HTTPS 原理与证书实践

惨绿少年 Linux运维, 安全, 运维基本功 0评论 来源：本站原创 44℃ 字体： 小 中 大

1.1 网络安全知识

1.1.1 网结安全出现背景

网络就是实现不同主机之间的通讯，网络出现之初利用TCP/IP协议簇的相关协议概念，已经满足了互连两台主机之间可以进行通汛的目的，虽然看似简简单单几句话，就描述了网络概念与网络出现的目的，但是为了真正实现两台主机之间的稳定可靠通讯，其实是一件非常困难的事情了，如果还要再通讯的基础上保证数据传输的安全性，可想而知，绝对是难上加难，因此，网络发明之初，并没有太关注TCP/IP互联协议中的安全问题。

对于默认的两台主机而言，早期传输数据信息并没有通过加密方式传输数据，设备两端传输的数据本身实际是明文的，只要能截取到传输的数据包，就可以直接看到传输的数据信息，所以根本没有安全性可言。

1.1.2 网络安全涉及问题

早期网络设备间进行通讯时，是采用明文进行数据传输的，并没有网络安全技术可言。我们可以提出一种假设，如果在网络上的两台主机之间传输数据，就是采用明文的方式；并且对于网络传输而言，并没有相关的网络安全技术保驾护航，这样在互联网上进行数据传输，都有哪些网络风险需要进行面对。

- **网络安全问题—数据机密性**

在网络传输数据信息时，对数据的加密是至关重要的，否则所有传输的数据都是可以随时被第三方看到，完全没有机密性可言。

- **网络安全问题—数据完整性**

网络传输数据的完整性，也是安全领域中斋要考虑的里要环节，如果不能保证传输数据的完整性，那传输过程中的数据就有可能被任何人所篡改，而传输数琚双方又不能及早的进行发现，将会造成互连通汛双方所表达信息的意义完全不一致。因此，对于不芫整的数据信息，接收方应该进行相应判断，如果完整性验证错误，就拒绝接收相应的数据。

- **网络安全问题—身份验证问题**

网络中传输数据时，很有可能传输的双方是第一次建立连接，进行相互通讯，既然是第一次见面沟通，如何确认对方的身份信恿，的确是我要进行通讯的对象呢？如果不是正确的通讯对象，在经过通讯后，岂不是将所有数据信息发送给了一个陌生人。

1.2 网络安全问题的解决

1.2.1 网络安全解决问题—如何保证数据的机密性

- **数据机密性解决思路—利用普通算法解决机密性**

为了保证数据的机密性，首先可以采用的方法就是将数据通过相应算法，转换为其它的数据信息，然后再通过相应算法反推出真正的数据是什么，这样一来就保证了数据在网络传输过程中安全性，不会被其它人轻易的看到传输过程中的数据信息.数据加密前的信息称为明文数据（plaintext）,经过加密算法转换后进行传输的信息称为密文数据（ciphertext）;反之经过解密算法转换后，会将密文数据恢复为明文数据进行显示接收。

图 – 数据机密性处理示意图

- 优点：

  实现了数据机密传输，避免了明文传输数据的危险性。

- 缺点：

  利用加密算法，将明文改密文，如果第三方获得加密算法，即可将传输密文再次变为明文

➢ **数据机密性解决思路—利用对称加密算法解决机密性**

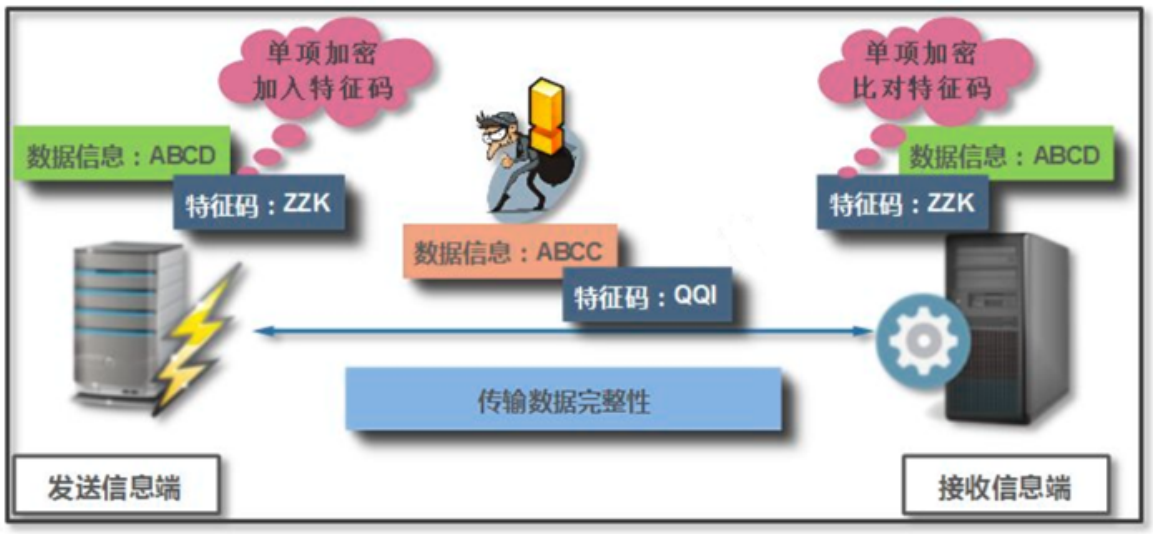
普通算法虽然已经解决了明文数据的机密性，可以在网络传输过程中不被直接看到明文数据。但是新的问题又产生了，既然明文数据是通过算法改变成了新的数据信息，如果第三方获得了算法，利用算法也是可以将密文数据信息，再次转换为明文数据信息，因此出现了对称加密算法，形象比喻来说：数据加密算法就好比是一本密码规则手册，而对称加密算法就是将手册放在了一个保险柜中进行了上锁传输，只有传递数据信息的双方知道打开保险柜的密码。

图 – 数据机密性处理示意图

1.2.2 网络安全解决问题–如何保证数据的完整

➢ **数据完整性解决思路—利用单项加密算法**

利用数据的单项加密算法（提取数据指纹），进行提取数据特征码的方式，从而完成数据传输的完整性验证.实际的算法实现过程为：在一段明文数据信息后加上数据信息的特征码，这个特征码是通过结合数据信息进行相应算法获得的数据特征码，接收方当收到数据信息后，会利用相同的加密算法对获取的数据进行加密，确认加密后得到的特征码是否与传送过来数据后面描述的特征码一致；如果一致，可以表示数据没有被篡改过，如果不一致表示数据完整性遭到了破坏，数据一概不予以接收处理。



- **数据完整性解决思路—利用单项加密算法（加密特征码）**

由于可能存在中间人攻击的可能性，因此可以对传输过程中数据特征码进行加密，发送方利用对称密钥方式对手中的特征码进行加密，接收方会利用相同的密钥对手中的特征码进行解密，从而确认特征码是否一致.如果中间人将新的特征码也进行了加密，发送给接收方，但接收方无法利用和发送方协商好的解密密钥对特征码进行解密，最终无法识别中间人发送过来的数据特征码信息。

在此需要了解一下加密算法的原理↓

通讯双方需要进行数据密钥信息约定（密钥协商过程），在密钥相互可以获悉的过程中，需要借助密钥交换机制（Internet key exchange IKE）,进而实现密钥的交互。为了满足网络中，两台主机间密钥交换统一的过程，需要引入一种新的协议diffie-hellman协议。

diffie-hellman协议算法实现的过程：

1) 首先发送方选取一个大素数P(只能被1和自己整除的数)，再选取一个生产数g ,并且发送方将P与g经过互联网传输到接收方。

2) 数据传输的两端，发送方选取一个随机数x,接收方选取一个随机数y;发送方只知道随机数x ,接收方只知道随机数y , x和y不在互联网上进行传输。

3) 接收数据双方开始进行计算，对于发送方进行计算g的x次方对P取模的结果，传输给接收者；而接收方进行计算g的y次方对P取模的结果，传输给发送者。

4) 此时对于接收方获取到了发送方的g的x次方对P取模的结果，在取模结果的基础上进行y次方的运算，y就是接收方自身产生的随机数y;而对于发送方获取到了接收方的g的y次方对P取模的结果，在取模结果的基础上进行x次方的运算。x就是发送方自身产生的随机数x#此次，双方的加密运算密钥就实现了交换，并且是统一一致的，最终的密钥为g的xy次方对P取模的结果。

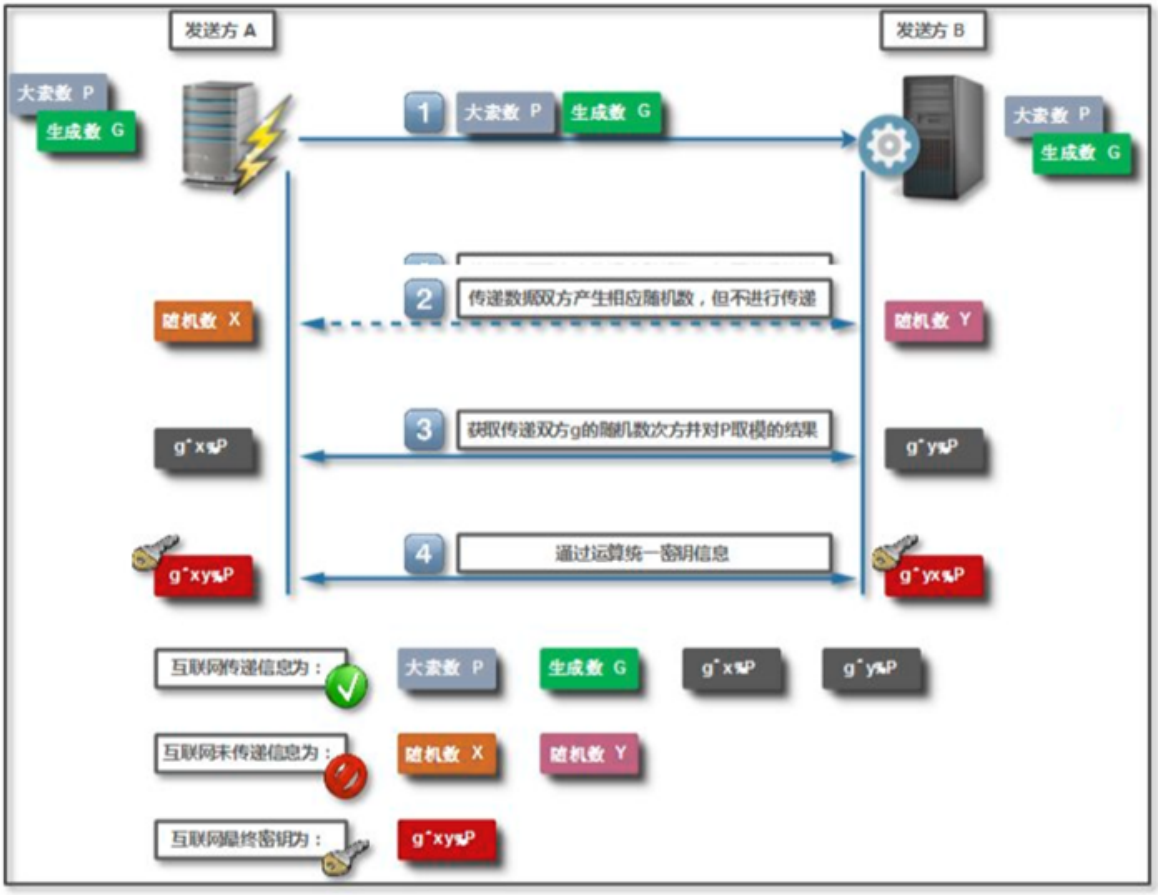


图 – diffie-hellman协议算法示意图

通过diffie-hellman协议算法最终可以实现了网络传输双方的密钥交换，并且通讯的双方也从此不用再对密钥进行管理记忆，只需要在进行传输数据前，进行一次密钥的交换过程；即完成了对称密钥的生成过程。

并且如果网络中有攻击者在尝试破解出交换密钥时，就算通过比较复杂的数学运算，获悉了密钥，但也只是了解了上一次的数据交换密钥信息；下一次数据传输双方通讯时，还会交换新的密钥用于新的数据传输。

1.2.3 网络安全解决问题–如何进行传输双方身份验证

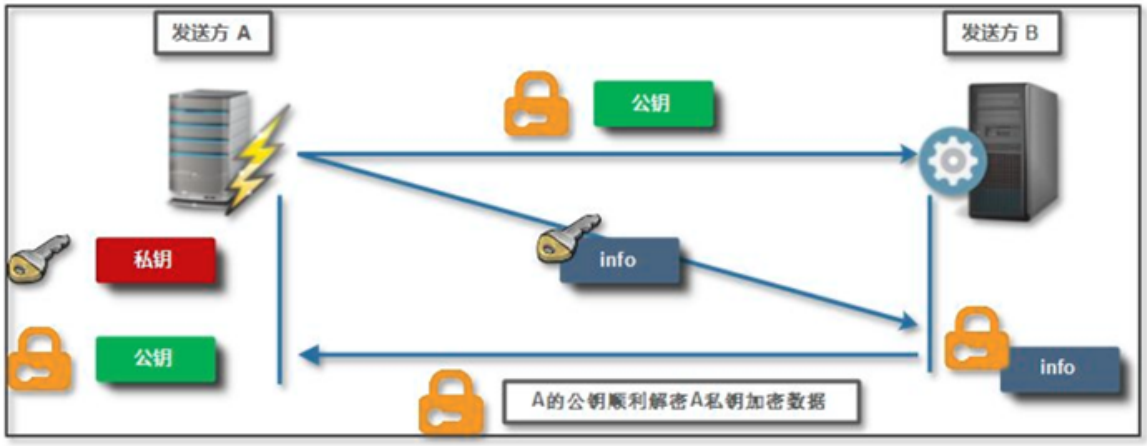
- **网络安全身份验证解决方案**

以上信息只是解决密钥的交换获取问题，但是网络的身份验证问题依旧没有进行解决，换句话说，通讯双方的确可以获取统一的密钥信息了，并且是通过相对安全的方式获取得知的，但是通讯双方在交换获取密钥前，又怎么确认得知，交换密钥的对方的确是要进行通讯的发送方或接收方呢?

- **安全身份验证解决思路–利用非对称密钥加密算法（公钥加密算法）**

利用非对称加密算法（公钥加密算法），可以从根本上有效解决网络中，数据传输双方的安全身份验证问题。非对称加密算法中，存在密钥对的概念，即拥有公钥（public key）与私钥（pnvate key）,其中公钥不是自行创建出来的而是从私钥中提取出来一部分作为公钥，因此可以说公钥是来自于私钥的，而私钥才决定了密钥加密的安全性，于是私钥的长度可能会非常长，从最初的1024,2048,到4096一直到更多的位数，将私钥密钥位增加的很长，从而提升了密钥安全性。

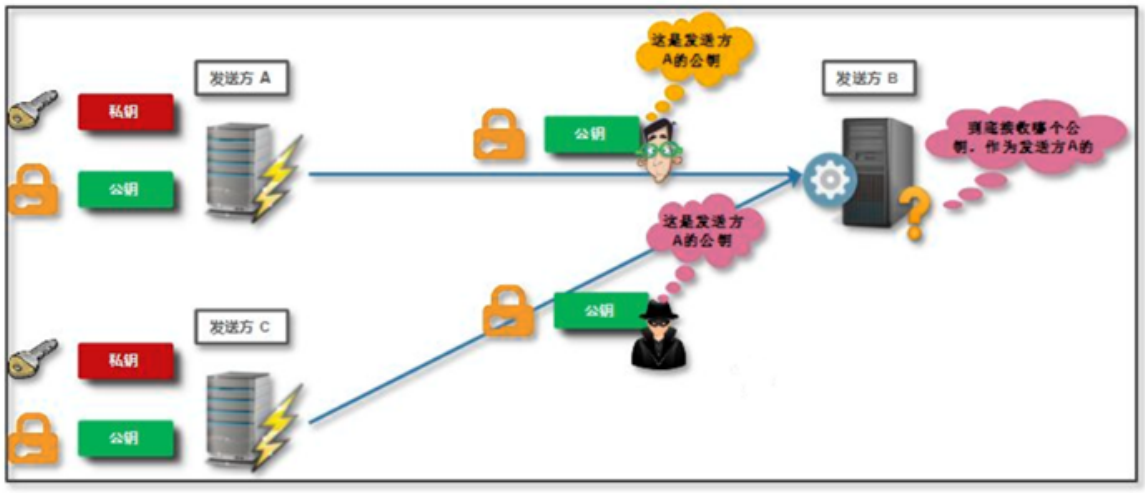
利用非对称加密算法，需要遵循一个基本原则：公钥加密的只能利用与之配对的私钥进行解密，反之也是一样的.但是非对称加密算法并不能用于对数据完整性进行验证，因为私钥只有一份，但公钥可以有很多份。尽管非对称加密算法不能满足数据 完整性验证，但完全可以满足对传输者身份验证的需求，因为接收者可以拥有相应的公钥，只有与之对应的发送者用相应的私钥进行加密信息，用对应的公钥自然即可解密，否则可以确认发送者身份已经发生了变化。



1.3 证书的由来

1.3.1 如何获取公钥信息

默认公钥在网络中进行传递时，默认情况下也是会出现问题的如下图所示：



对发送方的公钥信息进行公正步骤：（借助第三方安全机构）

a) A和B端首先生成自己的公钥和私钥的密钥对，为了使对方能相信自己的公钥信息，将自己的公钥信息告知给第三方发证机构，利用第三方机构对自己的公钥进行公证，第三方机构会制作一个数字证书（机构编号以及发证机构的联），并且第三方机构也要给自己设置一个合法的公钥和私钥，并且公钥设为第三方机构的公钥证书。

b) 发证机关计算出数字证书数据的特征码，并用自己的私钥进行加密，并将加密的信息附加到 特征码后成为数字签名。

c) A和B两锭获得公正过的证书信息，并通过证书信息传递.得到对方的公钥。

d) A和B两端与第三方机构建立连接，获得第三方证书，通过第三方证书获得第三方公钥，利用第三方公钥只要能解密数字签名即可。

1.3.2 证书信息所包含什么内容

目前标准的证书存储格式是x509，还有其他的证书格式，需要包含的内容为：

- ✓ 公钥信息，以及证书过期时间
- ✓ 证书的合法拥有人信息
- ✓ 证书该如何被使用
- ✓ CA颁发机构信息
- ✓ CA签名的校验码

互联网上使用的SSL和TLS证书管理机制均使用x509的格式。

1.4 加密算法的简介

1.4.1 对称加密算法

对称加密算法特性是加密和解密使用同一个密钥，利用对称算法可以将明文改为密文（加密），密文还原为明文（解密）。

对称加密算法常见的有：

- 最早期的称为DES(Data Encryption Standard),是美国国家安全局征集加密算法时，由一个美国公司提出的，是公开可以使用的，使用的是56位的密钥长度，但是由于计算机的发展，可以使用计算机对56位的密钥进行暴力破解了，因此DES渐渐不再被使用。
- 一种新的算法.将DES加密后，再进行一次DES加密，然后再进行一次DES ,称为3DES算法，是目前使用比较多的加密算法。
- 更安全的加密算法，AES（高级加密标准）加密算法产生.默认使用128位的加密密钥，但是也有特殊的AES（AES192 AES256 AES512等），密钥越长安全性提高的同时，加密效率就会降低，因此应该选择比较合适的加密算法。
- blowfish加密算法，加密不是按位进行加密的，而是将数据分成大小相同的数据块进行加密的。

1.4.2 单向加密算法

单向加密算法常见的有：

- DH加密算法，主要用于密钥的协商交换
- MD4 MD5(128)
- SHA1(160) SHA(192) SHA(256) SHA(384)
- CRC-32(循环输出校验码），不是加密机制，只是一种校验机制，不提供安全性，正常加密算法是不允许出现输入不一样，输出一样的情况，但CRC是可以有这样情况的，因为CRC只是具有校验功能，不具有加密功能。

单项加密算法的特征：

- 数据输入一样，特征码信息输出必然相同
- 雪崩效应，输入的微小的改变，将造成输出的巨大改变
- 定长输出，无论源数据多大，但结果都是一样的
- 不可逆的，无法根据数据指纹，还原出原来的数据信息

1.4.3 非对称加密算法

非对称加密算法可以实现身份认证功能（通过数字签名实现），数据加密功能，以及现密钥交换功能。

非对称加密算法常见的有：

- RSA,RSA既是一个公司的名称，也是三个创始人的名称，RSA既可以加密又可以进行签名。
- DSA，只能实现数字签名功能。
- ELGamal，属于商业化的加密算法。

加密和解密需要算法来实现，因此需要主机上可以有相应的软件工具来控制加密和解密的算法，相应的工具就是加密和解密算法的具体实现，对称加密算法可以实现的工具：openssl、gpg。

1.5 OpenSSL软件介绍

Netscape网景公司生产了最初的浏览器，但为了提高浏览器访问页面的安全性，对TCP/IP模型进行了一定改进，在传输层与应用层之间，创建了一个3.5层的概念，称为SSL（Secure Sockets Layer安全套接层）层，SSL不是一个软件，只是一个库，让应用层将数据传输到传输层前，调用了ssl层的功能对数据进行了加密，目前比较流行的版本是（SSLv2 v3）。

由于SSL是Netscape公司进行定义的，不够开放性。因此为了使加密功能更加开放，TSL（传输层安全协议）协议就出现了，目前比较流行的版本是（TSLv1==sslv3），TSL更像是传输层上实现的数据加密。

1.5.1 OpenSSL软件概念说明

ssl功能的开源实现，就称为openssl,功能非常强大，几乎实现了市面上主流的加密算法，并且工作性能非常的好。openssl的言方链接：http://openssl.org/。

1.5.2 OpenSSL软件组成部分

openssl是由三部分组成：

- libcrpto :通用加密库
- libssl:TSL/SSL功能的实现，基于会话的，实现了身份认证，数据机密性和会话完整性的 TSL/SSL 库
- openssl:提供的命令行工具，多用途命令工具，横拟实现私有证书颁发机构；命令行工具是通过多种子命令实现openssl的相应功能

1.5.3 OpenSSL的使用

系统环境说明

```
[root@web01 ~]# cat /etc/redhat-release
CentOS release 6.9 (Final)
[root@web01 ~]# uname -r
2.6.32-696.el6.x86_64
[root@web01 ~]# sestatus
SELinux status:                 disabled
[root@web01 ~]# /etc/init.d/iptables status
iptables: Firewall is not running.
```

检查OpenSLL软件版本：

方法一：

```
[root@web01 ~]# rpm -qa openssl
openssl-1.0.1e-57.el6.x86_64
```

方法二：

```
[root@web01 ~]# openssl version
OpenSSL 1.0.1e-fips 11 Feb 2013
```

主配置文件位置：

```
/etc/pki/tls/openssl.cnf  # 主配置文件位置
```

openssl使用帮助

⊞⊟

```
[root@web01 ~]# openssl ?
openssl:Error: '? ' is an invalid command.
# openssl: 错误: "? " 是无效命令。
Standard commands
# 标准命令
asn1parse         ca                ciphers           cms
crl               crl2pkcs7         dgst              dh
dhparam           dsa               dsaparam          ec
ecparam           enc               engine            errstr
gendh             gendsa            genpkey           genrsa
nseq              ocsp              passwd            pkcs12
pkcs7             pkcs8             pkey              pkeyparam
pkeyutl           prime             rand              req
rsa               rsautl            s_client          s_server
s_time            sess_id           smime             speed
spkac             ts                verify            version
x509

Message Digest commands (see the `dgst' command for more details)
# 单向加密 消息摘要命令 (有关详细信息, 请参阅 "dgst" 命令)
md2               md4               md5               rmd160
sha               sha1

Cipher commands (see the `enc' command for more details)
# 密码命令 (有关详细信息, 请参阅 "enc" 命令)
aes-128-cbc       aes-128-ecb       aes-192-cbc       aes-192-ecb
aes-256-cbc       aes-256-ecb       base64            bf
bf-cbc            bf-cfb            bf-ecb            bf-ofb
camellia-128-cbc  camellia-128-ecb  camellia-192-cbc  camellia-192-ecb
camellia-256-cbc  camellia-256-ecb  cast              cast-cbc
cast5-cbc         cast5-cfb         cast5-ecb         cast5-ofb
des               des-cbc           des-cfb           des-ecb
des-ede           des-ede-cbc       des-ede-cfb       des-ede-ofb
des-ede3          des-ede3-cbc      des-ede3-cfb      des-ede3-ofb
des-ofb           des3              desx              idea
idea-cbc          idea-cfb          idea-ecb          idea-ofb
rc2               rc2-40-cbc        rc2-64-cbc        rc2-cbc
rc2-cfb           rc2-ecb           rc2-ofb           rc4
rc4-40            seed              seed-cbc          seed-cfb
seed-ecb          seed-ofb          zlib
```

View Code openssl使用帮助

加密一个文件的方法:

```
openssl enc -des3 -salt -a -in inittab -out initab.des3 # 输入密码后BP加密成功
openssl enc -des3 -d -salt -a -in initab.des3 -out inittab    # 输入钥后即解密成功
```

说明：其中命令中的salt参数，主要用于避免密码加密后，对密钥串的反推

输出一个文件的特征码方式

```
md5sum inittab
shalsum inittab
openssl dgst -shal inittab      # 利用openssl 生成文件特征码
                                # dgst---表示指定使用摘要命令
                                # -shal---表示指定摘要命令选用shal算法
```

生成和用户一样的密码串

```
openssl  passwd -1  # 采用md5加密用户密码串
```

生成伪随机数方法

```
openssl rand -base 64 45 # 给出一个任意数字，就会生产任意的随机数
```

1.5.4 OpenSSL软件建立是有CA

- ✓ **创建私钥与公钥信息**

需要先给ca证书颁发机构生成证书，即生成一对密钥；genrsa – generate an RSA private key利用gerusa生成密钥信息。虽然只是生成私钥，但需要清除公钥是通过私钥进行提取得到的，所以只要有私钥，就可以有公钥，私钥信息是非常重要的，因此生成的私钥文件应该是600的权限。

```
openssl genrsa 1024 >server.key    # 创建私钥信息，并指定私钥的长度为2048，并将生成的私钥信息保存在一个文件中
openssl genrsa -out server.key 1024 # 将私钥信息直接进行保存，加密长度一定要放在输出文件后面
 (umask 077;openssl genrsa -out server1024.key 1024) # 利用小括号，实现子shell功能，临时修改umask，使之创建的私钥文件

说明：密钥文件也可以进行加密的，并且支持后期手工加密，但不建议加密，每次使用私钥文件还要进行解密，比较麻烦
openssl rsa -in server1024.key -pubout     #读取私钥文件命令
```

- ✓ **生成自签署的证书**

```
[root@web01 ~]# openssl req -new -x509 -key server1024.key -out server.crt -days 365
参数说明
req         # 用于创建新的证书
new         # 表示创建的是新的证书
x509        # 表示定义证书的格式为标准格式
key         # 表示调用的私钥文件信息
out         # 表示输出证书文件信息
days        # 表示证书的有效期
```

生产自签发证书的过程。

```
[root@web01 key]# openssl req -new -x509 -key server1024.key -out server.crt -days 3650
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) [XX]:CN  # 定义生成证书的国家
State or Province Name (full name) []:BJ  # 定义生成证书的省份
Locality Name (eg, city) [Default City]:BJ  # 定义生成证书的城市
Organization Name (eg, company) [Default Company Ltd]:nmtui.com  # 生成证书的组织
Organizational Unit Name (eg, section) []:ops     # 生成证书的职能部门
Common Name (eg, your name or your server's hostname) []:blog.nmtui.com # 主机名称
Email Address []:admin@nmtui.com  # 邮件地址
# 说明：此输出信息非常重要，客户端在获取证书前，会利用主机名与相应服务器之间建立连接，然后获得证书。
```

查看生成证书的信息的方法

```
[root@web01 key]# openssl x509 -text -in server.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 9747921528343358470 (0x874792fbbb49ec06)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=CN, ST=BJ, L=BJ, O=nmtui.com, OU=ops, CN=blog.nmtui.com/emailAddress=admin@nmtui.com
```

CA目签发证书实际创建过程

```
cd /etc/pki/CA/private/         # 进入到私钥保存目录中
(umask 077;openssl genrsa -out ./cakey.pem 2048)  #  创建一个 ca 私钥文件
cd /etc/pki/CA                     # 进入到CA目签发保存目录中
openssl req -new -x509 -key private/cakey.pem -out cacert.pem # 生成自签发证书
# 说明：由于下面配置文件中定义了一些证书信息,所以默认即可。
```

- ✓ **相关配置文件参数设定**

证书颁发机构的配置文件信息设定

ca颁发机构的私钥和证书是不能随便放置的，并且需要配置私有颁发机构的配置文件

```
/etc/pki/tls/openssl.cnf
```

[CA_default] 模块参数说明：

参数	配置	官方配置说明	解释配置说明
dir	= /etc/pki/CA	Where everything is kept	创建并定义CA目录信息
certs	= $dir/certs	Where the issued certs are kept	证书文件保存目录
crl_dir	= $dir/crl	Where the issued crl are kept	证书吊销文件保存目录
database	= $dir/index.txt	database index file.	表示发过哪些证书，都要文件进行记录
new_certs_dir	= $dir/newcerts	default place for new certs.	默认新证书的存放路径
certificate	= $dir/cacert.pem	The CA certificate	定义ca机构自己的证书
serial	= $dir/serial	The current serial number	表示证书对应的序列号，一般从01开始
		the current crl	表示吊销证书

crlnumber	= $dir/crlnumber	number	对应的序列号
crl	= $dir/crl.pem	The current CRL	表示当前证书吊销列表文件
private_key	= $dir/private/cakey.pem	The private key	表示ca机构自身的私钥文件
RANDFILE	= $dir/private/.rand	private random number file	私钥随机数文件，此文件默认会自己建立
在/ctc/pki/CA的证书路径下，还需要要有certs crl newcerts三个子目录信息			

指定证书相关的有效期配置

参数	配置	官方配置说明	解释配置说明
default_days	= 365	how long to certify for	定义证书的有效期
default_crl_days	= 30	how long before next CRL	默认证书放置到吊销列表中的保存时间
default_md	= default	use public key default MD	指定单向加密算法采用的是默认的
preserve	= no	keep passed DN ordering	–

定义[req_distinguished_name]模块参数信息，即指定证书中的一些基本属性信息

参数	配置	举例配置	解释配置说明
countryName_default	= XX	= CN	国家或地区

stateOrProvinceName_default	= Default Province	= BJ	省份/州
localityName_default	= Default City	= BJ	城市名称
0.organizationName_default	= Default Company Ltd	= nmtui.com	公司组织名称
organizationalUnitName_default	=	= ops	部门名称
commonName_default	–	= blog.nmtui.com	主机名/域名
emailAddress_default	–	= admin@nmtui.com	邮件地址

1.5.5 公司中申请证书文件的流程

```
# 第一步创建私钥文件:
 (umask 077;openssl genrsa -out httpd.key 1024)        # 模拟客户端，创建web服务的私钥
# 第二步创建证书请求文件
openssl req -new -key httpd.key -out httpd.csr        # 创建向CA申请证书的请求证书
# 第三步将请求文件发送给证书颁发机构
```

1.6 WEB服务实现HTTPS访问

1.6.1 证书的创建

说明：再测试环境中，可以使用只生产的证书进行测试（使用只生产的证书在访问时会报证书不安全！），自生产证书的方法如上所示。

这是使用的时腾讯云申请的免费SSL证书进行测试。

证书获取方法:

登陆腾讯云，访问https://console.cloud.tencent.com/ssl证书管理。

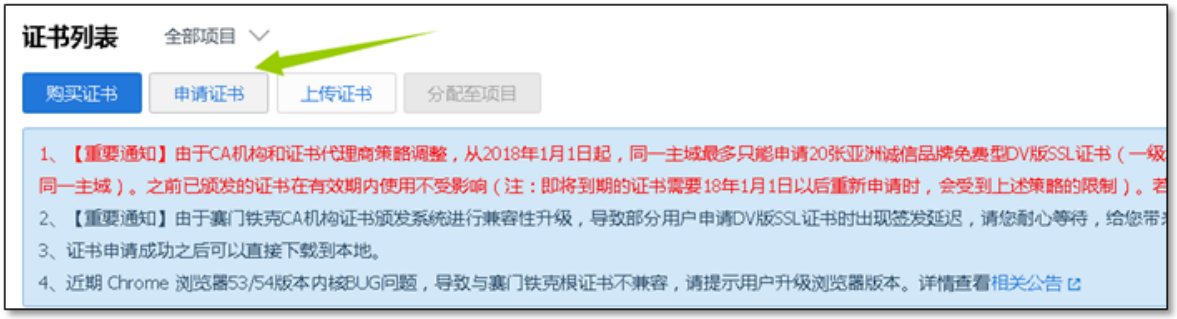


图 – 申请证书

图 – 选择《亚洲诚信》免费证书

图 – 输入域名信息

注意：域名需要为你所持有的域名

图 – 验证域名的所有权

域名验证通过后，即可下载生成的密钥

图 – 下载证书文件

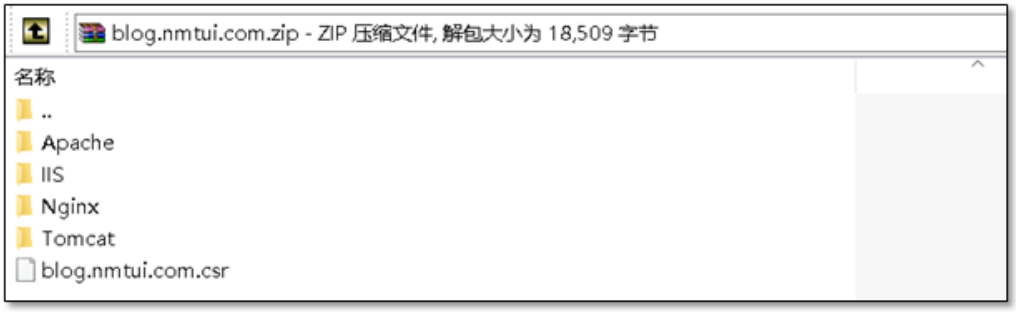


图 – 得到的证书文件

1.6.2 nginx配置https访问

nginx的搭建参考：http://www.cnblogs.com/clsn/p/8025324.html

创建证书存放目录，讲获取的证书放到这个目录

```
[root@web01 key]# mkdir -p cd /application/nginx/conf/key/
[root@web01 ~]# cd /application/nginx/conf/key/
[root@web01 key]# ll
total 8
-rw-r--r-- 1 www www 3307 Jan 18 10:48 1_blog.nmtui.com_bundle.crt
-rw-r--r-- 1 www www 1700 Jan 18 10:48 2_blog.nmtui.com.key
```

主配置文件

```
[root@web01 ~]# cat /application/nginx/conf/nginx.conf
worker_processes  1;
events {
    worker_connections  1024;
}
http {
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;
    client_max_body_size 1000M;
    client_body_buffer_size 10M;
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    include extra/blog.conf;
}
```

blog站点配置文件

```
[root@web01 ~]# cat /application/nginx/conf/extra/blog.conf
    server{
        listen       80;
        server_name  blog.nmtui.com;
```

```
        rewrite ^(.*) https://$host$1 permanent;
    }
    server {
        listen       443;
        server_name  blog.nmtui.com;
        ssl on;
        ssl_certificate /application/nginx/conf/key/1_blog.nmtui.com_bundle.crt;
        ssl_certificate_key /application/nginx/conf/key/2_blog.nmtui.com.key;
        ssl_session_timeout 5m;
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:HIGH:!aNULL:!MD5:!RC4:!DHE;
        ssl_prefer_server_ciphers on;
        location / {
            root   html/blog;
            try_files $uri $uri/ /index.php?$args;
            index  index.php index.html index.htm;
    }
       rewrite /wp-admin$ $scheme://$host$uri/ permanent;
       location ~* .*\.(php|php5)?$ {
                    root html/blog;
                    fastcgi_pass  127.0.0.1:9000;
                    fastcgi_index index.php;
                    include fastcgi.conf;
        }
        access_log  logs/access_blog.log  main;
    }
```

创建测试面页

```
[root@web01 ~]# cat >> /application/nginx/html/blog/clsn.html <
web01
https://blog.nmtui.com
EOF
```

配置完成后重启，在浏览器进访问测试

1.6.3 实现http访问自动跳转到https的方法

方法一：利用地址重写功能

```
server {
    listen  80;
    server_name blog.nmtui.com;
    rewrite ^(.*)$  https://$host$1 permanent;
}
# 说明：在https配置server基础上再添加http跳转server
```

方法二：利用error_page识别错误码信息进行跳转

```
server {
        listen       443;
        listen       80;
        server_name  blog.nmtui.com;
        ssl on;
        ssl_certificate /application/nginx/conf/key/1_blog.nmtui.com_bundle.crt;
        ssl_certificate_key /application/nginx/conf/key/2_blog.nmtui.com.key;
        location / {
            root   html/www;
            index  index.html index.htm;
```

```
        }
       error_page 497  https://$host$uri;
}
```

说明：497为内置错误码，当访问http无法处理，需要利用https处理时

1.6.4 利用nginx反向代理服务器进行http到https跳转

第一个里程碑：修改地址池信息

```
    upstream www_server_pools {
                server 10.0.0.8:443;
    }
```

第二个里程碑：修改地址池调用信息

```
    server {
        listen 443;
        server_name blog.nmtui.com;
        ssl on;
        ssl_certificate /application/nginx/conf/key/1_blog.nmtui.com_bundle.crt;
        ssl_certificate_key /application/nginx/conf/key/2_blog.nmtui.com.key;
        ssl_session_timeout 5m;
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:HIGH:!aNULL:!MD5:!RC4:!DHE;
        ssl_prefer_server_ciphers on;
        location / {
            proxy_pass https://server_pools;
        }
    }
```

第三个里程碑：定义http到https跳转配置信息

```
    server {
        listen       80;
        server_name  blog.nmtui.com;
        rewrite ^(.*)$  https://$host$1 permanent;
    }
```

1.7 附录 – ngx_http_ssl_module模块说明

ngx_http_ssl_module模块为实现 HTTPS提供了必要的支持。

此模块不是在默认情况下生成的, 在安装nginx时使用–with-http_ssl_module配置参数启用它。此模块需要OpenSSL库。详情参照：http://www.cnblogs.com/clsn/p/8025324.html#_label3

1.7.1 示例配置

为了减轻处理器负载，建议

- ✓ 设置工作进程的数量等于处理器的数量；
- ✓ 启用保持连接；
- ✓ 启用共享会话缓存；
- ✓ 禁用内置的会话缓存；
- ✓ 建议增加会话超时时间 （默认为5分钟）。

```
worker_processes auto;

http {

    ...
```

```
    server {
        listen              443 ssl;
        keepalive_timeout   70;

        ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers         AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
        ssl_certificate     /usr/local/nginx/conf/cert.pem;
        ssl_certificate_key /usr/local/nginx/conf/cert.key;
        ssl_session_cache   shared:SSL:10m;
        ssl_session_timeout 10m;

        ...
    }
```

1.7.2 SSL模块指令说明

1.7.2.1 ssl

```
Syntax:    ssl on | off;
Default:    ssl off;
Context:    http, server
```

为指定的虚拟服务器启用HTTPS协议，默认关闭。

1.7.2.2 ssl_buffer_size

```
Syntax:    ssl_buffer_size size;
Default:    ssl_buffer_size 16k;
Context:    http, server
# This directive appeared in version 1.5.9.
```

该指令用于设置用于发送数据的缓冲区的大小。

默认情况下，缓冲区大小为16k，这对应于发送大响应时的最小开销。为了将发送第一个字节的时间减少，可以使用较小的值，例如：

```
ssl_buffer_size 4k;
```

1.7.2.3 ssl_certificate

```
Syntax:    ssl_certificate file;
Default:    —
Context:    http, server
```

指定虚拟主机的 PEM 格式的**证书文件**路径 。如果除了主证书外,还要指定中间证书, 则应按以下顺序在同一文件中指定它们: 主证书首先出现, 然后是中间证书。PEM 格式的密钥可以放在同一个文件中。

从1.11.0版本开始，可以多次使用该指令来加载不同类型的证书，例如RSA和ECDSA：

```
server {
    listen              443 ssl;
    server_name         example.com;

    ssl_certificate     example.com.rsa.crt;
    ssl_certificate_key example.com.rsa.key;

    ssl_certificate     example.com.ecdsa.crt;
    ssl_certificate_key example.com.ecdsa.key;

    ...
}
```

注意：只有OpenSSL 1.0.2及以上版本，支持加载不同类型的证书。对于较旧的版本，只能使用同一类型的单个证书。

注意：使用nginx配置HTTPS多虚拟主机时，不同的主机要监听不同的地址，否则在初次访问时，SSL连接在浏览器发送HTTPs请求之前建立，而nginx不知道请求的服务器的名称。因此，它只能提供默认的服务器证书。将会导致业务的异常。详情参照：http://nginx.org/en/docs/http/configuring_https_servers.html#name_based_https_servers

1.7.2.4 ssl_certificate_key

```
Syntax:    ssl_certificate_key file;
Default:    —
Context:    http, server
```

指定虚拟主机的 PEM 格式的**密钥文件**存放路径 。

在1.7.9版本*engine:name:id* 可以指定替代密钥文件，该指令能够从OpenSSL中加载指定ID的密钥 *name*。

1.7.2.5 ssl_ciphers

```
Syntax:    ssl_ciphers ciphers;
Default:    ssl_ciphers HIGH:!aNULL:!MD5;
Context:    http, server
```

指定证书的加密方式。加密方式要是openssl可以识别的方式，例如：

```
ssl_ciphers ALL:!aNULL:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
```

所有的加密方式可以在系统中通过openssl ciphers命令查看。

选用加密方式时要注意不同版本之间的兼容性问题，详情参考http://nginx.org/en/docs/http/configuring_https_servers.html#compatibility

1.7.2.6 ssl_client_certificate

```
Syntax:    ssl_client_certificate file;
Default:    —
Context:    http, server
```

如果启用了ssl_stapling，则需要指定PEM格式的可信CA证书文件路径，用于验证客户端证书和OCSP(Online Certificate Status Protocol，在线证书状态协议)响应。

注意:使用该参数时证书列表将被发送给客户。如果不需要，可以使用ssl_trusted_certificate 指令。

1.7.2.7 ssl_crl

```
Syntax:    ssl_crl file;
Default:    —
Context:    http, server
# This directive appeared in version 0.8.7.
```

指定用于验证客户端证书的 PEM 格式指定具有吊销证书 (CRL) 的文件 。

1.7.2.8 ssl_dhparam

```
Syntax:    ssl_dhparam file;
Default:    —
Context:    http, server
# This directive appeared in version 0.7.2.
```

指定DHE加密方式的DH证书文件位置。

1.7.2.9 ssl_ecdh_curve

```
Syntax:    ssl_ecdh_curve curve;
Default:
ssl_ecdh_curve auto;
```

```
Context:    http, server
# This directive appeared in versions 1.1.0 and 1.0.6.
```

指定一个curve用于ECDHE密码。

当使用的OpenSSL 1.0.2或更高版本时，可以指定多curve（1.11.0），例如：

```
ssl_ecdh_curve prime256v1:secp384r1;
```

特殊值auto（1.11.0）指示nginx在使用OpenSSL 1.0.2或更高版本时使用OpenSSL库中内置的列表为prime256v1，或使用旧版本。

在版本1.11.0 之前, 默认curve为prime256v1。

1.7.2.10 ssl_password_file

```
Syntax:    ssl_password_file file;
Default:    —
Context:    http, server
# This directive appeared in version 1.7.3.
```

为密钥指定一个密码文件，其中每个口令都在单独的行上指定密码。密码在加载密钥时依次尝试。

示例：

```
http {
    ssl_password_file /etc/keys/global.pass;
    ...

    server {
        server_name www1.example.com;
        ssl_certificate_key /etc/keys/first.key;
    }

    server {
        server_name www2.example.com;

        # named pipe can also be used instead of a file
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

1.7.2.11 ssl_prefer_server_ciphers

```
Syntax:    ssl_prefer_server_ciphers on | off;
Default:    ssl_prefer_server_ciphers off;
Context:    http, server
```

指定在使用 SSLv3 和 TLS 协议时, 服务器密码应优先于客户端密码。

1.7.2.12 ssl_protocols

```
Syntax:    ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];
Default:    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
Context:    http, server
```

启用指定的SSL协议。

说明：

```
TLSv1.1和 TLSv1.2参数 (1.1.13、1.0.12) 仅在使用 OpenSSL 1.0.1 或更高时才起作用。
TLSv1.3参数 (1.13.0) 仅在使用用 TLSv1.3 仅在使用 OpenSSL 1.1.1 时有效。
```

1.7.2.13 ssl_session_cache

```
Syntax:    ssl_session_cache off | none | [builtin[:size]] [shared:name:size];
Default:    ssl_session_cache none;
Context:    http, server
```

设置存储会话参数的高速缓存的类型和大小。缓存可以是以下任何一种类型：

类型	类型说明
off	严格禁止使用会话缓存: nginx 会明确告诉客户端会话可能无法重用。
none	不允许使用会话缓存: nginx 告诉客户端会话可以重用, 但实际上不会将会话参数存储在缓存中。
builtin	建立 OpenSSL 的缓存：仅由一个工作进程使用。缓存大小在会话中指定。如果未给定大小, 则默认为20480个会话。 注意：使用内置缓存会导致产生内存碎片。
shared	所有工作进程之间共享的缓存：缓存大小以字节为单位指定，一兆字节可以存储大约4000个会话。每个共享缓存都应具有名称。可以在多个虚拟服务器中使用同名的缓存。

两种不同的缓存类型都可以同时使用, 例如:

```
ssl_session_cache builtin:1000 shared:SSL:10m;
```

但是, 在没有内置缓存的情况下使用共享缓存才会更有效。

1.7.2.14 ssl_session_ticket_key

```
Syntax:    ssl_session_ticket_key file;
Default:    —
Context:    http, server
# This directive appeared in version 1.5.7.
```

指定用于加密和解密TLS会话令牌的密钥文件存放位置。如果需要在多个服务器之间共享相同的密钥,则需要使用该指令。默认情况下, 使用随机生成的密钥。

如果指定了多个密钥, 则仅使用第一个密钥对 TLS 会话进行加密。密钥对时可以进行轮换的，例如：

```
ssl_session_ticket_key current.key;
ssl_session_ticket_key previous.key;
```

该文件必须包含80或48字节的随机数, 可以使用以下命令进行创建：

```
openssl rand 80 > ticket.key
```

根据文件大小的不同, AES256 (对于80字节密钥、1.11.8) 或AES128 (对于48字节密钥) 用于加密。

1.7.2.15 ssl_session_tickets

```
Syntax:    ssl_session_tickets on | off;
Default:    ssl_session_tickets on;
Context:    http, server
# This directive appeared in version 1.5.9.
```

通过TLS session tickets启用或禁用会话恢复 。详情参考：https://tools.ietf.org/html/rfc5077

1.7.2.16 ssl_session_timeout

```
Syntax:    ssl_session_timeout time;
Default:    ssl_session_timeout 5m;
Context:    http, server
```

指定客户端可以重用会话参数的时间。

1.7.2.17 ssl_stapling

```
Syntax:    ssl_stapling on | off;
Default:    ssl_stapling off;
Context:    http, server
# This directive appeared in version 1.3.7.
```

启用或禁用服务器对OSCP的响应，详情参考：https://tools.ietf.org/html/rfc4366#section-3.6

配置示例:

```
ssl_stapling on;
resolver 192.0.2.1;
```

要使 OCSP 正常工作, 需要知道服务器证书颁发者的证书

如果ssl_certificate文件不包含中间证书, 则应在ssl_trusted_certificate文件中显示服务器证书颁发者的证书。

对于 OCSP 响应方主机名的解析, 还应指定解析程序指令。

1.7.2.18 ssl_stapling_file

```
Syntax:    ssl_stapling_file file;
Default:    —
Context:    http, server
# This directive appeared in version 1.3.7.
```

设置后，将从指定的file中采取订阅的 ocsp 响应, 而不是查询在服务器证书中指定的 ocsp 应答器。

```
该文件应该是由“ openssl ocsp”命令产生的DER格式。
```

1.7.2.19 ssl_stapling_responder

```
Syntax:    ssl_stapling_responder url;
Default:    —
Context:    http, server
# This directive appeared in version 1.3.7.
```

重写在 “颁发机构信息访问” 证书扩展中指定的 OCSP 响应程序的 URL。

参考文献：https://tools.ietf.org/html/rfc5280#section-4.2.2.1

仅支持 “http://” OCSP 响应程序，例如:

```
ssl_stapling_responder http://ocsp.example.com/;
```

1.7.2.20 ssl_stapling_verify

```
Syntax:    ssl_stapling_verify on | off;
Default:    ssl_stapling_verify off;
Context:    http, server
# This directive appeared in version 1.3.7.
```

启用或禁用服务器对 OCSP 响应的验证。

要进行验证, 应使用ssl_trusted_certificate指令将服务器证书颁发者、根证书和所有中间证书的证书配置为受信任。

1.7.2.21 ssl_trusted_certificate

```
Syntax:    ssl_trusted_certificate file;
Default:    —
Context:    http, server
# This directive appeared in version 1.3.7.
```

指定一个 PEM 格式的文件 , 其中带有用于验证 ssl_stapling启用时使用的校验证书和 OCSP 进行验证。

与ssl_client_certificate设置的证书正好相反, 这些证书的列表将不会发送到客户端。

1.7.2.22 ssl_verify_client

```
Syntax:    ssl_verify_client on | off | optional | optional_no_ca;
Default:    ssl_verify_client off;
Context:    http, server
```

启用对客户端证书的验证。验证结果存储在 *$ssl_client_verify*变量中。

可选参数optional（0.8.7+）请求客户端证书、验证证书是否存在。

可选参数optional_no_ca (1.3.8, 1.2.5)请求客户端证书而不需要将签署由受信任的CA证书。这适用于nginx外部的服务运行实际证书验证的情况。证书的内容可以通过变量*$ssl_client_cert*进行访问 。

1.7.2.23 ssl_verify_depth

```
Syntax:    ssl_verify_depth number;
Default:    ssl_verify_depth 1;
Context:    http, server
```

设置客户端证书的验证深度。

1.7.3 ngx_http_ssl_module模块中常见错误处理

在ngx_http_ssl_module模块支持几种非标准错误代码, 可用于使用error_page指令进行重定向。

错误代码	产生原因
495	an error has occurred during the client certificate verification; 在客户端证书验证过程中发生错误;
496	a client has not presented the required certificate; 客户未出示所需证书;
497	a regular request has been sent to the HTTPS port. 常规请求已发送到HTTPS端口。

在对请求进行了完全分析并且变量 (如$request_uri、 $uri、$args和其他项) 可用之后, 重定向发生。

1.7.4 ngx_http_ssl_module模块中嵌入变量

ngx_http_ssl_module模块支持多个嵌入变量:

变量	变量说明
$ssl_cipher	返回用于已建立的 SSL 连接的密码字符串;
$ssl_ciphers	返回客户端支持的密码列表（1.11.7）。
$ssl_client_escaped_cert	以 PEM 格式 (urlencoded) 返回已建立的 SSL 连接 (1.13.5) 的客户端证书。
$ssl_client_cert	以建立的SSL连接的PEM格式返回客户端证书，除第一行之外的每一行都加上制表符；这是为了在proxy_set_header指令中使用; 该变量已被弃用，$ssl_client_escaped_cert应该使用该变量。
$ssl_client_fingerprint	为建立的SSL连接（1.7.1）返回客户端证书的SHA1指纹;
$ssl_client_i_dn	根据RFC 2253 (1.11.6) 返回已建立的 SSL 连接的客户端证书的“颁发者 DN” 字符串;
$ssl_client_i_dn_legacy	返回已建立的 SSL 连接的客户端证书的 “颁发者 DN” 字符串; 在版本1.11.6 之前, 变量名为$ssl_client_i_dn.
$ssl_client_raw_cert	以 PEM 格式返回已建立的 SSL 连接的客户端证书;
$ssl_client_s_dn	根据RFC 2253（1.11.6），为建立的SSL连接返回客户端证书的“主题DN”字符串；
$ssl_client_s_dn_legacy	为建立的SSL连接返回客户端证书的“主题DN”字符串; 在版本1.11.6之前，变量名是$ssl_client_s_dn
$ssl_client_serial	为建立的SSL连接返回客户端证书的序列号;
$ssl_client_v_end	返回客户端证书的结束日期（1.11.7）；
$ssl_client_v_remain	返回客户端证书过期的天数（1.11.7）；
$ssl_client_v_start	返回客户端证书的开始日期（1.11.7）；
$ssl_client_verify	如果证书不存在，则返回客户端证书验证的结果：“SUCCESS”，“FAILED:reason”和“NONE”； 在版本1.11.7之前，“FAILED”结果不包含reason字符串。
$ssl_curves	返回由客户端 (1.11.7) 支持的curves 仅当使用 OpenSSL 版本1.0.2 或更高时, 才支持该变量。
$ssl_protocol	返回已建立的 SSL 连接的协议;

$ssl_server_name	通过SNI （1.7.0）返回请求的服务器名称；
$ssl_session_id	返回建立的SSL连接的会话标识符;
$ssl_session_reused	如果重新使用了 SSL 会话，则返回“r”，否则为 “.” (1.5.11)。

1.8 参考文献

[1] https://cloud.tencent.com/document/product/400/4143

[2] https://www.openssl.org

[3] http://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_ciphers

[4] https://baike.baidu.com/item/openssl/5454803?fr=aladdin

[5] http://nginx.org/en/docs/http/configuring_https_servers.html#name_based_https_servers