Дополнительная информация по использованию openssl

openssl может использоваться во множестве случаев и умеет выполнять следующие задачи:

• Создавать и управлять ключами RSA и DSA — команды rsa, dsa, dsaparam.

• Создавать сертификаты формата x509, запросы на сертификацию, восстановление — команды x509, req, verify, ca, crl, pks12, pks7.

• Зашифровывать данные с помощью симметрического или асимметрического шифрования — команды enc, rsautl.

• Высчитывать хеши различных типов — команда dgst.

• Работа с S/MIME — команда s/mime.

• Проверка работы серверов и клиентов ssl — команды s_client, s_server.

Для создания rsa-ключей используется команда genrsa:

openssl genrsa [-out file] [-des | -des3 | -idea] [-rand file] [bits]

Команда genrsa создает секретный ключ длиной bits в формате PEM, шифрует его одним из алгоритмов: des (56 бит), des3 (3-й des 168 бит) или idea (128 бит). Опция -out говорит программе, что вывод нужно осуществлять не в stdout, а в файл file (опция -out присутствует во множестве других компонентов openssl и используется аналогичным образом для указания выходного файла). Опция -rand указывает на файл[ы] (разделенные «:»), из которых будут считываться данные для установки seed (зерна) генератора случайных чисел.

Пример генерации 4096 битового секретного ключа RSA:

openssl genrsa -out /etc/openssl/key.pem -des3 -rand /var/log/messages 4096

Generating RSA private key

.....++*...++++++++*

Enter PEM passphrase:

Verify PEM passphrase:

После этого секретный ключ зашифровывается и записывается в файл (в текстовом виде). В начале ключа указывается алгоритм шифрования. Для создания публичного ключа rsa на основе секретного используется команда openssl rsa. Даная команда имеет следующий формат:

openssl rsa -in filename [-out file] [-des | -des3 |-idea] [-check] [-pubout]

Утилита openssl rsa способна изменять пароль и алгоритм шифрования секретного ключа, будучи вызвана с параметром -in и -out. Если применить параметр -pubout, то в указанный файл -out будет записан публичный ключ, вычисленный на основе -in секретного. Например, создание публичного ключа на основании секретного:

openssl rsa -in /etc/openssl/key.pem -out /etc/openssl/pubkey.pem -pubout

Изменение пароля и алгоритма шифрования секретного ключа с des3 на idea:

openssl rsa -in /etc/openssl/key.pem -out /etc/openssl/key1.pem -idea

Для создания ключей DSA используется утилита openssl gendsa, аналогичная genrsa, но есть два отличия: во-первых, для ключей DSA нельзя прямо указывать длину в битах и, во-вторых, ключи DSA могут генерироваться согласно некоторым параметрам, записанным в файл paramfile утилитой openssl dsaparam. dsaparam имеет следующий формат:

openssl dsaparam [-rand file{s}] [-C] [-genkey] [-out file] numbits

где numbits — длина желаемого ключа, -C заставляет dsaparam вывести на stdout код на СИ для программной генерации DSA на основе необходимых параметров, а опция -genkey говорит, что в выходной файл, наряду с параметрами, дополнительно записывается созданный секретный ключ DSA, но нельзя его сразу же зашифровать, поэтому удобнее воспользоваться утилитой openssl gendsa, которая имеет схожий синтаксис с командой genrsa, но вместо числа бит указывается файл параметров, созданный dsaparam:

openssl gendsa -out /etc/openssl/dsakey.pem -rand /boot/vmlinuz -idea paramfile

Enter PEM passphrase:

Verify PEM passphrase:

Для управления ключами dsa используется программа openssl dsa, которая абсолютно аналогична (в параметрах) утилите openssl rsa. Пример генерации публичного ключа DSA:

openssl dsa -in /etc/openssl/dsakey.pem -out /etc/openssl/pubdsakey.pem -pubout

Для выполнения симметричного шифрования используется утилита openssl enc -cipher или ее сокращенная запись openssl cipher, где cipher — это одно из символических имен симметрических шифров. Наиболее популярными являются следующие: base-64 (преобразование в текстовый вид), bf (blowfish — 128 бит), des (56 бит), des3 (168 бит), rc4 (128 бит), rc5 (128 бит), rc2 и idea (128 бит). Для указания входного и выходного файлов используется опции -in и -out соответственно.

Для расшифровки зашифрованных данных примените openssl cipher с опцией -d (алгоритм шифрования и дешифрования должен совпадать!), а для одновременной обработки данных base64 можно воспользоваться опцией -a. Шифрование по умолчанию происходит с подмешиванием, т.е. для выбора алгоритма подмешивания используется случайная соль (salt), в этом случае, если вы шифруете один и тот же файл в разное время одним и тем же алгоритмом и паролем, то результаты скорее всего будут разными (это затрудняет атаку по словарю). Также по умолчанию используется cbc-режим алгоритмов, когда ключ меняется в течение всего сеанса работы согласно передаваемым данным, этот прием очень сильно затрудняет брутфорс, т.к. атакующему сложно поймать нужный момент времени. Примеры:

* зашифруем файл, используя алгоритм des3

openssl des3 -in file -out file.des3

* расшифруем полученный файл

openssl des3 -d -in file.des3 -out file

* зашифруем файл, используя алгоритм blowfish(bf), и

* закодируем base64

openssl bf -a -in file -out file.bf64

* теперь расшифруем его и обработаем сразу же base64

openssl bf -a -d -in file.bf64 -out file

Для вычисления хешей (их еще называют отпечатками или контрольными суммами) используется команда

openssl dgst -hashalg

или краткая форма openssl hashalg (первая команда может также выполнять манипуляции с ЭЦП, но об этом далее). Обычное использование данной команды таково:

openssl hashalg [-c] file[s]

Вычисляется хеш-сообщения фиксированной длины в виде одной строки или, если указана опция -c, строки, разделенной на пары HEX-чисел двоеточием. Из алгоритмов хеширования могут применяться следующие: md2 (128 бит), md4(128 бит), md5 (128 бит), mdc2 (128 бит), sha (160 бит), sha1 (160 бит), ripemd160 (160 бит). Примеры:

* вычислим md5-хеш файла:

openssl md5 -c file

MD5(file)= 81:fd:20:ff:db:06:d5:2d:c3:55:b5:7d:3f:37:ac:94

* а теперь SHA1-хеш этого же файла:

openssl sha1 file

SHA1(file)= 13f2b3abd8a7add2f3025d89593a0327a8eb83af

Утилита openssl dgst может использоваться для подписывания сообщения секретным ключом и проверки ЭЦП публичным ключом. Для этого используется следующий синтаксис:

openssl dgst -sign private_key -out signature -hashalg file[s]

Подписывание file с помощью секретного ключа "private_key", используя алгоритм хеширования "hasalg" (обычно применяются sha1 или md5).

openssl dgst -signature signature -verify public_key file[s]

Проверка подписи в "file", используя публичный ключ "public_key" и ЭЦП "signature". Данная программа выводит «Verification OK» при правильной подписи или «Verification Failure» в любом другом случае. Учтите, что ЭЦП в таком случае хранится отдельно от файла, который ею подписан.

Для шифрования и дешифрования RSA алгоритмом используется программа rsautl. Данная утилита имеет также возможность подписывать и проверять подпись сообщений (однако работать все равно приходится с хешем сообщения, т.к. подписывать можно только небольшой объем данных, по этой причине лучше применять openssl dgst). Для шифрования/дешифрования используется следующий синтаксис:

openssl rsautl -in file -out file.cr -keyin pubkey.pem -pubin -encrypt

(Шифрование "file" с использованием публичного ключа "pubkey.pem")

openssl rsautl -in file.cr -out file -keyin secretkey.pem -decrypt

(Дешифрование "file.cr" с использованием секретного ключа "secretkey.pem")

Openssl имеет возможность генерировать сертификаты, управлять ЭЦП и шифрованием с помощью сертификатов.

Сертификат содержит публичный ключ, подписанный одним из корневых доверенных центров сертификации (или комплементарным секретным ключом), данные об организации, выдавшей сертификат и в некоторых случаях зашифрованный секретный ключ, а также отпечаток (хеш) публичного ключа. Сертификаты имеют время действия, по окончанию которого они автоматически считаются недействительными, иерархия сертификатов обычно строится на основании сети доверия (бывают довольно длинные цепочки сертификатов, ведущие к доверенному ключу из root CA). Таким образом, сертификат — это полный комплекс системы асимметричного шифрования, предоставляющий гораздо больше возможностей, чем сами по себе ключи (а также являющийся

более защищенной системой). Основным привлекательным моментом сертификата является возможность записи в него информации об организации, этот ключ выдавшей.

Общее содержание сертификатов определено стандартом x509, в то время как форматы записей сертификатов могут внести некоторую путаницу. Openssl по умолчанию использует формат PKCS#10, Microsoft использует по умолчанию формат PKCS#12, формат PKCS#7 используется для запросов на сертификацию к CA (центр сертификации) и не может содержать секретного ключа, также для этой цели может использоваться DER закодированный сертификат (DER-кодирование подобно кодированию base64, но имеет специальное назначение для использования в криптографических системах) также без секретного ключа. Учтите, что при использовании DER-формата убираются маркеры начала и конца сертификата, а его содержимое кодируется base64, поэтому в файле DER можно хранить только один сертификат, с другой стороны DER сертификаты поддерживаются MS (стандартное расширение .cer), поэтому иногда бывает нужно преобразовать сертификаты из одного формата в другой (имеется ввиду PEM или DER):

PEM-->DER: openssl x509 -inform PEM -in cert.pem -outform DER -out cert.cer

DER-->PEM: openssl x509 -inform DER -in cert.cer -outform PEM -out cert.pem

Таким же образом можно конвертировать и ключи асимметричного шифрования (используя утилиты rsa или dsa).

Проще говоря: клиент создает пару ключей (секретный/публичный) и отправляет свой запрос на сертификат в центр сертификации. В центре сертификации обрабатывается запрос клиента (запрос на сертификацию), и сертификат клиента подписывается секретным ключом центра сертификации. Клиент, имея публичный ключ центра сертификации, проверяет подлинность подписи и может далее использовать свой сертификат. Для организации можно предложить следующее решение: на сервере генерируется запрос на сертификацию и отправляется к некоему доверенному центру сертификации (который будет известен всем клиентам и персоналу данной организации); получается сертификат организации, который можно использовать при создании сертификатов клиентов. Последние создаются так: клиент посылает запрос на выдачу сертификата; сервер создает сертификат клиента и подписывает его сертификатом организации; клиент получает сертификат клиента и сертификат организации; после проверки достоверности ключа организации (предполагается, что клиент доверяет CA, которым был подписан сертификат организации) проверяется достоверность сертификата клиента. После такой операции клиент будет точно уверен, что получил сертификат от данной организации, и может его использовать для работы с ней. По такой схеме построены все центры выдачи сертификатов(правда зачастую сертификат организации бывает подписан самим собой, что требует от клиента добавить сертификат организации к доверенным, а в первой схеме сертификат организации принадлежит к группе промежуточных центров сертификации, и этот случай предпочтительнее с точки зрения безопасности и удобства клиента, но требует больше работы от администратора).

Для создания сертификата используется инструмент openssl req. Требуется конфигурационный файл, который имеет следующий формат:

[req]

Секция основных опций

default_bits = 2048

Число бит

default_keyfile = keyfile.pem

Имя ключа, используемого для сертификата

distinguished_name = req_distinguished_name

DN организации, выдавшей сертификат

prompt = no

Брать параметры из конфига неинтерактивный режим

[req_distinguished_name]

DN организации

CN=RU

Страна

ST=MO

Область

L=Moscow

Город

O=BMSTU

Название организации

OU=RK6

Название отделения

CN=Your personal certificate

Имя для сертификата (персоны, получающей сертификат)

emailAddress=certificate@maildomain.ru

Эл. почта организации

Если не указывать prompt no, то значения для параметров будут считаны в интерактивном режиме (то бишь с клавиатуры), а значения параметров будут являться подсказками при вводе данных. При интерактивном режиме можно указывать значения по умолчанию, а также минимальное и максимальное значения для параметров (для строковых параметров устанавливается ограничение на длину). В таком случае общий формат параметра таков:

имя = подсказка

имя_default = значение_по_умолчанию

имя_max = максимум

имя_min = минимум

Пример интерактивного файла конфигурации:

[req]

default_bits = 1024

default_keyfile = privkey.pem

distinguished_name = req_distinguished_name

[req_distinguished_name]

countryName = Country Name (2 letter code)

countryName_default = RU

countryName_min = 2

countryName_max = 2

localityName = Locality Name (eg, city)

organizationName = Organization Name(eg, org)

organizationalUnitName = Organizational Unit Name (eg, section)

commonName = Common Name (eg, YOUR name)

commonName_max = 64

emailAddress = Email Address

emailAddress_max = 40

openssl req -new -newkey rsa:2048 -keyout rsa_key.pem -config cfg -out certreq.pem

Создание запроса на сертификацию (-new) на основе создаваемого секретного ключа rsa (-newkey rsa:2048), который записывается в файл -keyout(и шифруется тройным DES). Запрос на сертификацию создается на основе конфигурационного файла — config.

openssl req -x509 -new -key private_key.pem -config cfg -out selfcert.pem -days 365

Создание (-new) self-signed сертификата (-x509) для использования в качестве сертификата сервера или сертификата CA. Сертификат создается с использованием секретного ключа -key и конфигурационного файла -config. Создаваемый сертификат будет действителен в течение 365 дней (-days), опция -days не применима к запросам на сертификацию.

Для управления сертификатами x509 используется утилита openssl x509. С ее помощью можно подписать сертификат или запрос на сертификацию сертификатом CA. Также можно просмотреть содержимое сертификата в читаемой форме (DN, публичный ключ, время действия, отпечаток и.т.д.). Примеры:

openssl x509 -in cert.pem -noout -text

Просмотреть информацию о сертификате в «нормальной» форме. Для этого можно использовать дополнительные опции: -fingerprint (необходимо сочетать с одной из опций -sha1, -md5 или -mdc2), -modulus (вывод публичного ключа), -serial, -subject, -issuer (организация, выдавшая сертификат), -email, -startdate, -enddate.

openssl x509 -req -in clientreq.pem -extfile /usr/lib/ssl/openssl.cnf \-extensions /usr/lib/ssl/openssl.cnf -CA CAcert.pem -CAkey serverkey.pem \-CAcreateserial -out clientcert.pem

Подписать запрос на сертификацию (-req) файла -in, используя доверенный CA сертификат -CA и его секретный ключ -CAkey. В конечный сертификат клиента (-out), записываются дополнительные параметры сертификата 3-й версии из файла /usr/lib/ssl/openssl.cnf (конфигурационный файл по умолчанию).

openssl x509 -in CAcert.pem -addtrust sslclient -alias "myorganization CA" \-out CAtrust.pem

Преобразование сертификата -in в доверенный сертификат для использования в SSL-клиентах (sslserver — использование в качестве сертификата сервера, emailProtection — использование в качестве сертификата S/MIME).

Пример построение собственного CA с self-signed сертификатом:

1) Генерируем секретный ключ:
openssl genrsa -out CAkey.pem -rand randfile -des3 4096
2) Создаем self-signed запрос на сертификат:
openssl req -new -x509 -key CAkey.pem -out CAcert.pem -days 365 -config cfg

3) Пример скрипта, генерирующего клиентские запросы на сертификаты:

```
#!/bin/bash

dd if=/dev/random of=/tmp/.rnd count=64

RAND="/var/log/messages:/boot/vmlinuz:/tmp/.rnd"

REQ="openssl req"

X509="openssl x509"

RSA="openssl rsa"

GENRSA="openssl genrsa"

O="company"

C="RU"

ST="region"

L="city"

PURPOSES="digitalSignature, keyEncipherment"

CERTTYPE="client, email, objsign"

CA="/etc/openssl/CAcert.pem"

CAkey="/etc/openssl/CAkey.pem"

OUTDIR="/etc/openssl/clientcert/"

CN="client"

BITS=2048

DAYS=365

#Создаем секретный ключ во временной папке БЕЗ шифрования

TMP="/tmp/ssl-$$"

mkdir $TMP


if [ ! -d $OUTDIR ];then

mkdir $OUTDIR

fi
```

```
pushd $TMP > /dev/null
$GENRSA -rand $RAND -out tmp.key $BITS

# Создаем конфиг для клиента
cat > cfg <

[ req ]
default_bits = $BITS
distinguished_name = req_DN
extensions = v3_req
[ req_DN ]
countryName = "1. Country Name (2 letter code)"
countryName_default = "$C"
countryName_min = 2
countryName_max = 2
stateOrProvinceName = "2. State or Province Name (full name) "
stateOrProvinceName_default = "$ST"
localityName = "3. Locality Name (eg, city) "
localityName_default = "$L"
0.organizationName = "4. Organization Name (eg, company) "
0.organizationName_default = "$O"
organizationalUnitName = "5. Organizational Unit Name (eg, section) "
organizationalUnitName_default = "$OU"
commonName = "6. Common Name (eg, CA name) "
commonName_max = 64
commonName_default = "$CN"
emailAddress = "7. Email Address (eg, name@FQDN)"
emailAddress_max = 40
emailAddress_default = ""
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = $PURPOSES
nsCertType = $CERTTYPE
EOT
```

```
# Создаем запрос на сертификацию
$REQ -new -key tmp.key -config cfg -rand $RAND -out $CN.pem
# Этот файл лучше удалить побыстрее: мало ли чего...
rm -fr /tmp/.rnd

if [ $? -ne 0 ]; then
echo "Failed to make a certificate due to error: $?"
popd > /dev/null
rm -fr $TMP
exit $?
fi
# Подписываем сертификат сертификатом сервера
$X509 -req -in $CN.pem -CA $CA -CAkey $CAkey -CAsetserial \
-extensions -config cfg -days $DAYS -out $OUTDIR$CN.pem

chmod 0400 $OUTDIR$CN.pem
chown root:root $OUTDIR$CN.pem

# Шифруем секретный ключ
$RSA -in tmp.key -des3 -out $OUTDIR$CN-key.pem
chmod 0400 $OUTDIR$CN-key.pem
chown root:root $OUTDIR$CN-key.pem

# Выполняем заключительные действия
popd > /dev/null
rm -fr $TMP
echo -e "Generation complete, go to $OUTDIR and give to client $CN his certificate and \
\n private key (for windows users you should use openssl pkcs12 utility)"
```

Для перевода к PKCS#12 используется команда openssl pkcs12:

```
openssl pkcs12 -export -in client.pem -inkey client-key.pem -out client.p12 \
-name "Client certificate from our organization"
```

Для обратного преобразования используется синтаксис:

```
openssl pkcs12 -in client.p12 -out client.pem
```

В выходной файл записываются сертификат клиента, ca сертификат, секретный ключ клиента (его можно зашифровать опцией -des3, -idea и.т.д.). Такое поведение позволяет использовать для вывода только формат pem (маркеры здесь обязательны!). Для экспорта сертификата организации можно воспользоваться командой pkcs12 (конечно же, без параметра inkey ;), можно также обработать сертификат организации base64 и сохранить в файле .cer (openssl x509 -in CA.pem -outform DER -out CA.cer).