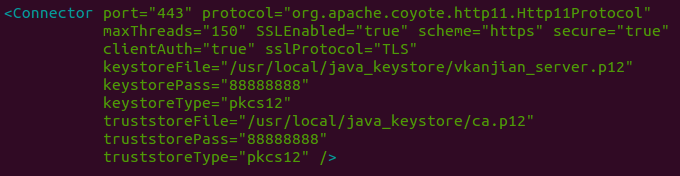
因为这里给“[www.vkanjian.com](http://www.vkanjian.com)”签发的CA不是国际知名的CA机构，浏览器里内置的CA列表里是找不到的，所以我们必须把给“[www.vkanjian.com](http://www.vkanjian.com)”签发证书的根证书导入到浏览器中来。我们给“[www.vkanjian.com](http://www.vkanjian.com)”签发的根证书是ca.cer，我们就把它导入。导入ca.cer后，访问就正常了（因为服务器证书是由根证书签发的，所以浏览器只需导入该根证书，其签发的其他证书都将受信任）。



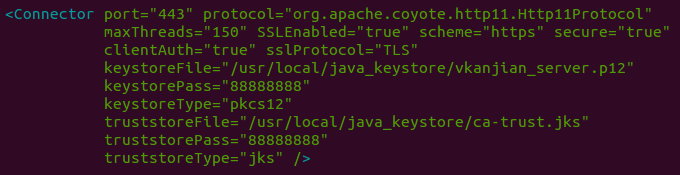
1. Tomcat双向认证（https）

客户端和服务器端相互验证对方的数字证书。这里客户端为“gdyycn”，给客户端签发的证书是，服务器端为“[www.vkanjian.com](http://www.vkanjian.com)”。

1. 配置tomcat server.xml文件



或



clientAuth:true 表示需要客户端认证

truststoreFile : PKCS#12格式根证书库（信任库）路径

truststorePass: 根证书库密码

truststoreType: 证书库类型（为PKCS#12格式）

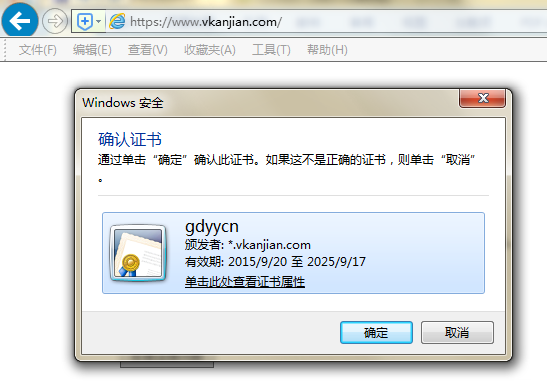
1. 将客户端证书（gdyycn\_client.p12）导入到浏览器

首次访问会提示你选择要导入的客户端证书（即个人证书）

谷歌浏览器提示



IE浏览器提示



选择完成后即可访问服务器“[www.vkanjian.com](http://www.vkanjian.com)”。

1. 附录

## 证书相关文件列表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 密钥 | 签发申请 | 数字证书 | PKCS#12格式 | 签发对象 |
| CA根证书 | ca.key.pem | ca.csr | ca.cer | ca.p12 | \*.vkanjian.com |
| 服务器端证书 | server.key.pem | server.csr | server.cer | vkanjian\_server.p12 | [www.vkanjian.com](http://www.vkanjian.com) |
| 客户端证书 | client.key.pem | client.csr | client.cer | gdyycn\_client.pk12 | gdyycn |

## 浏览器导入数字证书（谷歌）



