**数据加密、解密分为三种：**

**对称加密:**

**公钥加密**

**单向加密**

**对称加密：是为了做到机密性功能**

加密方和解密方使用同一个密钥；但是从来没有见过面的双方无法交流密钥。怎么实现密钥的交换呢？ 而且对不同的对象通信使用不同的密码。

56bit;

128bit;

256bit;

这种情况下出现了**公钥加密**算法。

每个加密解密算法都是成对出现的。公钥公开，私钥自己保存。先用对方的公钥加密，对方收到用私钥解密。用对方的公钥加密，每个用户只需要记住自己的一对密钥就可以了

公钥其实是完全从私钥中提取出来的，私钥可以取得公钥。

但这里还是有一个问题，这个位数必须非常长。最短的也都是512位，甚至2048bit

缺点就是：**加密速度极慢**！这个的用处所以就一般就是：

1、密钥交换， IKE: Internet Key Exchange

2、数字签名：身份验正，防抵赖 这部分还不是很理解

**单向加密：验证完整性**

实质上不是加密，提取数据特征码：不可逆，

单向加密特征：

雪崩效应

定长输出

怎样通过可靠的方案获取特征码？

**完整的整个过程：**

**发送方：**

**1、发送方用单向加密算法计算数据的特征码；**

**2、发送方用自己的私钥加密这段特征码，并附加在数据尾部；**

**3、发送生成一个对称密钥；**

**4、用对称密钥加密数据和加密后特征码；**

**5、发送方用接收方的公钥加密这个对称密钥，附加在密文的尾部，并发送之；**

**接收方：**

**1、用自己的私钥解密加密过的对称密码；**

**2、用密码解密密文；**

**3、对发送方的公钥解密发送方私钥加密的特征码；若果能解密证明确实是发送方**

**4、用同样的单向加密算法计算出原始数据的特征码；**

**5、比较两段特征码；**

对称加密常用算法：

DES： Data Encryption Standard 数据加密算法，已经被破解

3DES

AES: Advanced data Encryption Standard

AES256

AES192

AES512

CAST5

BLOWFISH

TOWFISH

公钥加密常用算法：

RSA: 加密/签名

DSA：签名

单向加密的常见算法：

md5: Message Degist, 定长128bit

sha1: Secure Hash Algorithm 1, 160bit

SHA256:

SHA512:

在/usr/src/linux/crypto下能看支持的加密算法

这是否就天下大同了呢？

还有什么薄弱的么？

**过程中严重依赖于对方的公钥。**

这个公钥不能确定就是对方真正的公钥。

必须找一个类似于公安局一样有公信力的机构。

公钥被第三方公正过得才可信

Digital Certificate 数字证书 类似于身份证。

每个证书里一般都有：公钥 地址 国家代码等信息

公钥机构验证，然后核准这个证，核准的过程就是第三方机构用私钥加密给别人，用第三方机构的公钥来验证是否由第三方签发。

但是怎么验证第三方机构？全球只有一个根，下面有分支机构。信任传递。

实际上可能还是有很多的。非常有**公信力**的证书还是比较贵的。

买卖证书。

还有一种方法就是自己做证书。

自己的范围内用一个证书。

实在不行还有另外的验证机制U盾之类的另一层保护机制。私钥如果在本地保存还得考虑这个问题。

这个第三方机构就叫做CA

公共CA

私有CA：

PKI: Public Key Infrastructure

公钥基础设施

PKI： 是一种框架。现代电子互联网的框架。核心就是CA

身份丢了怎么办？媒体声明作废

重新设定一个，但是怎么作废呢？

CRL：Certificate Revoke List 证书吊销列表

但国内可能大多用户没有验证

ssl:

secure socket layer

osi模型中没有加密层，在最上边第一层和第二次层中间加了半层ssl

sslv1, sslv2, sslv3

协议的实现：

openssl

TLS: Transport Layer security, tlsv1

https: 443

http: 80

http三次握手

https 三次握手，服务端证书发给客户端，证书正确通信

linux上的ssl实现：openssl

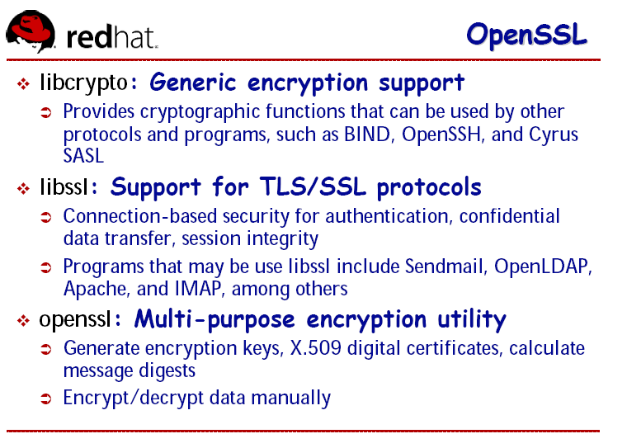
gpg rpm包签名用到

openssl由着三部分组成

1通用加密库

2 libssl 实现协议的支撑

3 openssl 多用途的加密工具 许多子命令



**openssl:可以：**

对称加密算法

非对称加密算法

单向加密算法

做一个简单的CA (专业点的：OpenCA)

既能做CA又能实现CA格式转换：x509 <--> pcks#12

默认配置文件:

**/etc/tls/pki/openssl.cnf**

还可以证书格式的转换

x509, 一种证书格式；类似于pkcs#10

pkcs#12, pkcs#7 如何将证书存储的格式

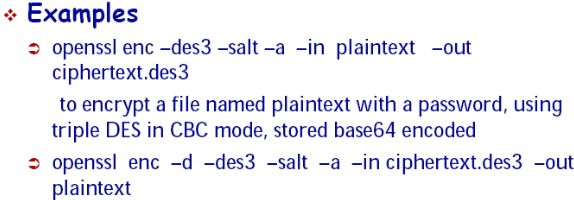
**openssl使用：**

用openssl加解密：

openssl passwd 类似于passwd命令



密码一样能干，salt一样，加密之后一样



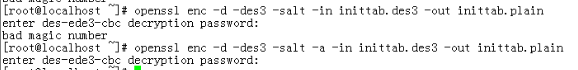


enc加密的意思

-a asc2码输入

**解密：**

**-d还原解密**



也得带-a

**bash64：**



-n 表示不带换行符

**生成密钥对：**

openssl genrsa -out key.pem 1024

rsa格式密钥对，生成的是私钥

**然后提取公钥**



再提取公钥

openssl rsa –in key.magedu –pubout –text

openssl rsa –in key.magedu –pubout 不加text就只显示密钥

openssl rsa –in key.magedu –pubout –out key.magedu.pub -out保存

或者还可以这样生成公钥

openssl genrsa 1024 > key.www cat查看

**生成权限均为644 ，最好改为600**

**这样生成的不用改：在子shell中 执行 这部分？**

**(umask 077;** openssl genrsa -out key.test 2048)

**单向加密算法应用：**

**opensll md5 inittab.plain**

**opensll sha1 inittab.plain**

**echo –n “redhat” | openssl md5**

**使用openssl 创建简易CA:**

1、先自己发一证书；

2、某主机或用户自己生成证书颁发请求；

3、递交请求；

4、服务验正信息，并签署；

配置文件：

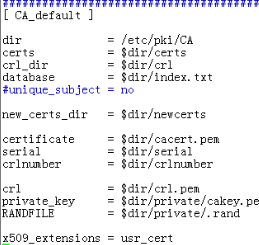
/etc/pki/tls/openssl.cnf

默认工作属性，场所：

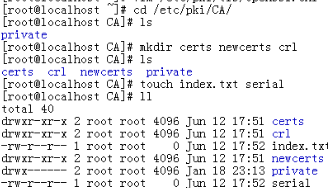
/etc/pki/CA/ 自己给自己发证

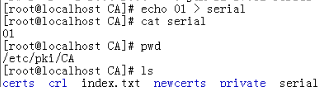
一、CA服务器的配置文件的修改；

vim /etc/pki/ tls/openssl.cnf **(先备份)**



切换目录创建配置文件中需要的目录和文件





自己给自己发证先声明密钥对：

必须在peivate下，且必须叫cakey.pem 配置文件中规定

#(umask 077;openssl genrsa –out private/cakey.pem 2048)

ls /private 下面就有了

**生成：一个自签证书**

openssl req -x509 -newkey rsa:1024 –keyout key.pem –out req.pem

-x509只有自签的时候采用

如果已经有了key不用生成可以不用再输入-newkey了

#openssl req –x509 –new –key private/cakey.pem –out cacert.pem –days 365

国家名CN 省份HA 城市ZZ 公司名MEGEDU 部门：IT 名字（服务器上必须要与浏览的网页一致）：ca.maedu.com 邮箱：

这样就生成了一个cacert.pem文件证书

**解开：**

查看证书的内容：

openssl x509 -noout -in /path/to/certificate -text

openssl x509 -noout -in cacert.pem -text

更详细的信息 ：man x509

别的主机想让发个证：

OpenSSL:

1、创建CA

自己生成一对密钥；

生成自签证书；

2、客户端

生成一对密钥；

生成证书颁发请求, .csr；

将请求发给CA；

3、CA端

签署此证书；

传送给客户端；

**CA服务器：整个过程：**

1、vim /etc/pki/tls/openssl.cnf

dir=/etc/pki/CA

2、cd /etc/pki/CA

# make certs newcerts crl

# touch index.txt

# echo 01 > serial

3、自签证书

# (umask 077; openssl genrsa -out private/cakey.pem 2048)

# openssl req -x509 -new -key private/cakey.pem -out cacert.pem -days 3650

证书申请方：

1、为某服务生成密钥：

# cd /etc/vsfptd/

# mkdir ssl

# cd ssl

# (umask 077; openssl genrsa 1024 > vsftpd.key)

# openssl req -new -key vsftpd.key -out vsftpd.csr (csr 证书申请请求的缩写)

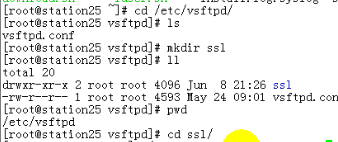
2、将此请求通过某方式传递给CA服务器

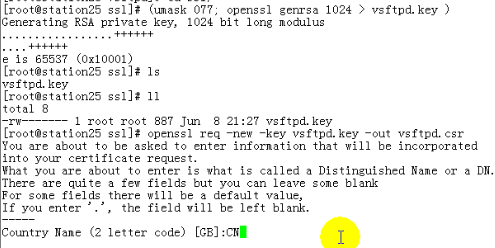
3、CA签署证书(在CA服务器上操作)

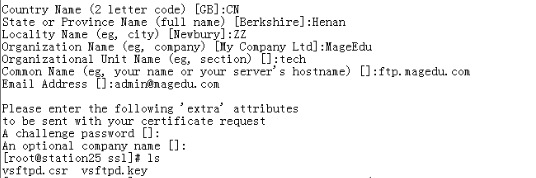
# openssl ca -in vsftpd.csr -out vsftpd.crt -days N

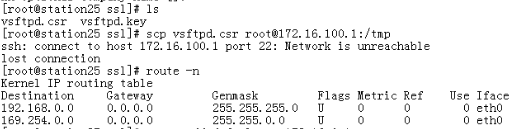
**实例：**

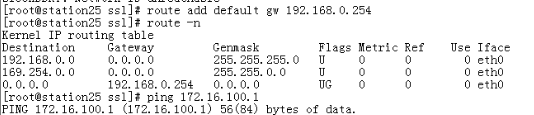
**vsftpd的ssl：**

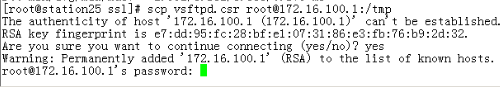










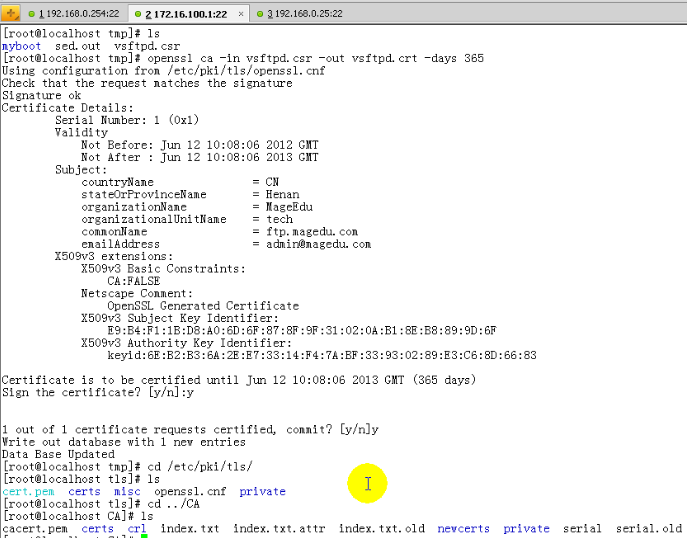


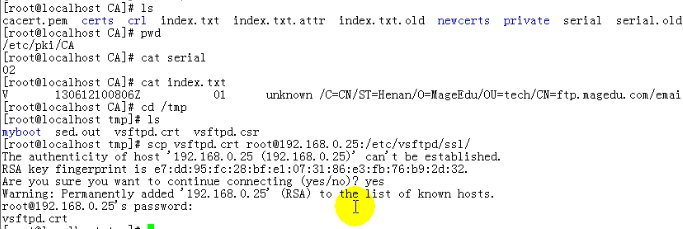
切换到172.16.100.1

cd /tmp

签一下证就行了

#openssl ca –in vsftpd.csr –out vsftpd.crt –days 365





openssl:

对称加密算法

非对称加密算法

单向加密算法

CA (OpenCA)

x509 <--> pcks#12

openssl passwd

/etc/tls/pki/openssl.cnf

openssl CA:

/etc/pki/tls/openssl.cnf

/etc/pki/CA/

1、先自己发一证书；

2、某主机或用户自己生成证书颁发请求；

3、递交请求；

4、服务验正信息，并签署；

pkcs12

Parse a PKCS#12 file and output it to a file:

openssl pkcs12 -in file.p12 -out file.pem

Output only client certificates to a file:

openssl pkcs12 -in file.p12 -clcerts -out file.pem

Don't encrypt the private key:

openssl pkcs12 -in file.p12 -out file.pem -nodes

Print some info about a PKCS#12 file:

openssl pkcs12 -in file.p12 -info -noout

Create a PKCS#12 file:

openssl pkcs12 -export -in file.pem -out file.p12 -name "My Certificate"

Include some extra certificates:

openssl pkcs12 -export -in file.pem -out file.p12 -name "My Certificate" \

-certfile othercerts.pem

Create a private key and then generate a certificate request from it:

openssl genrsa -out key.pem 1024

openssl req -new -key key.pem -out req.pem

The same but just using req:

openssl req -newkey rsa:1024 -keyout key.pem -out req.pem

查看证书的内容：

openssl x509 -noout -in /path/to/certificate -text

Generate a self signed root certificate:

openssl req -x509 -newkey rsa:1024 -keyout key.pem -out req.pem

Openssl command line:

req

-new

this option generates a new certificate request. It will prompt the user for the relevant field values. The actual fields prompted for and their maximum and minimum sizes are specified in the configuration file and any requested extensions.

If the -key option is not used it will generate a new RSA private key using information specified in the configuration file.

-pubkey

outputs the public key.

-text

prints out the certificate request in text form.

-out filename

This specifies the output filename to write to or standard output by default.

-in filename

This specifies the input filename to read a request from or standard input if this option is not specified. A request is only read if the creation options (-new and -newkey) are not specified.

-x509

this option outputs a self signed certificate instead of a certificate request. This is typically used to generate a test certificate or a self signed root CA. The extensions added to the certificate (if any) are specified in the configuration file. Unless specified using the set\_serial option 0 will be used for the serial number.

-days n

when the -x509 option is being used this specifies the number of days to certify the certificate for. The default is 30 days.

s\_client

This implements a generic SSL/TLS client which can establish a transparent connection to a remote server speaking SSL/TLS. It's intended for testing purposes only and provides only rudimentary interface functionality but internally uses mostly all functionality of the OpenSSL ssl library.

-connect host:port

This specifies the host and optional port to connect to. If not specified then an attempt is made to connect to the local host on port 4433.

-state

prints out the SSL session states.

-ssl2, -ssl3, -tls1, -no\_ssl2, -no\_ssl3, -no\_tls1

these options disable the use of certain SSL or TLS protocols. By default the initial handshake uses a method which should be compatible with all servers and permit them to use SSL v3, SSL v2 or TLS as appropriate.

Unfortunately there are a lot of ancient and broken servers in use which cannot handle this technique and will fail to connect. Some servers only work if TLS is turned off with the -no\_tls option others will only support SSL v2 and may need the -ssl2 option.

passwd - compute password hashes

-1

Use the MD5 based BSD password algorithm 1.

-salt string

Use the specified salt. When reading a password from the terminal, this implies -noverify.

-stdin

Read passwords from stdin.

Encrypt a file using triple DES in CBC mode using a prompted password:

openssl des3 -salt -in file.txt -out file.des3

Decrypt a file using a supplied password:

openssl des3 -d -salt -in file.des3 -out file.txt -k mypassword

CA服务器：

1、vim /etc/pki/tls/openssl.cnf

dir=/etc/pki/CA

2、cd /etc/pki/CA

# make certs newcerts crl

# touch index.txt

# echo 01 > serial

3、自签证书

# (umask 077; openssl genrsa -out private/cakey.pem 2048)

# openssl req -x509 -new -key private/cakey.pem -out cacert.pem -days 3650

证书申请方：

1、为某服务生成密钥：

# cd /etc/vsfptd/

# mkdir ssl

# cd ssl

# (umask 077; openssl genrsa 1024 > vsftpd.key)

# openssl req -new -key vsftpd.key -out vsftpd.csr

2、将此请求通过某方式传递给CA服务器

3、CA签署证书(在CA服务器上操作)

# openssl ca -in vsftpd.csr -out vsftpd.crt -days N

The Internet Corporation for Assigned Names and Numbers

Founded September 18, 1998

Location Marina del Rey, California, U.S.

The WHOIS system originated as a method for system administrators to obtain contact information for IP address assignments or domain name administrators.

openssl:

libcrypto

libssl

openssl

pkcs12

Parse a PKCS#12 file and output it to a file:

openssl pkcs12 -in file.p12 -out file.pem

Output only client certificates to a file:

openssl pkcs12 -in file.p12 -clcerts -out file.pem

Don't encrypt the private key:

openssl pkcs12 -in file.p12 -out file.pem -nodes

Print some info about a PKCS#12 file:

openssl pkcs12 -in file.p12 -info -noout

Create a PKCS#12 file:

openssl pkcs12 -export -in file.pem -out file.p12 -name "My Certificate"

Include some extra certificates:

openssl pkcs12 -export -in file.pem -out file.p12 -name "My Certificate" \

-certfile othercerts.pem

Create a private key and then generate a certificate request from it:

openssl req -new -key key.pem -out req.pem

The same but just using req:

Generate a self signed root certificate:

openssl req -x509 -newkey rsa:1024 -keyout key.pem -out req.pem

Openssl command line:

req

-new

this option generates a new certificate request. It will prompt the user for the relevant field values. The actual fields prompted for and their maximum and minimum sizes are specified in the configuration file and any requested extensions.

If the -key option is not used it will generate a new RSA private key using information specified in the configuration file.

-pubkey

outputs the public key.

-text

prints out the certificate request in text form.

-out filename

This specifies the output filename to write to or standard output by default.

-in filename

This specifies the input filename to read a request from or standard input if this option is not specified. A request is only read if the creation options (-new and -newkey) are not specified.

-x509

this option outputs a self signed certificate instead of a certificate request. This is typically used to generate a test certificate or a self signed root CA. The extensions added to the certificate (if any) are specified in the configuration file. Unless specified using the set\_serial option 0 will be used for the serial number.

-days n

when the -x509 option is being used this specifies the number of days to certify the certificate for. The default is 30 days.

s\_client

This implements a generic SSL/TLS client which can establish a transparent connection to a remote server speaking SSL/TLS. It's intended for testing purposes only and provides only rudimentary interface functionality but internally uses mostly all functionality of the OpenSSL ssl library.

-connect host:port

This specifies the host and optional port to connect to. If not specified then an attempt is made to connect to the local host on port 4433.

-state

prints out the SSL session states.

-ssl2, -ssl3, -tls1, -no\_ssl2, -no\_ssl3, -no\_tls1

these options disable the use of certain SSL or TLS protocols. By default the initial handshake uses a method which should be compatible with all servers and permit them to use SSL v3, SSL v2 or TLS as appropriate.

Unfortunately there are a lot of ancient and broken servers in use which cannot handle this technique and will fail to connect. Some servers only work if TLS is turned off with the -no\_tls option others will only support SSL v2 and may need the -ssl2 option.

passwd - compute password hashes

-1

Use the MD5 based BSD password algorithm 1.

-salt string

Use the specified salt. When reading a password from the terminal, this implies -noverify.

-stdin

Read passwords from stdin.

Encrypt a file using triple DES in CBC mode using a prompted password:

openssl des3 -salt -in file.txt -out file.des3

Decrypt a file using a supplied password:

openssl des3 -d -salt -in file.des3 -out file.txt -k mypassword

SRV Record

SRV Record: Also known as a Service record. An SRV record is intended to provide information on available services. A SRV record has four fields and a unique system for naming. The naming system is an underscore followed by the name of the service, followed by a period, and underscore, and then the protocol, another dot, and then the name of the domain. The four fields are.

1. Priority, just a in MX records

2. Weight, used to determine relative capacity between to SRV fields with the priority. Hits will be assigned proportionately by weight, allowing a powerful and a weak server to share appropriate loads.

3. Port, the port of the service offered

4. Hostname

For Example

\_http.\_tcp.example.com. SRV 10 5 80. www.example.com

TSIG (Transaction SIGnature) is a computer networking protocol defined in RFC 2845. It is used primarily by the Domain Name System (DNS) to provide a means of authenticating updates to a Dynamic DNS database, although it can also be used between servers and for regular queries. TSIG uses shared secret keys and one-way hashing to provide a cryptographically secure means of identifying each endpoint of a connection as being allowed to make or respond to a DNS update.