OpenSSL

0.9.7

Applications

Table of Contents
Applications

ASN1PARSE

asn1parse - ASN.1 parsing tool

SYNOPSIS

openssl asn1parse [**-inform PEM|DER**] [**-in filename**] [**-out filename**] [**-noout**] [**-offset number**] [**-length number**] [**-i**] [**-oid filename**] [**-strparse offset**]

DESCRIPTION

The **asn1parse** command is a diagnostic utility that can parse ASN.1 structures. It can also be used to extract data from ASN.1 formatted data.

OPTIONS

-inform DER|PEM
the input format. **DER** is binary format and **PEM** (the default) is base64 encoded.

-in filename
the input file, default is standard input

-out filename
output file to place the DER encoded data into. If this option is not present then no data will be output. This is most useful when combined with the **-strparse** option.

-noout
don‘t output the parsed version of the input file.

-offset number
starting offset to begin parsing, default is start of file.

-length number
number of bytes to parse, default is until end of file.

-i
indents the output according to the "depth" of the structures.

-oid filename
a file containing additional OBJECT IDENTIFIERs (OIDs). The format of this file is described in the NOTES section below.

-strparse offset
parse the contents octets of the ASN.1 object starting at **offset**. This option can be used multiple times to "drill down" into a nested structure.

OUTPUT

The output will typically contain lines like this:

```
  0:d=0  hl=4 l= 681 cons: SEQUENCE
```

.....

```
  229:d=3  hl=3 l= 141 prim: BIT STRING
  373:d=2  hl=3 l= 162 cons: cont [ 3 ]
  376:d=3  hl=3 l= 159 cons: SEQUENCE
  379:d=4  hl=2 l=  29 cons: SEQUENCE
  381:d=5  hl=2 l=   3 prim: OBJECT            :X509v3 Subject Key Identifier
  386:d=5  hl=2 l=  22 prim: OCTET STRING
  410:d=4  hl=2 l= 112 cons: SEQUENCE
  412:d=5  hl=2 l=   3 prim: OBJECT            :X509v3 Authority Key Identifier
  417:d=5  hl=2 l= 105 prim: OCTET STRING
  524:d=4  hl=2 l=  12 cons: SEQUENCE
```

.....

This example is part of a self signed certificate. Each line starts with the offset in decimal. **d=XX** specifies the current depth. The depth is increased within the scope of any SET or SEQUENCE. **hl=XX** gives the header length (tag and length octets) of the current type. **l=XX** gives the length of the contents octets.

The **-i** option can be used to make the output more readable.

Some knowledge of the ASN.1 structure is needed to interpret the output.
In this example the BIT STRING at offset 229 is the certificate public key. The contents octets of this will contain the public key information. This can be examined using the option **-strparse 229** to yield:

```
    0:d=0  hl=3 l= 137 cons: SEQUENCE
    3:d=1  hl=3 l= 129 prim: INTEGER           :E5D21E1F5C8D208EA7A2166C7FAF9F6BDF
  135:d=1  hl=2 l=   3 prim: INTEGER           :010001
```

NOTES

If an OID is not part of OpenSSL's internal table it will be represented in numerical form (for example 1.2.3.4). The file passed to the **-oid** option allows additional OIDs to be included. Each line consists of three columns, the first column is the OID in numerical format and should be followed by white space. The second column is the "short name" which is a single word followed by white space. The final column is the rest of the line and is the "long name". **asn1parse** displays the long name. Example:

```
1.2.3.4 shortName       A long name
```

BUGS

There should be options to change the format of input lines. The output of some ASN.1 types is not well handled (if at all).

CA.pl

CA.pl - friendlier interface for OpenSSL certificate programs

SYNOPSIS

CA.pl [**-?**] [**-h**] [**-help**] [**-newcert**] [**-newreq**] [**-newreq-nodes**] [**-newca**] [**-xsign**] [**-sign**] [**-signreq**] [**-signcert**] [**-verify**] [**files**]

DESCRIPTION

The **CA.pl** script is a perl script that supplies the relevant command line arguments to the **openssl** command for some common certificate operations. It is intended to simplify the process of certificate creation and management by the use of some simple options.

COMMAND OPTIONS

?, **-h**, **-help**
: prints a usage message.

-newcert
: creates a new self signed certificate. The private key and certificate are written to the file "newreq.pem".

-newreq
: creates a new certificate request. The private key and request are written to the file "newreq.pem".

-newreq-nowdes
: is like **-newreq** except that the private key will not be encrypted.

-newca
: creates a new CA hierarchy for use with the **ca** program (or the **-signcert** and **-xsign** options). The user is prompted to enter the filename of the CA certificates (which should also contain the private key) or by hitting ENTER details of the CA will be prompted for. The relevant files and directories are created in a directory called "demoCA" in the current directory.

-pkcs12
: create a PKCS#12 file containing the user certificate, private key and CA certificate. It expects the user certificate and private key to be in the file "newcert.pem" and the CA certificate to be in the file demoCA/cacert.pem, it creates a file "newcert.p12". This command can thus be called after the **-sign** option. The PKCS#12 file can be imported directly into a browser. If there is an additional argument on the command line it will be used as the "friendly name" for the certificate (which is typically displayed in the browser list box), otherwise the name "My Certificate" is used.

-sign, **-signreq**, **-xsign**
: calls the **ca** program to sign a certificate request. It expects the request to be in the file "newreq.pem". The new certificate is written to the file "newcert.pem" except in the case of the **-xsign** option when it is written to standard output.

-signCA
: this option is the same as the **-signreq** option except it uses the configuration file section **v3_ca** and so makes the signed request a valid CA certificate. This is useful when creating intermediate CA from a root CA.

-signcert
: this option is the same as **-sign** except it expects a self signed certificate to be present in the file "newreq.pem".

-verify
: verifies certificates against the CA certificate for "demoCA". If no certificates are specified on the command line it tries to verify the file "newcert.pem".

files
: one or more optional certificate file names for use with the **-verify** command.

EXAMPLES

Create a CA hierarchy:

```
CA.pl -newca
```

Complete certificate creation example: create a CA, create a request, sign the request and finally create a PKCS#12 file containing it.

```
CA.pl -newca
CA.pl -newreq
CA.pl -signreq
CA.pl -pkcs12 "My Test Certificate"
```

DSA CERTIFICATES

Although the **CA.pl** creates RSA CAs and requests it is still possible to use it with DSA certificates and requests using the *req(1)|req(1)* command directly. The following example shows the steps that would typically be taken.

Create some DSA parameters:

```
openssl dsaparam -out dsap.pem 1024
```

Create a DSA CA certificate and private key:

```
openssl req -x509 -newkey dsa:dsap.pem -keyout cacert.pem -out cacert.pem
```

Create the CA directories and files:

```
CA.pl -newca
```

enter cacert.pem when prompted for the CA file name.

Create a DSA certificate request and private key (a different set of parameters can optionally be created first):

```
openssl req -out newreq.pem -newkey dsa:dsap.pem
```

Sign the request:

```
CA.pl -signreq
```

NOTES

Most of the filenames mentioned can be modified by editing the **CA.pl** script.

If the demoCA directory already exists then the **-newca** command will not overwrite it and will do nothing. This can happen if a previous call using the **-newca** option terminated abnormally. To get the correct behaviour delete the demoCA directory if it already exists.

Under some environments it may not be possible to run the **CA.pl** script directly (for example Win32) and the default configuration file location may be wrong. In this case the command:

```
perl -S CA.pl
```

can be used and the **OPENSSL_CONF** environment variable changed to point to the correct path of the configuration file "openssl.cnf".

The script is intended as a simple front end for the **openssl** program for use by a beginner. Its behaviour isn't always what is wanted. For more control over the behaviour of the certificate commands call the **openssl** command directly.

ENVIRONMENT VARIABLES

The variable **OPENSSL_CONF** if defined allows an alternative configuration file location to be specified, it should contain the full path to the configuration file, not just its directory.

SEE ALSO

x509(1)|x509(1), *ca(1)|ca(1)*, *req(1)|req(1)*, *pkcs12(1)|pkcs12(1)*, *config(5)|config(5)*

CA

ca - sample minimal CA application

SYNOPSIS

openssl ca [**-verbose**] [**-config filename**] [**-name section**] [**-gencrl**] [**-revoke file**] [**-crl_reason reason**] [**-crl_hold instruction**] [**-crl_compromise time**] [**-crl_CA_compromise time**] [**-subj arg**] [**-crldays days**] [**-crlhours hours**] [**-crlexts section**] [**-startdate date**] [**-enddate date**] [**-days arg**] [**-md arg**] [**-policy arg**] [**-keyfile arg**] [**-key arg**] [**-passin arg**] [**-cert file**] [**-in file**] [**-out file**] [**-notext**] [**-outdir dir**] [**-infiles**] [**-spkac file**] [**-ss_cert file**] [**-preserveDN**] [**-noemailDN**] [**-batch**] [**-msie_hack**] [**-extensions section**] [**-extfile section**] [**-engine id**]

DESCRIPTION

The **ca** command is a minimal CA application. It can be used to sign certificate requests in a variety of forms and generate CRLs it also maintains a text database of issued certificates and their status.
The options descriptions will be divided into each purpose.

CA OPTIONS

-config filename
specifies the configuration file to use.

-name section
specifies the configuration file section to use (overrides **default_ca** in the **ca** section).

-in filename
an input filename containing a single certificate request to be signed by the CA.

-ss_cert filename
a single self signed certificate to be signed by the CA.

-spkac filename
a file containing a single Netscape signed public key and challenge and additional field values to be signed by the CA. See the **SPKAC FORMAT** section for information on the required format.

-infiles
if present this should be the last option, all subsequent arguments are assumed to the the names of files containing certificate requests.

-out filename
the output file to output certificates to. The default is standard output. The certificate details will also be printed out to this file.

-outdir directory
the directory to output certificates to. The certificate will be written to a filename consisting of the serial number in hex with ".pem" appended.

-cert
the CA certificate file.

-keyfile filename
the private key to sign requests with.

-key password
the password used to encrypt the private key. Since on some systems the command line arguments are visible (e.g. Unix with the ‘ps’ utility) this option should be used with caution.

-passin arg
the key password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-verbose
this prints extra details about the operations being performed.

-notext
don‘t output the text form of a certificate to the output file.

-startdate date
this allows the start date to be explicitly set. The format of the date is YYMMDDHHMMSSZ (the same as an ASN1 UTCTime structure).

-enddate date
this allows the expiry date to be explicitly set. The format of the date is YYMMDDHHMMSSZ (the same as an ASN1 UTCTime structure).

-days arg
the number of days to certify the certificate for.

-md alg
the message digest to use. Possible values include md5, sha1 and mdc2. This option also applies to CRLs.

-policy arg
this option defines the CA "policy" to use. This is a section in the configuration file which decides which fields should be mandatory or match the CA certificate. Check out the **POLICY FORMAT** section for more information.

-msie_hack
this is a legacy option to make **ca** work with very old versions of the IE certificate enrollment control "certenr3". It used UniversalStrings for almost everything. Since the old control has various security bugs its use is strongly discouraged. The newer control "Xenroll" does not need this option.

-preserveDN
Normally the DN order of a certificate is the same as the order of the fields in the relevant policy section. When this option is set the order is the same as the request. This is largely for compatibility with the older IE enrollment control which would only accept certificates if their DNs match the order of the request. This is not needed for Xenroll.

-noemailDN
The DN of a certificate can contain the EMAIL field if present in the request DN, however it is good policy just having the e-mail set into the altName extension of the certificate. When this option is set the EMAIL field is removed from the certificate' subject and set only in the, eventually present, extensions. The **email_in_dn** keyword can be used in the configuration file to enable this behaviour.

-batch
this sets the batch mode. In this mode no questions will be asked and all certificates will be certified automatically.

-extensions section
the section of the configuration file containing certificate extensions to be added when a certificate is issued (defaults to **x509_extensions** unless the **-extfile** option is used). If no extension section is present then, a V1 certificate is created. If the extension section is present (even if it is empty), then a V3 certificate is created.

-extfile file
an additional configuration file to read certificate extensions from (using the default section unless the **-extensions** option is also used).

-engine id
specifying an engine (by it's unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

CRL OPTIONS

-gencrl
this option generates a CRL based on information in the index file.

-crldays num
the number of days before the next CRL is due. That is the days from now to place in the CRL nextUpdate field.

-crlhours num
the number of hours before the next CRL is due.

-revoke filename
a filename containing a certificate to revoke.

-crl_reason reason
revocation reason, where **reason** is one of: **unspecified**, **keyCompromise**, **CACompromise**, **affiliationChanged**, **superseded**, **cessationOfOperation**, **certificateHold** or **removeFromCRL**. The matching of **reason** is case insensitive. Setting any revocation reason will make the CRL v2.

In practive **removeFromCRL** is not particularly useful because it is only used in delta CRLs which are not currently implemented.

-crl_hold instruction

This sets the CRL revocation reason code to **certificateHold** and the hold instruction to **instruction** which must be an OID. Although any OID can be used only **holdInstructionNone** (the use of which is discouraged by RFC2459) **holdInstructionCallIssuer** or **holdInstructionReject** will normally be used.

-crl_compromise time

This sets the revocation reason to **keyCompromise** and the compromise time to **time**. **time** should be in GeneralizedTime format that is **YYYYMMDDHHMMSSZ**.

-crl_CA_compromise time

This is the same as **crl_compromise** except the revocation reason is set to **CACompromise**.

-subj arg

supersedes subject name given in the request. The arg must be formatted as */type0=value0/type1=value1/type2=...*, characters may be escaped by \ (backslash), no spaces are skipped.

-crlexts section

the section of the configuration file containing CRL extensions to include. If no CRL extension section is present then a V1 CRL is created, if the CRL extension section is present (even if it is empty) then a V2 CRL is created. The CRL extensions specified are CRL extensions and **not** CRL entry extensions. It should be noted that some software (for example Netscape) can't handle V2 CRLs.

CONFIGURATION FILE OPTIONS

The section of the configuration file containing options for **ca** is found as follows: If the **-name** command line option is used, then it names the section to be used. Otherwise the section to be used must be named in the **default_ca** option of the **ca** section of the configuration file (or in the default section of the configuration file). Besides **default_ca**, the following options are read directly from the **ca** section:

RANDFILE
preserve
msie_hack

With the exception of **RANDFILE**, this is probably a bug and may change in future releases.

Many of the configuration file options are identical to command line options. Where the option is present in the configuration file and the command line the command line value is used. Where an option is described as mandatory then it must be present in the configuration file or the command line equivalent (if any) used.

oid_file

This specifies a file containing additional **OBJECT IDENTIFIERS**. Each line of the file should consist of the numerical form of the object identifier followed by white space then the short name followed by white space and finally the long name.

oid_section

This specifies a section in the configuration file containing extra object identifiers. Each line should consist of the short name of the object identifier followed by = and the numerical form. The short and long names are the same when this option is used.

new_certs_dir

the same as the **-outdir** command line option. It specifies the directory where new certificates will be placed. Mandatory.

certificate

the same as **-cert**. It gives the file containing the CA certificate. Mandatory.

private_key

same as the **-keyfile** option. The file containing the CA private key. Mandatory.

RANDFILE

a file used to read and write random number seed information, or an EGD socket (see *RAND_egd(3)|RAND_egd(3)*).

default_days

the same as the **-days** option. The number of days to certify a certificate for.

default_startdate

the same as the **-startdate** option. The start date to certify a certificate for. If not set the current time is used.

default_enddate

the same as the **-enddate** option. Either this option or **default_days** (or the command line equivalents) must be present.

default_crl_hours default_crl_days

the same as the **-crlhours** and the **-crldays** options. These will only be used if neither command line option is present. At least one of these must be present to generate a CRL.

default_md

the same as the **-md** option. The message digest to use. Mandatory.

database

the text database file to use. Mandatory. This file must be present though initially it will be empty.

serialfile

a text file containing the next serial number to use in hex. Mandatory. This file must be present and contain a valid serial number.

x509_extensions

the same as **-extensions**.

crl_extensions

the same as **-crlexts**.

preserve

the same as **-preserveDN**

email_in_dn

the same as **-noemailDN**. If you want the EMAIL field to be removed from the DN of the certificate simply set this to 'no'. If not present the default is to allow for the EMAIL filed in the certificate's DN.

msie_hack

the same as **-msie_hack**

policy

the same as **-policy**. Mandatory. See the **POLICY FORMAT** section for more information.

nameopt, **certopt**

these options allow the format used to display the certificate details when asking the user to confirm signing. All the options supported by the **x509** utilities **-nameopt** and **-certopt** switches can be used here, except the **no_signame** and **no_sigdump** are permanently set and cannot be disabled (this is because the certificate signature cannot be displayed because the certificate has not been signed at this point).

For convenience the values **default_ca** are accepted by both to produce a reasonable output.

If neither option is present the format used in earlier versions of OpenSSL is used. Use of the old format is **strongly** discouraged because it only displays fields mentioned in the **policy** section, mishandles multicharacter string types and does not display extensions.

copy_extensions

determines how extensions in certificate requests should be handled. If set to **none** or this option is not present then extensions are ignored and not copied to the certificate. If set to **copy** then any extensions present in the request that are not already present are copied to the certificate. If set to **copyall** then all extensions in the request are copied to the certificate: if the extension is already present in the certificate it is deleted first. See the **WARNINGS** section before using this option.

The main use of this option is to allow a certificate request to supply values for certain extensions such as subjectAltName.

POLICY FORMAT

The policy section consists of a set of variables corresponding to certificate DN fields. If the value is "match" then the field value must match the same field in the CA certificate. If the value is "supplied" then it must be present. If the value is "optional" then it may be present. Any fields not mentioned in the policy section are silently deleted, unless the **-preserveDN** option is set but this can be regarded more of a quirk than intended behaviour.

SPKAC FORMAT

The input to the **-spkac** command line option is a Netscape signed public key and challenge. This will usually come from the **KEYGEN** tag in an HTML form to create a new private key. It is however possible to create SPKACs using the **spkac** utility.

The file should contain the variable SPKAC set to the value of the SPKAC and also the required DN components as name value pairs. If you need to include the same component twice then it can be preceded by a number and a '.'.

EXAMPLES

Note: these examples assume that the **ca** directory structure is already set up and the relevant files already exist. This usually involves creating a CA certificate and private key with **req**, a serial number file and an empty index file and placing them in the relevant directories.

To use the sample configuration file below the directories demoCA, demoCA/private and demoCA/newcerts would be created. The CA certificate would be copied to demoCA/cacert.pem and its private key to demoCA/private/cakey.pem. A file demoCA/serial would be created containing for example "01" and the empty index file demoCA/index.txt.

Sign a certificate request:

```
openssl ca -in req.pem -out newcert.pem
```

Sign a certificate request, using CA extensions:

```
openssl ca -in req.pem -extensions v3_ca -out newcert.pem
```

Generate a CRL

```
openssl ca -gencrl -out crl.pem
```

Sign several requests:

```
openssl ca -infiles req1.pem req2.pem req3.pem
```

Certify a Netscape SPKAC:

```
openssl ca -spkac spkac.txt
```

A sample SPKAC file (the SPKAC line has been truncated for clarity):

```
SPKAC=MIG0MGAwXDANBgkqhkiG9w0BAQEFAANLADBIAkEAn7PDhCeV/xIxUg8V70YRxK2A5
CN=Steve Test
emailAddress=steve@openssl.org
0.OU=OpenSSL Group
1.OU=Another Group
```

A sample configuration file with the relevant sections for **ca**:

```
[ ca ]
default_ca      = CA_default            # The default ca section

[ CA_default ]

dir             = ./demoCA              # top dir
database        = $dir/index.txt        # index file.
new_certs_dir   = $dir/newcerts         # new certs dir

certificate     = $dir/cacert.pem       # The CA cert
serial          = $dir/serial           # serial no file
private_key     = $dir/private/cakey.pem# CA private key
RANDFILE        = $dir/private/.rand    # random number file

default_days    = 365                   # how long to certify for
default_crl_days= 30                    # how long before next CRL
default_md      = md5                   # md to use

policy          = policy_any            # default policy
email_in_dn     = no                    # Don't add the email into cert DN
```

```
nameopt        = default_ca            # Subject name display option
certopt        = default_ca            # Certificate display option
copy_extensions = none                 # Don't copy extensions from request

[ policy_any ]
countryName            = supplied
stateOrProvinceName    = optional
organizationName       = optional
organizationalUnitName = optional
commonName             = supplied
emailAddress           = optional
```

FILES

Note: the location of all files can change either by compile time options, configuration file entries, environment variables or command line options. The values below reflect the default values.

```
/usr/local/ssl/lib/openssl.cnf - master configuration file
./demoCA                       - main CA directory
./demoCA/cacert.pem            - CA certificate
./demoCA/private/cakey.pem     - CA private key
./demoCA/serial                - CA serial number file
./demoCA/serial.old            - CA serial number backup file
./demoCA/index.txt             - CA text database file
./demoCA/index.txt.old         - CA text database backup file
./demoCA/certs                 - certificate output file
./demoCA/.rnd                  - CA random seed information
```

ENVIRONMENT VARIABLES

OPENSSL_CONF reflects the location of master configuration file it can be overridden by the **-config** command line option.

RESTRICTIONS

The text database index file is a critical part of the process and if corrupted it can be difficult to fix. It is theoretically possible to rebuild the index file from all the issued certificates and a current CRL: however there is no option to do this.

V2 CRL features like delta CRL support and CRL numbers are not currently supported.

Although several requests can be input and handled at once it is only possible to include one SPKAC or self signed certificate.

BUGS

The use of an in memory text database can cause problems when large numbers of certificates are present because, as the name implies the database has to be kept in memory.

It is not possible to certify two certificates with the same DN: this is a side effect of how the text database is indexed and it cannot easily be fixed without introducing other problems. Some S/MIME clients can use two certificates with the same DN for separate signing and encryption keys.

The **ca** command really needs rewriting or the required functionality exposed at either a command or interface level so a more friendly utility (perl script or GUI) can handle things properly. The scripts **CA.sh** and **CA.pl** help a little but not very much.

Any fields in a request that are not present in a policy are silently deleted. This does not happen if the **-preserveDN** option is used. To enforce the absence of the EMAIL field within the DN, as suggested by RFCs, regardless the contents of the request' subject the **-noemailDN** option can be used. The behaviour should be more friendly and configurable.

Cancelling some commands by refusing to certify a certificate can create an empty file.

WARNINGS

The **ca** command is quirky and at times downright unfriendly.

The **ca** utility was originally meant as an example of how to do things in a CA. It was not supposed to be used as a full blown CA itself: nevertheless some people are using it for this purpose.

The **ca** command is effectively a single user command: no locking is done on the various files and attempts to run more than one **ca** command on the same database can have unpredictable results.

The **copy_extensions** option should be used with caution. If care is not taken then it can be a security risk. For example if a certificate request contains a basicConstraints extension with CA:TRUE and the **copy_extensions** value is set to **copyall** and the user does not spot this when the certificate is displayed then this will hand the requestor a valid CA certificate.
This situation can be avoided by setting **copy_extensions** to **copy** and including basicConstraints with CA:FALSE in the configuration file. Then if the request contains a basicConstraints extension it will be ignored.
It is advisable to also include values for other extensions such as **keyUsage** to prevent a request supplying its own values.
Additional restrictions can be placed on the CA certificate itself. For example if the CA certificate has:

```
basicConstraints = CA:TRUE, pathlen:0
```

then even if a certificate is issued with CA:TRUE it will not be valid.

SEE ALSO

req(1)|req(1), *spkac(1)|spkac(1)*, *x509(1)|x509(1)*, *CA.pl(1)|CA.pl(1)*, *config(5)|config(5)*

CIPHERS

ciphers - SSL cipher display and cipher list tool.

SYNOPSIS

openssl ciphers [**-v**] [**-ssl2**] [**-ssl3**] [**-tls1**] [**cipherlist**]

DESCRIPTION

The **cipherlist** command converts OpenSSL cipher lists into ordered SSL cipher preference lists. It can be used as a test tool to determine the appropriate cipherlist.

COMMAND OPTIONS

-v

verbose option. List ciphers with a complete description of protocol version (SSLv2 or SSLv3; the latter includes TLS), key exchange, authentication, encryption and mac algorithms used along with any key size restrictions and whether the algorithm is classed as an "export" cipher. Note that without the **-v** option, ciphers may seem to appear twice in a cipher list; this is when similar ciphers are available for SSL v2 and for SSL v3/TLS v1.

-ssl3

only include SSL v3 ciphers.

-ssl2

only include SSL v2 ciphers.

-tls1

only include TLS v1 ciphers.

-h, **-?**

print a brief usage message.

cipherlist

a cipher list to convert to a cipher preference list. If it is not included then the default cipher list will be used. The format is described below.

CIPHER LIST FORMAT

The cipher list consists of one or more *cipher strings* separated by colons. Commas or spaces are also acceptable separators but colons are normally used.

The actual cipher string can take several different forms.

It can consist of a single cipher suite such as **RC4-SHA**.

It can represent a list of cipher suites containing a certain algorithm, or cipher suites of a certain type. For example **SHA1** represents all ciphers suites using the digest algorithm SHA1 and **SSLv3** represents all SSL v3 algorithms.

Lists of cipher suites can be combined in a single cipher string using the **+** character. This is used as a logical **and** operation. For example **SHA1+DES** represents all cipher suites containing the SHA1 **and** the DES algorithms.

Each cipher string can be optionally preceded by the characters **!**, **-** or **+**.

If **!** is used then the ciphers are permanently deleted from the list. The ciphers deleted can never reappear in the list even if they are explicitly stated.

If **-** is used then the ciphers are deleted from the list, but some or all of the ciphers can be added again by later options.

If **+** is used then the ciphers are moved to the end of the list. This option doesn‘t add any new ciphers it just moves matching existing ones.

If none of these characters is present then the string is just interpreted as a list of ciphers to be appended to the current preference list. If the list includes any ciphers already present they will be ignored: that is they will not moved to the end of the list.

Additionally the cipher string **@STRENGTH** can be used at any point to sort the current cipher list in order of encryption algorithm key length.

CIPHER STRINGS

The following is a list of all permitted cipher strings and their meanings.

DEFAULT

the default cipher list. This is determined at compile time and is normally **ALL:!ADH:RC4+RSA:+SSLv2:@STRENGTH**. This must be the first cipher string specified.

COMPLEMENTOFDEFAULT
the ciphers included in **ALL**, but not enabled by default. Currently this is **ADH**. Note that this rule does not cover **eNULL**, which is not included by **ALL** (use **COMPLEMENTOFALL** if necessary).

ALL
all ciphers suites except the **eNULL** ciphers which must be explicitly enabled.

COMPLEMENTOFALL
the cipher suites not enabled by **ALL**, currently being **eNULL**.

HIGH
"high" encryption cipher suites. This currently means those with key lengths larger than 128 bits.

MEDIUM
"medium" encryption cipher suites, currently those using 128 bit encryption.

LOW
"low" encryption cipher suites, currently those using 64 or 56 bit encryption algorithms but excluding export cipher suites.

EXP, **EXPORT**
export encryption algorithms. Including 40 and 56 bits algorithms.

EXPORT40
40 bit export encryption algorithms

EXPORT56
56 bit export encryption algorithms.

eNULL, **NULL**
the "NULL" ciphers that is those offering no encryption. Because these offer no encryption at all and are a security risk they are disabled unless explicitly included.

aNULL
the cipher suites offering no authentication. This is currently the anonymous DH algorithms. These cipher suites are vulnerable to a "man in the middle" attack and so their use is normally discouraged.

kRSA, **RSA**
cipher suites using RSA key exchange.

kEDH
cipher suites using ephemeral DH key agreement.

kDHr, **kDHd**
cipher suites using DH key agreement and DH certificates signed by CAs with RSA and DSS keys respectively. Not implemented.

aRSA
cipher suites using RSA authentication, i.e. the certificates carry RSA keys.

aDSS, **DSS**
cipher suites using DSS authentication, i.e. the certificates carry DSS keys.

aDH
cipher suites effectively using DH authentication, i.e. the certificates carry DH keys. Not implemented.

kFZA, **aFZA**, **eFZA**, **FZA**
ciphers suites using FORTEZZA key exchange, authentication, encryption or all FORTEZZA algorithms. Not implemented.

TLSv1, **SSLv3**, **SSLv2**
TLS v1.0, SSL v3.0 or SSL v2.0 cipher suites respectively.

DH
cipher suites using DH, including anonymous DH.

ADH
anonymous DH cipher suites.

AES

cipher suites using AES.

3DES
cipher suites using triple DES.

DES
cipher suites using DES (not triple DES).

RC4
cipher suites using RC4.

RC2
cipher suites using RC2.

IDEA
cipher suites using IDEA.

MD5
cipher suites using MD5.

SHA1, **SHA**
cipher suites using SHA1.

CIPHER SUITE NAMES

The following lists give the SSL or TLS cipher suites names from the relevant specification and their OpenSSL equivalents. It should be noted, that several cipher suite names do not include the authentication used, e.g. DES-CBC3-SHA. In these cases, RSA authentication is used.

SSL v3.0 cipher suites.

SSL_RSA_WITH_NULL_MD5	NULL-MD5
SSL_RSA_WITH_NULL_SHA	NULL-SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5	EXP-RC4-MD5
SSL_RSA_WITH_RC4_128_MD5	RC4-MD5
SSL_RSA_WITH_RC4_128_SHA	RC4-SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	EXP-RC2-CBC-MD5
SSL_RSA_WITH_IDEA_CBC_SHA	IDEA-CBC-SHA
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	EXP-DES-CBC-SHA
SSL_RSA_WITH_DES_CBC_SHA	DES-CBC-SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA	DES-CBC3-SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	Not implemented.
SSL_DH_DSS_WITH_DES_CBC_SHA	Not implemented.
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	Not implemented.
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	Not implemented.
SSL_DH_RSA_WITH_DES_CBC_SHA	Not implemented.
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	Not implemented.
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	EXP-EDH-DSS-DES-CBC-SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA	EDH-DSS-CBC-SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	EDH-DSS-DES-CBC3-SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	EXP-EDH-RSA-DES-CBC-SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA	EDH-RSA-DES-CBC-SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	EDH-RSA-DES-CBC3-SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	EXP-ADH-RC4-MD5
SSL_DH_anon_WITH_RC4_128_MD5	ADH-RC4-MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	EXP-ADH-DES-CBC-SHA
SSL_DH_anon_WITH_DES_CBC_SHA	ADH-DES-CBC-SHA
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	ADH-DES-CBC3-SHA
SSL_FORTEZZA_KEA_WITH_NULL_SHA	Not implemented.
SSL_FORTEZZA_KEA_WITH_FORTEZZA_CBC_SHA	Not implemented.
SSL_FORTEZZA_KEA_WITH_RC4_128_SHA	Not implemented.

TLS v1.0 cipher suites.

```
TLS_RSA_WITH_NULL_MD5                   NULL-MD5
TLS_RSA_WITH_NULL_SHA                   NULL-SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5          EXP-RC4-MD5
TLS_RSA_WITH_RC4_128_MD5                RC4-MD5
TLS_RSA_WITH_RC4_128_SHA                RC4-SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5      EXP-RC2-CBC-MD5
TLS_RSA_WITH_IDEA_CBC_SHA               IDEA-CBC-SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA       EXP-DES-CBC-SHA
TLS_RSA_WITH_DES_CBC_SHA                DES-CBC-SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA           DES-CBC3-SHA

TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA    Not implemented.
TLS_DH_DSS_WITH_DES_CBC_SHA             Not implemented.
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA        Not implemented.
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA    Not implemented.
TLS_DH_RSA_WITH_DES_CBC_SHA             Not implemented.
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA        Not implemented.
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA   EXP-EDH-DSS-DES-CBC-SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA            EDH-DSS-CBC-SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA       EDH-DSS-DES-CBC3-SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA   EXP-EDH-RSA-DES-CBC-SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA            EDH-RSA-DES-CBC-SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA       EDH-RSA-DES-CBC3-SHA

TLS_DH_anon_EXPORT_WITH_RC4_40_MD5      EXP-ADH-RC4-MD5
TLS_DH_anon_WITH_RC4_128_MD5            ADH-RC4-MD5
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA   EXP-ADH-DES-CBC-SHA
TLS_DH_anon_WITH_DES_CBC_SHA            ADH-DES-CBC-SHA
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA       ADH-DES-CBC3-SHA
```

AES ciphersuites from RFC3268, extending TLS v1.0

```
TLS_RSA_WITH_AES_128_CBC_SHA            AES128-SHA
TLS_RSA_WITH_AES_256_CBC_SHA            AES256-SHA

TLS_DH_DSS_WITH_AES_128_CBC_SHA         DH-DSS-AES128-SHA
TLS_DH_DSS_WITH_AES_256_CBC_SHA         DH-DSS-AES256-SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA         DH-RSA-AES128-SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA         DH-RSA-AES256-SHA

TLS_DHE_DSS_WITH_AES_128_CBC_SHA        DHE-DSS-AES128-SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA        DHE-DSS-AES256-SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA        DHE-RSA-AES128-SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA        DHE-RSA-AES256-SHA

TLS_DH_anon_WITH_AES_128_CBC_SHA        ADH-AES128-SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA        ADH-AES256-SHA
```

Additional Export 1024 and other cipher suites

Note: these ciphers can also be used in SSL v3.

```
TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA     EXP1024-DES-CBC-SHA
TLS_RSA_EXPORT1024_WITH_RC4_56_SHA      EXP1024-RC4-SHA
TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA EXP1024-DHE-DSS-DES-CBC-SHA
TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA  EXP1024-DHE-DSS-RC4-SHA
TLS_DHE_DSS_WITH_RC4_128_SHA            DHE-DSS-RC4-SHA
```

SSL v2.0 cipher suites.

```
SSL_CK_RC4_128_WITH_MD5                 RC4-MD5
SSL_CK_RC4_128_EXPORT40_WITH_MD5        EXP-RC4-MD5
```

```
SSL_CK_RC2_128_CBC_WITH_MD5             RC2-MD5
SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5    EXP-RC2-MD5
SSL_CK_IDEA_128_CBC_WITH_MD5            IDEA-CBC-MD5
SSL_CK_DES_64_CBC_WITH_MD5              DES-CBC-MD5
SSL_CK_DES_192_EDE3_CBC_WITH_MD5        DES-CBC3-MD5
```

NOTES

The non-ephemeral DH modes are currently unimplemented in OpenSSL because there is no support for DH certificates.

Some compiled versions of OpenSSL may not include all the ciphers listed here because some ciphers were excluded at compile time.

EXAMPLES

Verbose listing of all OpenSSL ciphers including NULL ciphers:

```
openssl ciphers -v 'ALL:eNULL'
```

Include all ciphers except NULL and anonymous DH then sort by strength:

```
openssl ciphers -v 'ALL:!ADH:@STRENGTH'
```

Include only 3DES ciphers and then place RSA ciphers last:

```
openssl ciphers -v '3DES:+RSA'
```

Include all RC4 ciphers but leave out those without authentication:

```
openssl ciphers -v 'RC4:!COMPLEMENTOFDEFAULT'
```

Include all chiphers with RSA authentication but leave out ciphers without encryption.

```
openssl ciphers -v 'RSA:!COMPLEMENTOFALL'
```

SEE ALSO

s_client(1)|s_client(1), *s_server(1)|s_server(1)*, *ssl(3)|ssl(3)*

HISTORY

The **COMPLENTOFALL** and **COMPLEMENTOFDEFAULT** selection options were added in version 0.9.7.

CONFIG

config - OpenSSL CONF library configuration files

DESCRIPTION

The OpenSSL CONF library can be used to read configuration files. It is used for the OpenSSL master configuration file **openssl.cnf** and in a few other places like **SPKAC** files and certificate extension files for the **x509** utility.

A configuration file is divided into a number of sections. Each section starts with a line **[section_name]** and ends when a new section is started or end of file is reached. A section name can consist of alphanumeric characters and underscores.

The first section of a configuration file is special and is referred to as the **default** section this is usually unnamed and is from the start of file until the first named section. When a name is being looked up it is first looked up in a named section (if any) and then the default section.

The environment is mapped onto a section called **ENV**.

Comments can be included by preceding them with the **#** character

Each section in a configuration file consists of a number of name and value pairs of the form **name=value**

The **name** string can contain any alphanumeric characters as well as a few punctuation symbols such as **.** **,** **;** and **_**.

The **value** string consists of the string following the **=** character until end of line with any leading and trailing white space removed.

The value string undergoes variable expansion. This can be done by including the form **$var** or **${var}**: this will substitute the value of the named variable in the current section. It is also possible to substitute a value from another section using the syntax **$section::name** or **${section::name}**. By using the form **$ENV::name** environment variables can be substituted. It is also possible to assign values to environment variables by using the name **ENV::name**, this will work if the program looks up environment variables using the **CONF** library instead of calling **getenv()** directly.

It is possible to escape certain characters by using any kind of quote or the \ character. By making the last character of a line a \ a **value** string can be spread across multiple lines. In addition the sequences **\n**, **\r**, **\b** and **\t** are recognized.

NOTES

If a configuration file attempts to expand a variable that doesn‘t exist then an error is flagged and the file will not load. This can happen if an attempt is made to expand an environment variable that doesn‘t exist. For example the default OpenSSL master configuration file used the value of **HOME** which may not be defined on non Unix systems.

This can be worked around by including a **default** section to provide a default value: then if the environment lookup fails the default value will be used instead. For this to work properly the default value must be defined earlier in the configuration file than the expansion. See the **EXAMPLES** section for an example of how to do this.

If the same variable exists in the same section then all but the last value will be silently ignored. In certain circumstances such as with DNs the same field may occur multiple times. This is usually worked around by ignoring any characters before an initial **.** e.g.

```
1.OU="My first OU"
2.OU="My Second OU"
```

EXAMPLES

Here is a sample configuration file using some of the features mentioned above.

```
# This is the default section.

HOME=/temp
RANDFILE= ${ENV::HOME}/.rnd
configdir=$ENV::HOME/config

[ section_one ]

# We are now in section one.

# Quotes permit leading and trailing whitespace
any = " any variable name "
```

```
other = A string that can \
cover several lines \
by including \\ characters

message = Hello World\n

[ section_two ]

greeting = $section_one::message
```

This next example shows how to expand environment variables safely.
Suppose you want a variable called **tmpfile** to refer to a temporary filename. The directory it is placed in can determined by the the **TEMP** or **TMP** environment variables but they may not be set to any value at all. If you just include the environment variable names and the variable doesn‘t exist then this will cause an error when an attempt is made to load the configuration file. By making use of the default section both values can be looked up with **TEMP** taking priority and /**tmp** used if neither is defined:

```
TMP=/tmp
# The above value is used if TMP isn't in the environment
TEMP=$ENV::TMP
# The above value is used if TEMP isn't in the environment
tmpfile=${ENV::TEMP}/tmp.filename
```

BUGS

Currently there is no way to include characters using the octal **\nnn** form. Strings are all null terminated so nulls cannot form part of the value.
The escaping isn‘t quite right: if you want to use sequences like **\n** you can‘t use any quote escaping on the same line.
Files are loaded in a single pass. This means that an variable expansion will only work if the variables referenced are defined earlier in the file.

SEE ALSO

x509(1)|x509(1), *req(1)|req(1)*, *ca(1)|ca(1)*

CRL

crl - CRL utility

SYNOPSIS

openssl crl [**-inform PEM|DER**] [**-outform PEM|DER**] [**-text**] [**-in filename**] [**-out filename**] [**-noout**] [**-hash**] [**-issuer**] [**-lastupdate**] [**-nextupdate**] [**-CAfile file**] [**-CApath dir**]

DESCRIPTION

The **crl** command processes CRL files in DER or PEM format.

COMMAND OPTIONS

-inform DER|PEM
: This specifies the input format. **DER** format is DER encoded CRL structure. **PEM** (the default) is a base64 encoded version of the DER form with header and footer lines.

-outform DER|PEM
: This specifies the output format, the options have the same meaning as the **-inform** option.

-in filename
: This specifies the input filename to read from or standard input if this option is not specified.

-out filename
: specifies the output filename to write to or standard output by default.

-text
: print out the CRL in text form.

-noout
: don‘t output the encoded version of the CRL.

-hash
: output a hash of the issuer name. This can be use to lookup CRLs in a directory by issuer name.

-issuer
: output the issuer name.

-lastupdate
: output the lastUpdate field.

-nextupdate
: output the nextUpdate field.

-CAfile file
: verify the signature on a CRL by looking up the issuing certificate in **file**

-CApath dir
: verify the signature on a CRL by looking up the issuing certificate in **dir**. This directory must be a standard certificate directory: that is a hash of each subject name (using **x509 -hash**) should be linked to each certificate.

NOTES

The PEM CRL format uses the header and footer lines:

```
-----BEGIN X509 CRL-----
-----END X509 CRL-----
```

EXAMPLES

Convert a CRL file from PEM to DER:

```
openssl crl -in crl.pem -outform DER -out crl.der
```

Output the text form of a DER encoded certificate:

```
openssl crl -in crl.der -text -noout
```

BUGS

Ideally it should be possible to create a CRL using appropriate options and files too.

SEE ALSO

crl2pkcs7(1)|crl2pkcs7(1), *ca(1)|ca(1)*, *x509(1)|x509(1)*

CRL2PKCS7

crl2pkcs7 - Create a PKCS#7 structure from a CRL and certificates.

SYNOPSIS

openssl crl2pkcs7 [**-inform PEM|DER**] [**-outform PEM|DER**] [**-in filename**] [**-out filename**] [**-certfile filename**] [**-nocrl**]

DESCRIPTION

The **crl2pkcs7** command takes an optional CRL and one or more certificates and converts them into a PKCS#7 degenerate "certificates only" structure.

COMMAND OPTIONS

-inform DER|PEM

This specifies the CRL input format. **DER** format is DER encoded CRL structure.**PEM** (the default) is a base64 encoded version of the DER form with header and footer lines.

-outform DER|PEM

This specifies the PKCS#7 structure output format. **DER** format is DER encoded PKCS#7 structure.**PEM** (the default) is a base64 encoded version of the DER form with header and footer lines.

-in filename

This specifies the input filename to read a CRL from or standard input if this option is not specified.

-out filename

specifies the output filename to write the PKCS#7 structure to or standard output by default.

-certfile filename

specifies a filename containing one or more certificates in **PEM** format. All certificates in the file will be added to the PKCS#7 structure. This option can be used more than once to read certificates form multiple files.

-nocrl

normally a CRL is included in the output file. With this option no CRL is included in the output file and a CRL is not read from the input file.

EXAMPLES

Create a PKCS#7 structure from a certificate and CRL:

```
openssl crl2pkcs7 -in crl.pem -certfile cert.pem -out p7.pem
```

Creates a PKCS#7 structure in DER format with no CRL from several different certificates:

```
openssl crl2pkcs7 -nocrl -certfile newcert.pem
       -certfile demoCA/cacert.pem -outform DER -out p7.der
```

NOTES

The output file is a PKCS#7 signed data structure containing no signers and just certificates and an optional CRL.

This utility can be used to send certificates and CAs to Netscape as part of the certificate enrollment process. This involves sending the DER encoded output as MIME type application/x-x509-user-cert.

The **PEM** encoded form with the header and footer lines removed can be used to install user certificates and CAs in MSIE using the Xenroll control.

SEE ALSO

pkcs7(1)|pkcs7(1)

DGST, MD5, MD4, MD2, SHA1, SHA, MDC2, RIPEMD160

dgst, md5, md4, md2, sha1, sha, mdc2, ripemd160 - message digests

SYNOPSIS

openssl dgst [**-md5**|**-md4**|**-md2**|**-sha1**|**-sha**|**-mdc2**|**-ripemd160**|**-dss1**] [**-c**] [**-d**] [**-hex**] [**-binary**] [**-out filename**] [**-sign filename**] [**-verify filename**] [**-prverify filename**] [**-signature filename**] [**file...**] [**md5**|**md4**|**md2**|**sha1**|**sha**|**mdc2**|**ripemd160**] [**-c**] [**-d**] [**file...**]

DESCRIPTION

The digest functions output the message digest of a supplied file or files in hexadecimal form. They can also be used for digital signing and verification.

OPTIONS

-c
print out the digest in two digit groups separated by colons, only relevant if **hex** format output is used.

-d
print out BIO debugging information.

-hex
digest is to be output as a hex dump. This is the default case for a "normal" digest as opposed to a digital signature.

-binary
output the digest or signature in binary form.

-out filename
filename to output to, or standard output by default.

-sign filename
digitally sign the digest using the private key in "filename".

-verify filename
verify the signature using the the public key in "filename". The output is either "Verification OK" or "Verification Failure".

-prverify filename
verify the signature using the the private key in "filename".

-signature filename
the actual signature to verify.

-rand file(s)
a file or files containing random data used to seed the random number generator, or an EGD socket (see *RAND_egd(3)*|*RAND_egd(3)*). Multiple files can be specified separated by a OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

file...
file or files to digest. If no files are specified then standard input is used.

NOTES

The digest of choice for all new applications is SHA1. Other digests are however still widely used.
If you wish to sign or verify data using the DSA algorithm then the dss1 digest must be used.
A source of random numbers is required for certain signing algorithms, in particular DSA.
The signing and verify options should only be used if a single file is being signed or verified.

DHPARAM

dhparam - DH parameter manipulation and generation

SYNOPSIS

openssl dhparam [**-inform DER|PEM**] [**-outform DER|PEM**] [**-in** *filename*] [**-out** *filename*] [**-dsaparam**] [**-noout**] [**-text**] [**-C**] [**-2**] [**-5**] [**-rand** *file(s)*] [**-engine id**] [*numbits*]

DESCRIPTION

This command is used to manipulate DH parameter files.

OPTIONS

-inform DER|PEM

This specifies the input format. The **DER** option uses an ASN1 DER encoded form compatible with the PKCS#3 DHparameter structure. The PEM form is the default format: it consists of the **DER** format base64 encoded with additional header and footer lines.

-outform DER|PEM

This specifies the output format, the options have the same meaning as the **-inform** option.

-in *filename*

This specifies the input filename to read parameters from or standard input if this option is not specified.

-out *filename*

This specifies the output filename parameters to. Standard output is used if this option is not present. The output filename should **not** be the same as the input filename.

-dsaparam

If this option is used, DSA rather than DH parameters are read or created; they are converted to DH format. Otherwise, "strong" primes (such that (p-1)/2 is also prime) will be used for DH parameter generation.

DH parameter generation with the **-dsaparam** option is much faster, and the recommended exponent length is shorter, which makes DH key exchange more efficient. Beware that with such DSA-style DH parameters, a fresh DH key should be created for each use to avoid small-subgroup attacks that may be possible otherwise.

-2, **-5**

The generator to use, either 2 or 5. 2 is the default. If present then the input file is ignored and parameters are generated instead.

-rand *file(s)*

a file or files containing random data used to seed the random number generator, or an EGD socket (see *RAND_egd(3)|RAND_egd(3)*). Multiple files can be specified separated by a OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

numbits

this option specifies that a parameter set should be generated of size *numbits*. It must be the last option. If not present then a value of 512 is used. If this option is present then the input file is ignored and parameters are generated instead.

-noout

this option inhibits the output of the encoded version of the parameters.

-text

this option prints out the DH parameters in human readable form.

-C

this option converts the parameters into C code. The parameters can then be loaded by calling the **get_dh***numbits***()** function.

-engine id

specifying an engine (by it‘s unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

WARNINGS

The program **dhparam** combines the functionality of the programs **dh** and **gendh** in previous versions of OpenSSL and SSLeay. The **dh** and **gendh** programs are retained for now but may have different purposes in future versions of OpenSSL.

NOTES

PEM format DH parameters use the header and footer lines:

```
-----BEGIN DH PARAMETERS-----
-----END DH PARAMETERS-----
```

OpenSSL currently only supports the older PKCS#3 DH, not the newer X9.42 DH.
This program manipulates DH parameters not keys.

BUGS

There should be a way to generate and manipulate DH keys.

SEE ALSO

dsaparam(1)|dsaparam(1)

HISTORY

The **dhparam** command was added in OpenSSL 0.9.5. The **-dsaparam** option was added in OpenSSL 0.9.6.

DSA

dsa - DSA key processing

SYNOPSIS

openssl dsa [**-inform PEM|DER**] [**-outform PEM|DER**] [**-in filename**] [**-passin arg**] [**-out filename**] [**-passout arg**] [**-des**] [**-des3**] [**-idea**] [**-text**] [**-noout**] [**-modulus**] [**-pubin**] [**-pubout**] [**-engine id**]

DESCRIPTION

The **dsa** command processes DSA keys. They can be converted between various forms and their components printed out. **Note** This command uses the traditional SSLeay compatible format for private key encryption: newer applications should use the more secure PKCS#8 format using the **pkcs8**

COMMAND OPTIONS

-inform DER|PEM

This specifies the input format. The **DER** option with a private key uses an ASN1 DER encoded form of an ASN.1 SEQUENCE consisting of the values of version (currently zero), p, q, g, the public and private key components respectively as ASN.1 INTEGERs. When used with a public key it uses a SubjectPublicKeyInfo structure: it is an error if the key is not DSA.

The **PEM** form is the default format: it consists of the **DER** format base64 encoded with additional header and footer lines. In the case of a private key PKCS#8 format is also accepted.

-outform DER|PEM

This specifies the output format, the options have the same meaning as the **-inform** option.

-in filename

This specifies the input filename to read a key from or standard input if this option is not specified. If the key is encrypted a pass phrase will be prompted for.

-passin arg

the input file password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-out filename

This specifies the output filename to write a key to or standard output by is not specified. If any encryption options are set then a pass phrase will be prompted for. The output filename should **not** be the same as the input filename.

-passout arg

the output file password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-des|-des3|-idea

These options encrypt the private key with the DES, triple DES, or the IDEA ciphers respectively before outputting it. A pass phrase is prompted for. If none of these options is specified the key is written in plain text. This means that using the **dsa** utility to read in an encrypted key with no encryption option can be used to remove the pass phrase from a key, or by setting the encryption options it can be use to add or change the pass phrase. These options can only be used with PEM format output files.

-text

prints out the public, private key components and parameters.

-noout

this option prevents output of the encoded version of the key.

-modulus

this option prints out the value of the public key component of the key.

-pubin

by default a private key is read from the input file: with this option a public key is read instead.

-pubout

by default a private key is output. With this option a public key will be output instead. This option is automatically set if the input is a public key.

-engine id

specifying an engine (by it's unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

NOTES

The PEM private key format uses the header and footer lines:

```
-----BEGIN DSA PRIVATE KEY-----
-----END DSA PRIVATE KEY-----
```

The PEM public key format uses the header and footer lines:

```
-----BEGIN PUBLIC KEY-----
-----END PUBLIC KEY-----
```

EXAMPLES

To remove the pass phrase on a DSA private key:

```
openssl dsa -in key.pem -out keyout.pem
```

To encrypt a private key using triple DES:

```
openssl dsa -in key.pem -des3 -out keyout.pem
```

To convert a private key from PEM to DER format:

```
openssl dsa -in key.pem -outform DER -out keyout.der
```

To print out the components of a private key to standard output:

```
openssl dsa -in key.pem -text -noout
```

To just output the public part of a private key:

```
openssl dsa -in key.pem -pubout -out pubkey.pem
```

SEE ALSO

dsaparam(1)|dsaparam(1), *gendsa(1)|gendsa(1)*, *rsa(1)|rsa(1)*, *genrsa(1)|genrsa(1)*

DSAPARAM

dsaparam - DSA parameter manipulation and generation

SYNOPSIS

openssl dsaparam [**-inform DER|PEM**] [**-outform DER|PEM**] [**-in filename**] [**-out filename**] [**-noout**] [**-text**] [**-C**] [**-rand file(s)**] [**-genkey**] [**-engine id**] [**numbits**]

DESCRIPTION

This command is used to manipulate or generate DSA parameter files.

OPTIONS

-inform DER|PEM

This specifies the input format. The **DER** option uses an ASN1 DER encoded form compatible with RFC2459 (PKIX) DSS-Parms that is a SEQUENCE consisting of p, q and g respectively. The PEM form is the default format: it consists of the **DER** format base64 encoded with additional header and footer lines.

-outform DER|PEM

This specifies the output format, the options have the same meaning as the **-inform** option.

-in filename

This specifies the input filename to read parameters from or standard input if this option is not specified. If the **numbits** parameter is included then this option will be ignored.

-out filename

This specifies the output filename parameters to. Standard output is used if this option is not present. The output filename should **not** be the same as the input filename.

-noout

this option inhibits the output of the encoded version of the parameters.

-text

this option prints out the DSA parameters in human readable form.

-C

this option converts the parameters into C code. The parameters can then be loaded by calling the **get_dsaXXX()** function.

-genkey

this option will generate a DSA either using the specified or generated parameters.

-rand file(s)

a file or files containing random data used to seed the random number generator, or an EGD socket (see *RAND_egd(3)|RAND_egd(3)*). Multiple files can be specified separated by a OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

numbits

this option specifies that a parameter set should be generated of size **numbits**. It must be the last option. If this option is included then the input file (if any) is ignored.

-engine id

specifying an engine (by it's unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

NOTES

PEM format DSA parameters use the header and footer lines:

```
-----BEGIN DSA PARAMETERS-----
-----END DSA PARAMETERS-----
```

DSA parameter generation is a slow process and as a result the same set of DSA parameters is often used to generate several distinct keys.

SEE ALSO

gendsa(1)|gendsa(1), *dsa(1)|dsa(1)*, *genrsa(1)|genrsa(1)*, *rsa(1)|rsa(1)*

ENC

enc - symmetric cipher routines

SYNOPSIS

openssl enc -ciphername [**-in filename**] [**-out filename**] [**-pass arg**] [**-e**] [**-d**] [**-a**] [**-A**] [**-k password**] [**-kfile filename**] [**-K key**] [**-iv IV**] [**-p**] [**-P**] [**-bufsize number**] [**-nopad**] [**-debug**]

DESCRIPTION

The symmetric cipher commands allow data to be encrypted or decrypted using various block and stream ciphers using keys based on passwords or explicitly provided. Base64 encoding or decoding can also be performed either by itself or in addition to the encryption or decryption.

OPTIONS

-in filename

the input filename, standard input by default.

-out filename

the output filename, standard output by default.

-pass arg

the password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-salt

use a salt in the key derivation routines. This option should **ALWAYS** be used unless compatibility with previous versions of OpenSSL or SSLeay is required. This option is only present on OpenSSL versions 0.9.5 or above.

-nosalt

don‘t use a salt in the key derivation routines. This is the default for compatibility with previous versions of OpenSSL and SSLeay.

-e

encrypt the input data: this is the default.

-d

decrypt the input data.

-a

base64 process the data. This means that if encryption is taking place the data is base64 encoded after encryption. If decryption is set then the input data is base64 decoded before being decrypted.

-A

if the **-a** option is set then base64 process the data on one line.

-k password

the password to derive the key from. This is for compatibility with previous versions of OpenSSL. Superseded by the **-pass** argument.

-kfile filename

read the password to derive the key from the first line of **filename**. This is for computability with previous versions of OpenSSL. Superseded by the **-pass** argument.

-S salt

the actual salt to use: this must be represented as a string comprised only of hex digits.

-K key

the actual key to use: this must be represented as a string comprised only of hex digits. If only the key is specified, the IV must additionally specified using the **-iv** option. When both a key and a password are specified, the key given with the **-K** option will be used and the IV generated from the password will be taken. It probably does not make much sense to specify both key and password.

-iv IV

the actual IV to use: this must be represented as a string comprised only of hex digits. When only the key is specified using the **-K** option, the IV must explicitly be defined. When a password is being specified using one of the other options, the IV is generated from this password.

-p

print out the key and IV used.

-P

print out the key and IV used then immediately exit: don‘t do any encryption or decryption.

-bufsize number
set the buffer size for I/O

-nopad
disable standard block padding

-debug
debug the BIOs used for I/O.

NOTES

The program can be called either as **openssl ciphername** or **openssl enc -ciphername**.
A password will be prompted for to derive the key and IV if necessary.
The **-salt** option should **ALWAYS** be used if the key is being derived from a password unless you want compatibility with previous versions of OpenSSL and SSLeay.
Without the **-salt** option it is possible to perform efficient dictionary attacks on the password and to attack stream cipher encrypted data. The reason for this is that without the salt the same password always generates the same encryption key. When the salt is being used the first eight bytes of the encrypted data are reserved for the salt: it is generated at random when encrypting a file and read from the encrypted file when it is decrypted.
Some of the ciphers do not have large keys and others have security implications if not used correctly. A beginner is advised to just use a strong block cipher in CBC mode such as bf or des3.
All the block ciphers normally use PKCS#5 padding also known as standard block padding: this allows a rudimentary integrity or password check to be performed. However since the chance of random data passing the test is better than 1 in 256 it isn‘t a very good test.
If padding is disabled then the input data must be a multiple of the cipher block length.
All RC2 ciphers have the same key and effective key length.
Blowfish and RC5 algorithms use a 128 bit key.

SUPPORTED CIPHERS

```
base64            Base 64

bf-cbc            Blowfish in CBC mode
bf                Alias for bf-cbc
bf-cfb            Blowfish in CFB mode
bf-ecb            Blowfish in ECB mode
bf-ofb            Blowfish in OFB mode

cast-cbc          CAST in CBC mode
cast              Alias for cast-cbc
cast5-cbc         CAST5 in CBC mode
cast5-cfb         CAST5 in CFB mode
cast5-ecb         CAST5 in ECB mode
cast5-ofb         CAST5 in OFB mode

des-cbc           DES in CBC mode
des               Alias for des-cbc
des-cfb           DES in CBC mode
des-ofb           DES in OFB mode
des-ecb           DES in ECB mode

des-ede-cbc       Two key triple DES EDE in CBC mode
des-ede           Alias for des-ede
des-ede-cfb       Two key triple DES EDE in CFB mode
des-ede-ofb       Two key triple DES EDE in OFB mode

des-ede3-cbc      Three key triple DES EDE in CBC mode
des-ede3          Alias for des-ede3-cbc
des3              Alias for des-ede3-cbc
des-ede3-cfb      Three key triple DES EDE CFB mode
```

```
des-ede3-ofb       Three key triple DES EDE in OFB mode

desx               DESX algorithm.

idea-cbc           IDEA algorithm in CBC mode
idea               same as idea-cbc
idea-cfb           IDEA in CFB mode
idea-ecb           IDEA in ECB mode
idea-ofb           IDEA in OFB mode

rc2-cbc            128 bit RC2 in CBC mode
rc2                Alias for rc2-cbc
rc2-cfb            128 bit RC2 in CBC mode
rc2-ecb            128 bit RC2 in CBC mode
rc2-ofb            128 bit RC2 in CBC mode
rc2-64-cbc         64 bit RC2 in CBC mode
rc2-40-cbc         40 bit RC2 in CBC mode

rc4                128 bit RC4
rc4-64             64 bit RC4
rc4-40             40 bit RC4

rc5-cbc            RC5 cipher in CBC mode
rc5                Alias for rc5-cbc
rc5-cfb            RC5 cipher in CBC mode
rc5-ecb            RC5 cipher in CBC mode
rc5-ofb            RC5 cipher in CBC mode
```

EXAMPLES

Just base64 encode a binary file:

```
openssl base64 -in file.bin -out file.b64
```

Decode the same file

```
openssl base64 -d -in file.b64 -out file.bin
```

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
```

Encrypt a file then base64 encode it (so it can be sent via mail for example) using Blowfish in CBC mode:

```
openssl bf -a -salt -in file.txt -out file.bf
```

Base64 decode a file then decrypt it:

```
openssl bf -d -salt -a -in file.bf -out file.txt
```

Decrypt some data using a supplied 40 bit RC4 key:

```
openssl rc4-40 -in file.rc4 -out file.txt -K 0102030405
```

BUGS

The **-A** option when used with large files doesn‘t work properly.

There should be an option to allow an iteration count to be included.

The **enc** program only supports a fixed number of algorithms with certain parameters. So if, for example, you want to use RC2 with a 76 bit key or RC4 with an 84 bit key you can‘t use this program.

GENDSA

gendsa - generate a DSA private key from a set of parameters

SYNOPSIS

openssl gendsa [**-out filename**] [**-des**] [**-des3**] [**-idea**] [**-rand file(s)**] [**-engine id**] [**paramfile**]

DESCRIPTION

The **gendsa** command generates a DSA private key from a DSA parameter file (which will be typically generated by the **openssl dsaparam** command).

OPTIONS

-des|-des3|-idea

These options encrypt the private key with the DES, triple DES, or the IDEA ciphers respectively before outputting it. A pass phrase is prompted for. If none of these options is specified no encryption is used.

-rand file(s)

a file or files containing random data used to seed the random number generator, or an EGD socket (see *RAND_egd(3)|RAND_egd(3)*). Multiple files can be specified separated by a OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

-engine id

specifying an engine (by it's unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

paramfile

This option specifies the DSA parameter file to use. The parameters in this file determine the size of the private key. DSA parameters can be generated and examined using the **openssl dsaparam** command.

NOTES

DSA key generation is little more than random number generation so it is much quicker that RSA key generation for example.

SEE ALSO

dsaparam(1)|dsaparam(1), *dsa(1)|dsa(1)*, *genrsa(1)|genrsa(1)*, *rsa(1)|rsa(1)*

GENRSA

genrsa - generate an RSA private key

SYNOPSIS

openssl genrsa [**-out filename**] [**-passout arg**] [**-des**] [**-des3**] [**-idea**] [**-f4**] [**-3**] [**-rand file(s)**] [**-engine id**] [**numbits**]

DESCRIPTION

The **genrsa** command generates an RSA private key.

OPTIONS

-out filename

the output filename. If this argument is not specified then standard output is used.

-passout arg

the output file password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-des|-des3|-idea

These options encrypt the private key with the DES, triple DES, or the IDEA ciphers respectively before outputting it. If none of these options is specified no encryption is used. If encryption is used a pass phrase is prompted for if it is not supplied via the **-passout** argument.

-F4|-3

the public exponent to use, either 65537 or 3. The default is 65537.

-rand file(s)

a file or files containing random data used to seed the random number generator, or an EGD socket (see *RAND_egd(3)|RAND_egd(3)*). Multiple files can be specified separated by a OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

-engine id

specifying an engine (by it's unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

numbits

the size of the private key to generate in bits. This must be the last option specified. The default is 512.

NOTES

RSA private key generation essentially involves the generation of two prime numbers. When generating a private key various symbols will be output to indicate the progress of the generation. A **.** represents each number which has passed an initial sieve test, **+** means a number has passed a single round of the Miller-Rabin primality test. A newline means that the number has passed all the prime tests (the actual number depends on the key size).

Because key generation is a random process the time taken to generate a key may vary somewhat.

BUGS

A quirk of the prime generation algorithm is that it cannot generate small primes. Therefore the number of bits should not be less that 64. For typical private keys this will not matter because for security reasons they will be much larger (typically 1024 bits).

SEE ALSO

gendsa(1)|gendsa(1)

NSEQ

nseq - create or examine a netscape certificate sequence

SYNOPSIS

openssl nseq [**-in filename**] [**-out filename**] [**-toseq**]

DESCRIPTION

The **nseq** command takes a file containing a Netscape certificate sequence and prints out the certificates contained in it or takes a file of certificates and converts it into a Netscape certificate sequence.

COMMAND OPTIONS

-in filename

This specifies the input filename to read or standard input if this option is not specified.

-out filename

specifies the output filename or standard output by default.

-toseq

normally a Netscape certificate sequence will be input and the output is the certificates contained in it. With the **-toseq** option the situation is reversed: a Netscape certificate sequence is created from a file of certificates.

EXAMPLES

Output the certificates in a Netscape certificate sequence

```
openssl nseq -in nseq.pem -out certs.pem
```

Create a Netscape certificate sequence

```
openssl nseq -in certs.pem -toseq -out nseq.pem
```

NOTES

The **PEM** encoded form uses the same headers and footers as a certificate:

```
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
```

A Netscape certificate sequence is a Netscape specific form that can be sent to browsers as an alternative to the standard PKCS#7 format when several certificates are sent to the browser: for example during certificate enrollment. It is used by Netscape certificate server for example.

BUGS

This program needs a few more options: like allowing DER or PEM input and output files and allowing multiple certificate files to be used.

OCSP

ocsp - Online Certificate Status Protocol utility

SYNOPSIS

openssl ocsp [**-out file**] [**-issuer file**] [**-cert file**] [**-serial n**] [**-req_text**] [**-resp_text**] [**-text**] [**-reqout file**] [**-respout file**] [**-reqin file**] [**-respin file**] [**-nonce**] [**-no_nonce**] [**-url responder_url**] [**-host host:n**] [**-path**] [**-CApath file**] [**-CAfile file**] [**-VAfile file**] [**-verify_certs file**] [**-noverify**] [**-trust_other**] [**-no_intern**] [**-no_sig_verify**] [**-no_cert_verify**] [**-no_chain**] [**-no_cert_checks**] [**-validity_period nsec**] [**-status_age nsec**]

DESCRIPTION

WARNING: this documentation is preliminary and subject to change.

The Online Certificate Status Protocol (OCSP) enables applications to determine the (revocation) state of an identified certificate (RFC 2560).

The **ocsp** command performs many common OCSP tasks. It can be used to print out requests and responses, create requests and send queries to an OCSP responder and behave like a mini OCSP server itself.

OCSP CLIENT OPTIONS

-out filename

specify output filename, default is standard output.

-issuer filename

This specifies the current issuer certificate. This option can be used multiple times. The certificate specified in **filename** must be in PEM format.

-cert filename

Add the certificate **filename** to the request. The issuer certificate is taken from the previous **issuer** option, or an error occurs if no issuer certificate is specified.

-serial num

Same as the **cert** option except the certificate with serial number **num** is added to the request. The serial number is interpreted as a decimal integer unless preceded by **0x**. Negative integers can also be specified by preceding the value by a **-** sign.

-signer filename, **-signkey filename**

Sign the OCSP request using the certificate specified in the **signer** option and the private key specified by the **signkey** option. If the **signkey** option is not present then the private key is read from the same file as the certificate. If neither option is specified then the OCSP request is not signed.

-nonce, **-no_nonce**

Add an OCSP nonce extension to a request or disable OCSP nonce addition. Normally if an OCSP request is input using the **respin** option no nonce is added: using the **nonce** option will force addition of a nonce. If an OCSP request is being created (using **cert** and **serial** options) a nonce is automatically added specifying **no_nonce** overrides this.

-req_text, **-resp_text**, **-text**

print out the text form of the OCSP request, response or both respectively.

-reqout file, **-respout file**

write out the DER encoded certificate request or response to **file**.

-reqin file, **-respin file**

read OCSP request or response file from **file**. These option are ignored if OCSP request or response creation is implied by other options (for example with **serial**, **cert** and **host** options).

-url responder_url

specify the responder URL. Both HTTP and HTTPS (SSL/TLS) URLs can be specified.

-host hostname:port, **-path pathname**

if the **host** option is present then the OCSP request is sent to the host **hostname** on port **port**. **path** specifies the HTTP path name to use or "/" by default.

-CAfile file, **-CApath pathname**

file or pathname containing trusted CA certificates. These are used to verify the signature on the OCSP response.

-verify_certs file

file containing additional certificates to search when attempting to locate the OCSP response signing certificate. Some responders omit the actual signer‘s certificate from the response: this option can be used to supply the necessary certificate in such cases.

-trust_other

the certificates specified by the **-verify_certs** option should be explicitly trusted and no additional checks will be performed on them. This is useful when the complete responder certificate chain is not available or trusting a root CA is not appropriate.

-VAfile file

file containing explicitly trusted responder certificates. Equivalent to the **-verify_certs** and **-trust_other** options.

-noverify

don‘t attempt to verify the OCSP response signature or the nonce values. This option will normally only be used for debugging since it disables all verification of the responders certificate.

-no_intern

ignore certificates contained in the OCSP response when searching for the signers certificate. With this option the signers certificate must be specified with either the **-verify_certs** or **-VAfile** options.

-no_sig_verify

don‘t check the signature on the OCSP response. Since this option tolerates invalid signatures on OCSP responses it will normally only be used for testing purposes.

-no_cert_verify

don‘t verify the OCSP response signers certificate at all. Since this option allows the OCSP response to be signed by any certificate it should only be used for testing purposes.

-no_chain

do not use certificates in the response as additional untrusted CA certificates.

-no_cert_checks

don‘t perform any additional checks on the OCSP response signers certificate. That is do not make any checks to see if the signers certificate is authorised to provide the necessary status information: as a result this option should only be used for testing purposes.

-validity_period nsec, **-status_age age**

these options specify the range of times, in seconds, which will be tolerated in an OCSP response. Each certificate status response includes a **notBefore** time and an optional **notAfter** time. The current time should fall between these two values, but the interval between the two times may be only a few seconds. In practice the OCSP responder and clients clocks may not be precisely synchronised and so such a check may fail. To avoid this the **-validity_period** option can be used to specify an acceptable error range in seconds, the default value is 5 minutes.

If the **notAfter** time is omitted from a response then this means that new status information is immediately available. In this case the age of the **notBefore** field is checked to see it is not older than **age** seconds old. By default this additional check is not performed.

OCSP SERVER OPTIONS

-index indexfile

indexfile is a text index file in **ca** format containing certificate revocation information.

If the **index** option is specified the **ocsp** utility is in responder mode, otherwise it is in client mode. The request(s) the responder processes can be either specified on the command line (using **issuer** and **serial** options), supplied in a file (using the **respin** option) or via external OCSP clients (if **port** or **url** is specified).

If the **index** option is present then the **CA** and **rsigner** options must also be present.

-CA file

CA certificate corresponding to the revocation information in **indexfile**.

-rsigner file

The certificate to sign OCSP responses with.

-rother file

Additional certificates to include in the OCSP response.

-resp_no_certs

Don‘t include any certificates in the OCSP response.

-resp_key_id

Identify the signer certificate using the key ID, default is to use the subject name.

-rkey file

The private key to sign OCSP responses with: if not present the file specified in the **rsigner** option is used.

-port portnum

Port to listen for OCSP requests on. The port may also be specified using the **url** option.

-nrequest number

The OCSP server will exit after receiving **number** requests, default unlimited.

-nmin minutes, **-ndays days**

Number of minutes or days when fresh revocation information is available: used in the **nextUpdate** field. If neither option is present then the **nextUpdate** field is omitted meaning fresh revocation information is immediately available.

OCSP Response verification.

OCSP Response follows the rules specified in RFC2560.

Initially the OCSP responder certificate is located and the signature on the OCSP request checked using the responder certificate‘s public key.

Then a normal certificate verify is performed on the OCSP responder certificate building up a certificate chain in the process. The locations of the trusted certificates used to build the chain can be specified by the **CAfile** and **CApath** options or they will be looked for in the standard OpenSSL certificates directory.

If the initial verify fails then the OCSP verify process halts with an error.

Otherwise the issuing CA certificate in the request is compared to the OCSP responder certificate: if there is a match then the OCSP verify succeeds.

Otherwise the OCSP responder certificate‘s CA is checked against the issuing CA certificate in the request. If there is a match and the OCSPSigning extended key usage is present in the OCSP responder certificate then the OCSP verify succeeds.

Otherwise the root CA of the OCSP responders CA is checked to see if it is trusted for OCSP signing. If it is the OCSP verify succeeds.

If none of these checks is successful then the OCSP verify fails.

What this effectively means if that if the OCSP responder certificate is authorised directly by the CA it is issuing revocation information about (and it is correctly configured) then verification will succeed.

If the OCSP responder is a "global responder" which can give details about multiple CAs and has its own separate certificate chain then its root CA can be trusted for OCSP signing. For example:

```
openssl x509 -in ocspCA.pem -addtrust OCSPSigning -out trustedCA.pem
```

Alternatively the responder certificate itself can be explicitly trusted with the **-VAfile** option.

NOTES

As noted, most of the verify options are for testing or debugging purposes. Normally only the **-CApath**, **-CAfile** and (if the responder is a ‘global VA’) **-VAfile** options need to be used.

The OCSP server is only useful for test and demonstration purposes: it is not really usable as a full OCSP responder. It contains only a very simple HTTP request handling and can only handle the POST form of OCSP queries. It also handles requests serially meaning it cannot respond to new requests until it has processed the current one. The text index file format of revocation is also inefficient for large quantities of revocation data.

It is possible to run the **ocsp** application in responder mode via a CGI script using the **respin** and **respout** options.

EXAMPLES

Create an OCSP request and write it to a file:

```
openssl ocsp -issuer issuer.pem -cert c1.pem -cert c2.pem -reqout req.der
```

Send a query to an OCSP responder with URL http://ocsp.myhost.com/ save the response to a file and print it out in text form

```
openssl ocsp -issuer issuer.pem -cert c1.pem -cert c2.pem \
  -url http://ocsp.myhost.com/ -resp_text -respout resp.der
```

Read in an OCSP response and print out text form:

```
openssl ocsp -respin resp.der -text
```

OCSP server on port 8888 using a standard **ca** configuration, and a separate responder certificate. All requests and responses are printed to a file.

```
openssl ocsp -index demoCA/index.txt -port 8888 -rsigner rcert.pem -CA demoCA/cac
       -text -out log.txt
```

As above but exit after processing one request:

```
openssl ocsp -index demoCA/index.txt -port 8888 -rsigner rcert.pem -CA demoCA/cac
    -nrequest 1
```

Query status information using internally generated request:

```
openssl ocsp -index demoCA/index.txt -rsigner rcert.pem -CA demoCA/cacert.pem
    -issuer demoCA/cacert.pem -serial 1
```

Query status information using request read from a file, write response to a second file.

```
openssl ocsp -index demoCA/index.txt -rsigner rcert.pem -CA demoCA/cacert.pem
    -reqin req.der -respout resp.der
```

OPENSSL

openssl - OpenSSL command line tool

SYNOPSIS

openssl *command* [*command_opts*] [*command_args*]
openssl [**list-standard-commands** | **list-message-digest-commands** | **list-cipher-commands**]
openssl no-*XXX* [*arbitrary options*]

DESCRIPTION

OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols and related cryptography standards required by them.
The **openssl** program is a command line tool for using the various cryptography functions of OpenSSL's **crypto** library from the shell. It can be used for

- Creation of RSA, DH and DSA key parameters
- Creation of X.509 certificates, CSRs and CRLs
- Calculation of Message Digests
- Encryption and Decryption with Ciphers
- SSL/TLS Client and Server Tests
- Handling of S/MIME signed or encrypted mail

COMMAND SUMMARY

The **openssl** program provides a rich variety of commands (*command* in the SYNOPSIS above), each of which often has a wealth of options and arguments (*command_opts* and *command_args* in the SYNOPSIS).
The pseudo-commands **list-standard-commands**, **list-message-digest-commands**, and **list-cipher-commands** output a list (one entry per line) of the names of all standard commands, message digest commands, or cipher commands, respectively, that are available in the present **openssl** utility.
The pseudo-command **no-***XXX* tests whether a command of the specified name is available. If no command named *XXX* exists, it returns 0 (success) and prints **no-***XXX*; otherwise it returns 1 and prints *XXX*. In both cases, the output goes to **stdout** and nothing is printed to **stderr**. Additional command line arguments are always ignored. Since for each cipher there is a command of the same name, this provides an easy way for shell scripts to test for the availability of ciphers in the **openssl** program. (**no-***XXX* is not able to detect pseudo-commands such as **quit**, **list-...-commands**, or **no-***XXX* itself.)

STANDARD COMMANDS

asn1parse|*asn1parse(1)*
: Parse an ASN.1 sequence.

ca|*ca(1)*
: Certificate Authority (CA) Management.

ciphers|*ciphers(1)*
: Cipher Suite Description Determination.

crl|*crl(1)*
: Certificate Revocation List (CRL) Management.

crl2pkcs7|*crl2pkcs7(1)*
: CRL to PKCS#7 Conversion.

dgst|*dgst(1)*
: Message Digest Calculation.

dh
: Diffie-Hellman Parameter Management. Obsoleted by **dhparam**|*dhparam(1)*.

dsa|*dsa(1)*
: DSA Data Management.

dsaparam|*dsaparam(1)*
: DSA Parameter Generation.

enc|*enc(1)*
: Encoding with Ciphers.

errstr|*errstr(1)*
: Error Number to Error String Conversion.

dhparam|*dhparam(1)*
: Generation and Management of Diffie-Hellman Parameters.

gendh
: Generation of Diffie-Hellman Parameters. Obsoleted by **dhparam**|*dhparam(1)*.

gendsa|*gendsa(1)*

Generation of DSA Parameters.

genrsa|*genrsa(1)*
Generation of RSA Parameters.

ocsp|*ocsp(1)*
Online Certificate Status Protocol utility.

passwd|*passwd(1)*
Generation of hashed passwords.

pkcs12|*pkcs12(1)*
PKCS#12 Data Management.

pkcs7|*pkcs7(1)*
PKCS#7 Data Management.

rand|*rand(1)*
Generate pseudo-random bytes.

req|*req(1)* X.509 Certificate Signing Request (CSR) Management.

rsa|*rsa(1)* RSA Data Management.

rsautl|*rsautl(1)*
RSA utility for signing, verification, encryption, and decryption.

s_client|*s_client(1)*
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.

s_server|*s_server(1)*
This implements a generic SSL/TLS server which accepts connections from remote clients 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. It provides both an own command line oriented protocol for testing SSL functions and a simple HTTP response facility to emulate an SSL/TLS-aware webserver.

s_time|*s_time(1)*
SSL Connection Timer.

sess_id|*sess_id(1)*
SSL Session Data Management.

smime|*smime(1)*
S/MIME mail processing.

speed|*speed(1)*
Algorithm Speed Measurement.

verify|*verify(1)*
X.509 Certificate Verification.

version|*version(1)*
OpenSSL Version Information.

x509|*x509(1)*
X.509 Certificate Data Management.

MESSAGE DIGEST COMMANDS

md2 MD2 Digest

md5 MD5 Digest

mdc2 MDC2 Digest

rmd160 RMD-160 Digest

sha
SHA Digest

sha1 SHA-1 Digest

ENCODING AND CIPHER COMMANDS

base64 Base64 Encoding

bf bf-cbc bf-cfb bf-ecb bf-ofb
Blowfish Cipher

cast cast-cbc
CAST Cipher

cast5-cbc cast5-cfb cast5-ecb cast5-ofb
CAST5 Cipher

des des-cbc des-cfb des-ecb des-ede des-ede-cbc des-ede-cfb des-ede-ofb des-ofb
DES Cipher

des3 desx des-ede3 des-ede3-cbc des-ede3-cfb des-ede3-ofb
Triple-DES Cipher

idea idea-cbc idea-cfb idea-ecb idea-ofb
IDEA Cipher

rc2 rc2-cbc rc2-cfb rc2-ecb rc2-ofb
RC2 Cipher

rc4 RC4 Cipher

rc5 rc5-cbc rc5-cfb rc5-ecb rc5-ofb
RC5 Cipher

PASS PHRASE ARGUMENTS

Several commands accept password arguments, typically using **-passin** and **-passout** for input and output passwords respectively. These allow the password to be obtained from a variety of sources. Both of these options take a single argument whose format is described below. If no password argument is given and a password is required then the user is prompted to enter one: this will typically be read from the current terminal with echoing turned off.

pass:password
the actual password is **password**. Since the password is visible to utilities (like 'ps' under Unix) this form should only be used where security is not important.

env:var
obtain the password from the environment variable **var**. Since the environment of other processes is visible on certain platforms (e.g. ps under certain Unix OSes) this option should be used with caution.

file:pathname
the first line of **pathname** is the password. If the same **pathname** argument is supplied to **-passin** and **-passout** arguments then the first line will be used for the input password and the next line for the output password. **pathname** need not refer to a regular file: it could for example refer to a device or named pipe.

fd:number
read the password from the file descriptor **number**. This can be used to send the data via a pipe for example.

stdin
read the password from standard input.

SEE ALSO

asn1parse(1)|asn1parse(1), *ca(1)|ca(1)*, *config(5)|config(5)*, *crl(1)|crl(1)*, *crl2pkcs7(1)|crl2pkcs7(1)*, *dgst(1)|dgst(1)*, *dhparam(1)|dhparam(1)*, *dsa(1)|dsa(1)*, *dsaparam(1)|dsaparam(1)*, *enc(1)|enc(1)*, *gendsa(1)|gendsa(1)*, *genrsa(1)|genrsa(1)*, *nseq(1)|nseq(1)*, *openssl(1)|openssl(1)*, *passwd(1)|passwd(1)*, *pkcs12(1)|pkcs12(1)*, *pkcs7(1)|pkcs7(1)*, *pkcs8(1)|pkcs8(1)*, *rand(1)|rand(1)*, *req(1)|req(1)*, *rsa(1)|rsa(1)*, *rsautl(1)|rsautl(1)*, *s_client(1)|s_client(1)*, *s_server(1)|s_server(1)*, *smime(1)|smime(1)*, *spkac(1)|spkac(1)*, *verify(1)|verify(1)*, *version(1)|version(1)*, *x509(1)|x509(1)*, *crypto(3)|crypto(3)*, *ssl(3)|ssl(3)*

HISTORY

The openssl(1) document appeared in OpenSSL 0.9.2. The **list-*XXX*-commands** pseudo-commands were added in OpenSSL 0.9.3; the **no-*XXX*** pseudo-commands were added in OpenSSL 0.9.5a. For notes on the availability of other commands, see their individual manual pages.

PASSWD

passwd - compute password hashes

SYNOPSIS

openssl passwd [**-crypt**] [**-1**] [**-apr1**] [**-salt** *string*] [**-in** *file*] [**-stdin**] [**-noverify**] [**-quiet**] [**-table**] {*password*}

DESCRIPTION

The **passwd** command computes the hash of a password typed at run-time or the hash of each password in a list. The password list is taken from the named file for option **-in file**, from stdin for option **-stdin**, or from the command line, or from the terminal otherwise. The Unix standard algorithm **crypt** and the MD5-based BSD password algorithm **1** and its Apache variant **apr1** are available.

OPTIONS

-crypt
Use the **crypt** algorithm (default).

-1 Use the MD5 based BSD password algorithm **1**.

-apr1
Use the **apr1** algorithm (Apache variant of the BSD algorithm).

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

-in *file*
Read passwords from *file*.

-stdin
Read passwords from **stdin**.

-noverify
Don‘t verify when reading a password from the terminal.

-quiet
Don‘t output warnings when passwords given at the command line are truncated.

-table
In the output list, prepend the cleartext password and a TAB character to each password hash.

EXAMPLES

openssl passwd -crypt -salt xx password prints **xxj31ZMTZzkVA**.
openssl passwd -1 -salt xxxxxxxx password prints **1xxxxxxxx$UYCIxa628.9qXjpQCjM4a.**.
openssl passwd -apr1 -salt xxxxxxxx password prints
$apr1$xxxxxxxx$dxHfLAsjHkDRmG83UXe8K0.

PKCS12

pkcs12 - PKCS#12 file utility

SYNOPSIS

openssl pkcs12 [**-export**] [**-chain**] [**-inkey filename**] [**-certfile filename**] [**-name name**] [**-caname name**] [**-in filename**] [**-out filename**] [**-noout**] [**-nomacver**] [**-nocerts**] [**-clcerts**] [**-cacerts**] [**-nokeys**] [**-info**] [**-des**] [**-des3**] [**-idea**] [**-nodes**] [**-noiter**] [**-maciter**] [**-twopass**] [**-descert**] [**-certpbe**] [**-keypbe**] [**-keyex**] [**-keysig**] [**-password arg**] [**-passin arg**] [**-passout arg**] [**-rand file(s)**]

DESCRIPTION

The **pkcs12** command allows PKCS#12 files (sometimes referred to as PFX files) to be created and parsed. PKCS#12 files are used by several programs including Netscape, MSIE and MS Outlook.

COMMAND OPTIONS

There are a lot of options the meaning of some depends of whether a PKCS#12 file is being created or parsed. By default a PKCS#12 file is parsed a PKCS#12 file can be created by using the **-export** option (see below).

PARSING OPTIONS

-in filename

This specifies filename of the PKCS#12 file to be parsed. Standard input is used by default.

-out filename

The filename to write certificates and private keys to, standard output by default. They are all written in PEM format.

-pass arg, **-passin arg**

the PKCS#12 file (i.e. input file) password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-passout arg

pass phrase source to encrypt any outputed private keys with. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-noout

this option inhibits output of the keys and certificates to the output file version of the PKCS#12 file.

-clcerts

only output client certificates (not CA certificates).

-cacerts

only output CA certificates (not client certificates).

-nocerts

no certificates at all will be output.

-nokeys

no private keys will be output.

-info

output additional information about the PKCS#12 file structure, algorithms used and iteration counts.

-des

use DES to encrypt private keys before outputting.

-des3

use triple DES to encrypt private keys before outputting, this is the default.

-idea

use IDEA to encrypt private keys before outputting.

-nodes

don‘t encrypt the private keys at all.

-nomacver

don‘t attempt to verify the integrity MAC before reading the file.

-twopass

prompt for separate integrity and encryption passwords: most software always assumes these are the same so this option will render such PKCS#12 files unreadable.

FILE CREATION OPTIONS

-export

This option specifies that a PKCS#12 file will be created rather than parsed.

-out filename

This specifies filename to write the PKCS#12 file to. Standard output is used by default.

-in filename

The filename to read certificates and private keys from, standard input by default. They must all be in PEM format. The order doesn‘t matter but one private key and its corresponding certificate should be present. If additional certificates are present they will also be included in the PKCS#12 file.

-inkey filename

file to read private key from. If not present then a private key must be present in the input file.

-name friendlyname

This specifies the "friendly name" for the certificate and private key. This name is typically displayed in list boxes by software importing the file.

-certfile filename

A filename to read additional certificates from.

-caname friendlyname

This specifies the "friendly name" for other certificates. This option may be used multiple times to specify names for all certificates in the order they appear. Netscape ignores friendly names on other certificates whereas MSIE displays them.

-pass arg, **-passout arg**

the PKCS#12 file (i.e. output file) password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-passin password

pass phrase source to decrypt any input private keys with. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-chain

if this option is present then an attempt is made to include the entire certificate chain of the user certificate. The standard CA store is used for this search. If the search fails it is considered a fatal error.

-descert

encrypt the certificate using triple DES, this may render the PKCS#12 file unreadable by some "export grade" software. By default the private key is encrypted using triple DES and the certificate using 40 bit RC2.

-keypbe alg, **-certpbe alg**

these options allow the algorithm used to encrypt the private key and certificates to be selected. Although any PKCS#5 v1.5 or PKCS#12 algorithms can be selected it is advisable only to use PKCS#12 algorithms. See the list in the **NOTES** section for more information.

-keyex|-keysig

specifies that the private key is to be used for key exchange or just signing. This option is only interpreted by MSIE and similar MS software. Normally "export grade" software will only allow 512 bit RSA keys to be used for encryption purposes but arbitrary length keys for signing. The **-keysig** option marks the key for signing only. Signing only keys can be used for S/MIME signing, authenticode (ActiveX control signing) and SSL client authentication, however due to a bug only MSIE 5.0 and later support the use of signing only keys for SSL client authentication.

-nomaciter, **-noiter**

these options affect the iteration counts on the MAC and key algorithms. Unless you wish to produce files compatible with MSIE 4.0 you should leave these options alone.

To discourage attacks by using large dictionaries of common passwords the algorithm that derives keys from passwords can have an iteration count applied to it: this causes a certain part of the algorithm to be repeated and slows it down. The MAC is used to check the file integrity but since it will normally have the same password as the keys and certificates it could also be attacked. By default both MAC and encryption iteration counts are set to 2048, using these options the MAC and encryption iteration counts can be set to 1, since this reduces the file security you should not use these options unless you really have to. Most software supports both MAC and key iteration

counts. MSIE 4.0 doesn't support MAC iteration counts so it needs the **-nomaciter** option.

-maciter

This option is included for compatibility with previous versions, it used to be needed to use MAC iterations counts but they are now used by default.

-rand file(s)

a file or files containing random data used to seed the random number generator, or an EGD socket (see *RAND_egd(3)|RAND_egd(3)*). Multiple files can be specified separated by a OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

NOTES

Although there are a large number of options most of them are very rarely used. For PKCS#12 file parsing only **-in** and **-out** need to be used for PKCS#12 file creation **-export** and **-name** are also used.

If none of the **-clcerts**, **-cacerts** or **-nocerts** options are present then all certificates will be output in the order they appear in the input PKCS#12 files. There is no guarantee that the first certificate present is the one corresponding to the private key. Certain software which requires a private key and certificate and assumes the first certificate in the file is the one corresponding to the private key: this may not always be the case. Using the **-clcerts** option will solve this problem by only outputting the certificate corresponding to the private key. If the CA certificates are required then they can be output to a separate file using the **-nokeys -cacerts** options to just output CA certificates.

The **-keypbe** and **-certpbe** algorithms allow the precise encryption algorithms for private keys and certificates to be specified. Normally the defaults are fine but occasionally software can't handle triple DES encrypted private keys, then the option **-keypbe PBE-SHA1-RC2-40** can be used to reduce the private key encryption to 40 bit RC2. A complete description of all algorithms is contained in the **pkcs8** manual page.

EXAMPLES

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
```

BUGS

Some would argue that the PKCS#12 standard is one big bug :-)

Versions of OpenSSL before 0.9.6a had a bug in the PKCS#12 key generation routines. Under rare circumstances this could produce a PKCS#12 file encrypted with an invalid key. As a result some PKCS#12 files which triggered this bug from other implementations (MSIE or Netscape) could not be decrypted by OpenSSL and similarly OpenSSL could produce PKCS#12 files which could not be decrypted by other implementations. The chances of producing such a file are relatively small: less than 1 in 256.

A side effect of fixing this bug is that any old invalidly encrypted PKCS#12 files cannot no longer be parsed by the fixed version. Under such circumstances the **pkcs12** utility will report that the MAC is OK but fail with a decryption error when extracting private keys.

This problem can be resolved by extracting the private keys and certificates from the PKCS#12 file using an older version of OpenSSL and recreating the PKCS#12 file from the keys and certificates using a newer version of OpenSSL. For example:

```
old-openssl -in bad.p12 -out keycerts.pem
openssl -in keycerts.pem -export -name "My PKCS#12 file" -out fixed.p12
```

SEE ALSO

pkcs8(1)|pkcs8(1)

PKCS7

pkcs7 - PKCS#7 utility

SYNOPSIS

openssl pkcs7 [**-inform PEM|DER**] [**-outform PEM|DER**] [**-in filename**] [**-out filename**] [**-print_certs**] [**-text**] [**-noout**] [**-engine id**]

DESCRIPTION

The **pkcs7** command processes PKCS#7 files in DER or PEM format.

COMMAND OPTIONS

-inform DER|PEM

This specifies the input format. **DER** format is DER encoded PKCS#7 v1.5 structure.**PEM** (the default) is a base64 encoded version of the DER form with header and footer lines.

-outform DER|PEM

This specifies the output format, the options have the same meaning as the **-inform** option.

-in filename

This specifies the input filename to read from or standard input if this option is not specified.

-out filename

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

-print_certs

prints out any certificates or CRLs contained in the file. They are preceded by their subject and issuer names in one line format.

-text

prints out certificates details in full rather than just subject and issuer names.

-noout

don‘t output the encoded version of the PKCS#7 structure (or certificates is **-print_certs** is set).

-engine id

specifying an engine (by it‘s unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

EXAMPLES

Convert a PKCS#7 file from PEM to DER:

```
openssl pkcs7 -in file.pem -outform DER -out file.der
```

Output all certificates in a file:

```
openssl pkcs7 -in file.pem -print_certs -out certs.pem
```

NOTES

The PEM PKCS#7 format uses the header and footer lines:

```
-----BEGIN PKCS7-----
-----END PKCS7-----
```

For compatibility with some CAs it will also accept:

```
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
```

RESTRICTIONS

There is no option to print out all the fields of a PKCS#7 file.

This PKCS#7 routines only understand PKCS#7 v 1.5 as specified in RFC2315 they cannot currently parse, for example, the new CMS as described in RFC2630.

SEE ALSO

crl2pkcs7(1)|crl2pkcs7(1)

PKCS8

pkcs8 - PKCS#8 format private key conversion tool

SYNOPSIS

openssl pkcs8 [**-topk8**] [**-inform PEM|DER**] [**-outform PEM|DER**] [**-in filename**] [**-passin arg**] [**-out filename**] [**-passout arg**] [**-noiter**] [**-nocrypt**] [**-nooct**] [**-embed**] [**-nsdb**] [**-v2 alg**] [**-v1 alg**] [**-engine id**]

DESCRIPTION

The **pkcs8** command processes private keys in PKCS#8 format. It can handle both unencrypted PKCS#8 PrivateKeyInfo format and EncryptedPrivateKeyInfo format with a variety of PKCS#5 (v1.5 and v2.0) and PKCS#12 algorithms.

COMMAND OPTIONS

-topk8

Normally a PKCS#8 private key is expected on input and a traditional format private key will be written. With the **-topk8** option the situation is reversed: it reads a traditional format private key and writes a PKCS#8 format key.

-inform DER|PEM

This specifies the input format. If a PKCS#8 format key is expected on input then either a **DER** or **PEM** encoded version of a PKCS#8 key will be expected. Otherwise the **DER** or **PEM** format of the traditional format private key is used.

-outform DER|PEM

This specifies the output format, the options have the same meaning as the **-inform** option.

-in filename

This specifies the input filename to read a key from or standard input if this option is not specified. If the key is encrypted a pass phrase will be prompted for.

-passin arg

the input file password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-out filename

This specifies the output filename to write a key to or standard output by default. If any encryption options are set then a pass phrase will be prompted for. The output filename should **not** be the same as the input filename.

-passout arg

the output file password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-nocrypt

PKCS#8 keys generated or input are normally PKCS#8 EncryptedPrivateKeyInfo structures using an appropriate password based encryption algorithm. With this option an unencrypted PrivateKeyInfo structure is expected or output. This option does not encrypt private keys at all and should only be used when absolutely necessary. Certain software such as some versions of Java code signing software used unencrypted private keys.

-nooct

This option generates RSA private keys in a broken format that some software uses. Specifically the private key should be enclosed in a OCTET STRING but some software just includes the structure itself without the surrounding OCTET STRING.

-embed

This option generates DSA keys in a broken format. The DSA parameters are embedded inside the PrivateKey structure. In this form the OCTET STRING contains an ASN1 SEQUENCE consisting of two structures: a SEQUENCE containing the parameters and an ASN1 INTEGER containing the private key.

-nsdb

This option generates DSA keys in a broken format compatible with Netscape private key databases. The PrivateKey contains a SEQUENCE consisting of the public and private keys respectively.

-v2 alg

This option enables the use of PKCS#5 v2.0 algorithms. Normally PKCS#8 private keys are encrypted with the password based encryption algorithm called **pbeWithMD5AndDES-CBC** this uses 56 bit DES encryption but it was the strongest encryption algorithm supported in PKCS#5 v1.5. Using the **-v2** option PKCS#5 v2.0 algorithms are used which can use any encryption algorithm such as 168 bit triple DES or 128 bit RC2 however not many implementations support PKCS#5 v2.0 yet. If you are just using private keys with OpenSSL then this doesn't matter.
The **alg** argument is the encryption algorithm to use, valid values include **des**, **des3** and **rc2**. It is recommended that **des3** is used.

-v1 alg

This option specifies a PKCS#5 v1.5 or PKCS#12 algorithm to use. A complete list of possible algorithms is included below.

-engine id

specifying an engine (by it's unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

NOTES

The encrypted form of a PEM encode PKCS#8 files uses the following headers and footers:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
-----END ENCRYPTED PRIVATE KEY-----
```

The unencrypted form uses:

```
-----BEGIN PRIVATE KEY-----
-----END PRIVATE KEY-----
```

Private keys encrypted using PKCS#5 v2.0 algorithms and high iteration counts are more secure that those encrypted using the traditional SSLeay compatible formats. So if additional security is considered important the keys should be converted.
The default encryption is only 56 bits because this is the encryption that most current implementations of PKCS#8 will support.
Some software may use PKCS#12 password based encryption algorithms with PKCS#8 format private keys: these are handled automatically but there is no option to produce them.
It is possible to write out DER encoded encrypted private keys in PKCS#8 format because the encryption details are included at an ASN1 level whereas the traditional format includes them at a PEM level.

PKCS#5 v1.5 and PKCS#12 algorithms.

Various algorithms can be used with the **-v1** command line option, including PKCS#5 v1.5 and PKCS#12. These are described in more detail below.

PBE-MD2-DES PBE-MD5-DES

These algorithms were included in the original PKCS#5 v1.5 specification. They only offer 56 bits of protection since they both use DES.

PBE-SHA1-RC2-64 PBE-MD2-RC2-64 PBE-MD5-RC2-64 PBE-SHA1-DES

These algorithms are not mentioned in the original PKCS#5 v1.5 specification but they use the same key derivation algorithm and are supported by some software. They are mentioned in PKCS#5 v2.0. They use either 64 bit RC2 or 56 bit DES.

PBE-SHA1-RC4-128 PBE-SHA1-RC4-40 PBE-SHA1-3DES PBE-SHA1-2DES PBE-SHA1-RC2-128 PBE-SHA1-RC2-40

These algorithms use the PKCS#12 password based encryption algorithm and allow strong encryption algorithms like triple DES or 128 bit RC2 to be used.

EXAMPLES

Convert a private from traditional to PKCS#5 v2.0 format using triple DES:

```
openssl pkcs8 -in key.pem -topk8 -v2 des3 -out enckey.pem
```

Convert a private key to PKCS#8 using a PKCS#5 1.5 compatible algorithm (DES):

```
openssl pkcs8 -in key.pem -topk8 -out enckey.pem
```

Convert a private key to PKCS#8 using a PKCS#12 compatible algorithm (3DES):

```
openssl pkcs8 -in key.pem -topk8 -out enckey.pem -v1 PBE-SHA1-3DES
```

Read a DER unencrypted PKCS#8 format private key:

```
openssl pkcs8 -inform DER -nocrypt -in key.der -out key.pem
```

Convert a private key from any PKCS#8 format to traditional format:

```
openssl pkcs8 -in pk8.pem -out key.pem
```

STANDARDS

Test vectors from this PKCS#5 v2.0 implementation were posted to the pkcs-tng mailing list using triple DES, DES and RC2 with high iteration counts, several people confirmed that they could decrypt the private keys produced and Therefore it can be assumed that the PKCS#5 v2.0 implementation is reasonably accurate at least as far as these algorithms are concerned.

The format of PKCS#8 DSA (and other) private keys is not well documented: it is hidden away in PKCS#11 v2.01, section 11.9. OpenSSL's default DSA PKCS#8 private key format complies with this standard.

BUGS

There should be an option that prints out the encryption algorithm in use and other details such as the iteration count.

PKCS#8 using triple DES and PKCS#5 v2.0 should be the default private key format for OpenSSL: for compatibility several of the utilities use the old format at present.

SEE ALSO

dsa(1)|dsa(1), *rsa(1)|rsa(1)*, *genrsa(1)|genrsa(1)*, *gendsa(1)|gendsa(1)*

RAND

rand - generate pseudo-random bytes

SYNOPSIS

openssl rand [**-out** *file*] [**-rand** *file(s)*] [**-base64**] *num*

DESCRIPTION

The **rand** command outputs *num* pseudo-random bytes after seeding the random number generator once. As in other **openssl** command line tools, PRNG seeding uses the file *$HOME*/**.rnd** or **.rnd** in addition to the files given in the **-rand** option. A new *$HOME*/**.rnd** or **.rnd** file will be written back if enough seeding was obtained from these sources.

OPTIONS

-out *file*

Write to *file* instead of standard output.

-rand *file(s)*

Use specified file or files or EGD socket (see *RAND_egd(3)|RAND_egd(3)*) for seeding the random number generator. Multiple files can be specified separated by a OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

-base64

Perform base64 encoding on the output.

SEE ALSO

RAND_bytes(3)|RAND_bytes(3)

REQ

req - PKCS#10 certificate request and certificate generating utility.

SYNOPSIS

openssl req [**-inform PEM|DER**] [**-outform PEM|DER**] [**-in filename**] [**-passin arg**] [**-out filename**] [**-passout arg**] [**-text**] [**-pubkey**] [**-noout**] [**-verify**] [**-modulus**] [**-new**] [**-rand file(s)**] [**-newkey rsa:bits**] [**-newkey dsa:file**] [**-nodes**] [**-key filename**] [**-keyform PEM|DER**] [**-keyout filename**] [**-[md5|sha1|md2|mdc2]**] [**-config filename**] [**-subj arg**] [**-x509**] [**-days n**] [**-set_serial n**] [**-asn1-kludge**] [**-newhdr**] [**-extensions section**] [**-reqexts section**] [**-utf8**] [**-nameopt**] [**-batch**] [**-verbose**] [**-engine id**]

DESCRIPTION

The **req** command primarily creates and processes certificate requests in PKCS#10 format. It can additionally create self signed certificates for use as root CAs for example.

COMMAND OPTIONS

-inform DER|PEM

This specifies the input format. The **DER** option uses an ASN1 DER encoded form compatible with the PKCS#10. The **PEM** form is the default format: it consists of the **DER** format base64 encoded with additional header and footer lines.

-outform DER|PEM

This specifies the output format, the options have the same meaning as the **-inform** option.

-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.

-passin arg

the input file password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-out filename

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

-passout arg

the output file password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-text

prints out the certificate request in text form.

-pubkey

outputs the public key.

-noout

this option prevents output of the encoded version of the request.

-modulus

this option prints out the value of the modulus of the public key contained in the request.

-verify

verifies the signature on the request.

-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.

-rand file(s)

a file or files containing random data used to seed the random number generator, or an EGD socket (see *RAND_egd(3)|RAND_egd(3)*). Multiple files can be specified separated by a OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

-newkey arg

this option creates a new certificate request and a new private key. The argument takes one of two forms. **rsa:nbits**, where **nbits** is the number of bits, generates an RSA key **nbits** in size. **dsa:filename** generates a DSA key using the parameters in the file **filename**.

-key filename

This specifies the file to read the private key from. It also accepts PKCS#8 format private keys for PEM format files.

-keyform PEM|DER

the format of the private key file specified in the **-key** argument. PEM is the default.

-keyout filename

this gives the filename to write the newly created private key to. If this option is not specified then the filename present in the configuration file is used.

-nodes

if this option is specified then if a private key is created it will not be encrypted.

-[md5|sha1|md2|mdc2]

this specifies the message digest to sign the request with. This overrides the digest algorithm specified in the configuration file. This option is ignored for DSA requests: they always use SHA1.

-config filename

this allows an alternative configuration file to be specified, this overrides the compile time filename or any specified in the **OPENSSL_CONF** environment variable.

-subj arg

sets subject name for new request or supersedes the subject name when processing a request. The arg must be formatted as */type0=value0/type1=value1/type2=...*, characters may be escaped by \ (backslash), no spaces are skipped.

-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.

-set_serial n

serial number to use when outputting a self signed certificate. This may be specified as a decimal value or a hex value if preceded by **0x**. It is possible to use negative serial numbers but this is not recommended.

-extensions section

-reqexts section

these options specify alternative sections to include certificate extensions (if the **-x509** option is present) or certificate request extensions. This allows several different sections to be used in the same configuration file to specify requests for a variety of purposes.

-utf8

this option causes field values to be interpreted as UTF8 strings, by default they are interpreted as ASCII. This means that the field values, whether prompted from a terminal or obtained from a configuration file, must be valid UTF8 strings.

-nameopt option

option which determines how the subject or issuer names are displayed. The **option** argument can be a single option or multiple options separated by commas. Alternatively the **-nameopt** switch may be used more than once to set multiple options. See the *x509(1)|x509(1)* manual page for details.

-asn1-kludge

by default the **req** command outputs certificate requests containing no attributes in the correct PKCS#10 format. However certain CAs will only accept requests containing no attributes in an invalid form: this option produces this invalid format.

More precisely the **Attributes** in a PKCS#10 certificate request are defined as a **SET OF Attribute**. They are **not OPTIONAL** so if no attributes are present then they should be encoded as an empty **SET OF**. The invalid form does not include the empty **SET OF** whereas the correct form does.

It should be noted that very few CAs still require the use of this option.

-newhdr

Adds the word **NEW** to the PEM file header and footer lines on the outputed request. Some software (Netscape certificate server) and some CAs need this.

-batch

non-interactive mode.

-verbose

print extra details about the operations being performed.

-engine id

specifying an engine (by it's unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

CONFIGURATION FILE FORMAT

The configuration options are specified in the **req** section of the configuration file. As with all configuration files if no value is specified in the specific section (i.e. **req**) then the initial unnamed or **default** section is searched too.

The options available are described in detail below.

input_password output_password

The passwords for the input private key file (if present) and the output private key file (if one will be created). The command line options **passin** and **passout** override the configuration file values.

default_bits

This specifies the default key size in bits. If not specified then 512 is used. It is used if the **-new** option is used. It can be overridden by using the **-newkey** option.

default_keyfile

This is the default filename to write a private key to. If not specified the key is written to standard output. This can be overridden by the **-keyout** option.

oid_file

This specifies a file containing additional **OBJECT IDENTIFIERS**. Each line of the file should consist of the numerical form of the object identifier followed by white space then the short name followed by white space and finally the long name.

oid_section

This specifies a section in the configuration file containing extra object identifiers. Each line should consist of the short name of the object identifier followed by = and the numerical form. The short and long names are the same when this option is used.

RANDFILE

This specifies a filename in which random number seed information is placed and read from, or an EGD socket (see *RAND_egd(3)|RAND_egd(3)*). It is used for private key generation.

encrypt_key

If this is set to **no** then if a private key is generated it is **not** encrypted. This is equivalent to the **-nodes** command line option. For compatibility **encrypt_rsa_key** is an equivalent option.

default_md

This option specifies the digest algorithm to use. Possible values include **md5 sha1 mdc2**. If not present then MD5 is used. This option can be overridden on the command line.

string_mask

This option masks out the use of certain string types in certain fields. Most users will not need to change this option.

It can be set to several values **default** which is also the default option uses PrintableStrings, T61Strings and BMPStrings if the **pkix** value is used then only PrintableStrings and BMPStrings will be used. This follows the PKIX recommendation in RFC2459. If the **utf8only** option is used then only UTF8Strings will be used: this is the PKIX recommendation in RFC2459 after 2003. Finally the **nombstr** option just uses PrintableStrings and T61Strings: certain software has problems with BMPStrings and UTF8Strings: in particular Netscape.

req_extensions

this specifies the configuration file section containing a list of extensions to add to the certificate request. It can be overridden by the **-reqexts** command line switch.

x509_extensions

this specifies the configuration file section containing a list of extensions to add to certificate generated when the **-x509** switch is used. It can be overridden by the **-extensions** command line switch.

prompt
if set to the value **no** this disables prompting of certificate fields and just takes values from the config file directly. It also changes the expected format of the **distinguished_name** and **attributes** sections.

utf8 if set to the value **yes** then field values to be interpreted as UTF8 strings, by default they are interpreted as ASCII. This means that the field values, whether prompted from a terminal or obtained from a configuration file, must be valid UTF8 strings.

attributes
this specifies the section containing any request attributes: its format is the same as **distinguished_name**. Typically these may contain the challengePassword or unstructuredName types. They are currently ignored by OpenSSL's request signing utilities but some CAs might want them.

distinguished_name
This specifies the section containing the distinguished name fields to prompt for when generating a certificate or certificate request. The format is described in the next section.

DISTINGUISHED NAME AND ATTRIBUTE SECTION FORMAT

There are two separate formats for the distinguished name and attribute sections. If the **prompt** option is set to **no** then these sections just consist of field names and values: for example,

```
CN=My Name
OU=My Organization
emailAddress=someone@somewhere.org
```

This allows external programs (e.g. GUI based) to generate a template file with all the field names and values and just pass it to **req**. An example of this kind of configuration file is contained in the **EXAMPLES** section.

Alternatively if the **prompt** option is absent or not set to **no** then the file contains field prompting information. It consists of lines of the form:

```
fieldName="prompt"
fieldName_default="default field value"
fieldName_min= 2
fieldName_max= 4
```

"fieldName" is the field name being used, for example commonName (or CN). The "prompt" string is used to ask the user to enter the relevant details. If the user enters nothing then the default value is used if no default value is present then the field is omitted. A field can still be omitted if a default value is present if the user just enters the '.' character.

The number of characters entered must be between the fieldName_min and fieldName_max limits: there may be additional restrictions based on the field being used (for example countryName can only ever be two characters long and must fit in a PrintableString).

Some fields (such as organizationName) can be used more than once in a DN. This presents a problem because configuration files will not recognize the same name occurring twice. To avoid this problem if the fieldName contains some characters followed by a full stop they will be ignored. So for example a second organizationName can be input by calling it "1.organizationName".

The actual permitted field names are any object identifier short or long names. These are compiled into OpenSSL and include the usual values such as commonName, countryName, localityName, organizationName, organizationUnitName, stateOrProvinceName. Additionally emailAddress is include as well as name, surname, givenName initials and dnQualifier.

Additional object identifiers can be defined with the **oid_file** or **oid_section** options in the configuration file. Any additional fields will be treated as though they were a DirectoryString.

EXAMPLES

Examine and verify certificate request:

openssl req -in req.pem -text -verify -noout

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
```

Generate a self signed root certificate:

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

Example of a file pointed to by the **oid_file** option:

```
1.2.3.4         shortName       A longer Name
1.2.3.6         otherName       Other longer Name
```

Example of a section pointed to by **oid_section** making use of variable expansion:

```
testoid1=1.2.3.5
testoid2=${testoid1}.6
```

Sample configuration file prompting for field values:

```
[ req ]
default_bits            = 1024
default_keyfile         = privkey.pem
distinguished_name      = req_distinguished_name
attributes              = req_attributes
x509_extensions         = v3_ca

dirstring_type = nobmp

[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = AU
countryName_min                 = 2
countryName_max                 = 2

localityName                    = Locality Name (eg, city)

organizationalUnitName          = Organizational Unit Name (eg, section)

commonName                      = Common Name (eg, YOUR name)
commonName_max                  = 64

emailAddress                    = Email Address
emailAddress_max                = 40

[ req_attributes ]
challengePassword               = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20

[ v3_ca ]

subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer:always
basicConstraints = CA:true
```

Sample configuration containing all field values:

```
RANDFILE                = $ENV::HOME/.rnd

[ req ]
default_bits            = 1024
default_keyfile         = keyfile.pem
distinguished_name      = req_distinguished_name
attributes              = req_attributes
prompt                  = no
output_password         = mypass

[ req_distinguished_name ]
C                       = GB
ST                      = Test State or Province
L                       = Test Locality
O                       = Organization Name
OU                      = Organizational Unit Name
CN                      = Common Name
emailAddress            = test@email.address

[ req_attributes ]
challengePassword       = A challenge password
```

NOTES

The header and footer lines in the **PEM** format are normally:

```
-----BEGIN CERTIFICATE REQUEST-----
-----END CERTIFICATE REQUEST-----
```

some software (some versions of Netscape certificate server) instead needs:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
-----END NEW CERTIFICATE REQUEST-----
```

which is produced with the **-newhdr** option but is otherwise compatible. Either form is accepted transparently on input.

The certificate requests generated by **Xenroll** with MSIE have extensions added. It includes the **keyUsage** extension which determines the type of key (signature only or general purpose) and any additional OIDs entered by the script in an extendedKeyUsage extension.

DIAGNOSTICS

The following messages are frequently asked about:

```
Using configuration from /some/path/openssl.cnf
Unable to load config info
```

This is followed some time later by...

```
unable to find 'distinguished_name' in config
problems making Certificate Request
```

The first error message is the clue: it can't find the configuration file! Certain operations (like examining a certificate request) don't need a configuration file so its use isn't enforced. Generation of certificates or requests however does need a configuration file. This could be regarded as a bug.

Another puzzling message is this:

```
Attributes:
    a0:00
```

this is displayed when no attributes are present and the request includes the correct empty **SET OF** structure (the DER encoding of which is 0xa0 0x00). If you just see:

```
Attributes:
```

then the **SET OF** is missing and the encoding is technically invalid (but it is tolerated). See the description of the command line option **-asn1-kludge** for more information.

ENVIRONMENT VARIABLES

The variable **OPENSSL_CONF** if defined allows an alternative configuration file location to be specified, it will be overridden by the **-config** command line switch if it is present. For compatibility reasons the **SSLEAY_CONF** environment variable serves the same purpose but its use is discouraged.

BUGS

OpenSSL‘s handling of T61Strings (aka TeletexStrings) is broken: it effectively treats them as ISO-8859-1 (Latin 1), Netscape and MSIE have similar behaviour. This can cause problems if you need characters that aren‘t available in PrintableStrings and you don‘t want to or can‘t use BMPStrings.

As a consequence of the T61String handling the only correct way to represent accented characters in OpenSSL is to use a BMPString: unfortunately Netscape currently chokes on these. If you have to use accented characters with Netscape and MSIE then you currently need to use the invalid T61String form.

The current prompting is not very friendly. It doesn‘t allow you to confirm what you‘ve just entered. Other things like extensions in certificate requests are statically defined in the configuration file. Some of these: like an email address in subjectAltName should be input by the user.

SEE ALSO

x509(1)|x509(1), *ca(1)|ca(1)*, *genrsa(1)|genrsa(1)*, *gendsa(1)|gendsa(1)*, *config(5)|config(5)*

RSA

rsa - RSA key processing tool

SYNOPSIS

openssl rsa [**-inform PEM|NET|DER**] [**-outform PEM|NET|DER**] [**-in filename**] [**-passin arg**] [**-out filename**] [**-passout arg**] [**-sgckey**] [**-des**] [**-des3**] [**-idea**] [**-text**] [**-noout**] [**-modulus**] [**-check**] [**-pubin**] [**-pubout**] [**-engine id**]

DESCRIPTION

The **rsa** command processes RSA keys. They can be converted between various forms and their components printed out. **Note** this command uses the traditional SSLeay compatible format for private key encryption: newer applications should use the more secure PKCS#8 format using the **pkcs8** utility.

COMMAND OPTIONS

-inform DER|NET|PEM

This specifies the input format. The **DER** option uses an ASN1 DER encoded form compatible with the PKCS#1 RSAPrivateKey or SubjectPublicKeyInfo format. The **PEM** form is the default format: it consists of the **DER** format base64 encoded with additional header and footer lines. On input PKCS#8 format private keys are also accepted. The **NET** form is a format is described in the **NOTES** section.

-outform DER|NET|PEM

This specifies the output format, the options have the same meaning as the **-inform** option.

-in filename

This specifies the input filename to read a key from or standard input if this option is not specified. If the key is encrypted a pass phrase will be prompted for.

-passin arg

the input file password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-out filename

This specifies the output filename to write a key to or standard output if this option is not specified. If any encryption options are set then a pass phrase will be prompted for. The output filename should **not** be the same as the input filename.

-passout password

the output file password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-sgckey

use the modified NET algorithm used with some versions of Microsoft IIS and SGC keys.

-des|-des3|-idea

These options encrypt the private key with the DES, triple DES, or the IDEA ciphers respectively before outputting it. A pass phrase is prompted for. If none of these options is specified the key is written in plain text. This means that using the **rsa** utility to read in an encrypted key with no encryption option can be used to remove the pass phrase from a key, or by setting the encryption options it can be use to add or change the pass phrase. These options can only be used with PEM format output files.

-text

prints out the various public or private key components in plain text in addition to the encoded version.

-noout

this option prevents output of the encoded version of the key.

-modulus

this option prints out the value of the modulus of the key.

-check

this option checks the consistency of an RSA private key.

-pubin

by default a private key is read from the input file: with this option a public key is read instead.

-pubout

by default a private key is output: with this option a public key will be output instead. This option is automatically set if the input is a public key.

-engine id

specifying an engine (by it‘s unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

NOTES

The PEM private key format uses the header and footer lines:

```
-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----
```

The PEM public key format uses the header and footer lines:

```
-----BEGIN PUBLIC KEY-----
-----END PUBLIC KEY-----
```

The **NET** form is a format compatible with older Netscape servers and Microsoft IIS .key files, this uses unsalted RC4 for its encryption. It is not very secure and so should only be used when necessary.
Some newer version of IIS have additional data in the exported .key files. To use these with the utility, view the file with a binary editor and look for the string "private-key", then trace back to the byte sequence 0x30, 0x82 (this is an ASN1 SEQUENCE). Copy all the data from this point onwards to another file and use that as the input to the **rsa** utility with the **-inform NET** option. If you get an error after entering the password try the **-sgckey** option.

EXAMPLES

To remove the pass phrase on an RSA private key:

```
openssl rsa -in key.pem -out keyout.pem
```

To encrypt a private key using triple DES:

```
openssl rsa -in key.pem -des3 -out keyout.pem
```

To convert a private key from PEM to DER format:

```
openssl rsa -in key.pem -outform DER -out keyout.der
```

To print out the components of a private key to standard output:

```
openssl rsa -in key.pem -text -noout
```

To just output the public part of a private key:

```
openssl rsa -in key.pem -pubout -out pubkey.pem
```

BUGS

The command line password arguments don‘t currently work with **NET** format.
There should be an option that automatically handles .key files, without having to manually edit them.

SEE ALSO

pkcs8(1)|pkcs8(1), *dsa(1)|dsa(1)*, *genrsa(1)|genrsa(1)*, *gendsa(1)|gendsa(1)*

RSAUTL

rsautl - RSA utility

SYNOPSIS

openssl rsautl [**-in file**] [**-out file**] [**-inkey file**] [**-pubin**] [**-certin**] [**-sign**] [**-verify**] [**-encrypt**] [**-decrypt**] [**-pkcs**] [**-ssl**] [**-raw**] [**-hexdump**] [**-asn1parse**]

DESCRIPTION

The **rsautl** command can be used to sign, verify, encrypt and decrypt data using the RSA algorithm.

COMMAND OPTIONS

-in filename
This specifies the input filename to read data from or standard input if this option is not specified.

-out filename
specifies the output filename to write to or standard output by default.

-inkey file
the input key file, by default it should be an RSA private key.

-pubin
the input file is an RSA public key.

-certin
the input is a certificate containing an RSA public key.

-sign
sign the input data and output the signed result. This requires and RSA private key.

-verify
verify the input data and output the recovered data.

-encrypt
encrypt the input data using an RSA public key.

-decrypt
decrypt the input data using an RSA private key.

-pkcs, -oaep, -ssl, -raw
the padding to use: PKCS#1 v1.5 (the default), PKCS#1 OAEP, special padding used in SSL v2 backwards compatible handshakes, or no padding, respectively. For signatures, only **-pkcs** and **-raw** can be used.

-hexdump
hex dump the output data.

-asn1parse
asn1parse the output data, this is useful when combined with the **-verify** option.

NOTES

rsautl because it uses the RSA algorithm directly can only be used to sign or verify small pieces of data.

EXAMPLES

Sign some data using a private key:

```
openssl rsautl -sign -in file -inkey key.pem -out sig
```

Recover the signed data

```
openssl rsautl -verify -in sig -inkey key.pem
```

Examine the raw signed data:

```
openssl rsautl -verify -in file -inkey key.pem -raw -hexdump

0000 - 00 01 ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0010 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0020 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0030 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0040 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0050 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
```

```
0060 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0070 - ff ff ff ff 00 68 65 6c-6c 6f 20 77 6f 72 6c 64   .....hello world
```

The PKCS#1 block formatting is evident from this. If this was done using encrypt and decrypt the block would have been of type 2 (the second byte) and random padding data visible instead of the 0xff bytes.
It is possible to analyse the signature of certificates using this utility in conjunction with **asn1parse**. Consider the self signed example in certs/pca-cert.pem . Running **asn1parse** as follows yields:

```
openssl asn1parse -in pca-cert.pem
```

```
  0:d=0  hl=4 l= 742 cons: SEQUENCE
  4:d=1  hl=4 l= 591 cons:  SEQUENCE
  8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
 10:d=3  hl=2 l=   1 prim:    INTEGER           :02
 13:d=2  hl=2 l=   1 prim:   INTEGER           :00
 16:d=2  hl=2 l=  13 cons:   SEQUENCE
 18:d=3  hl=2 l=   9 prim:    OBJECT            :md5WithRSAEncryption
 29:d=3  hl=2 l=   0 prim:    NULL
 31:d=2  hl=2 l=  92 cons:   SEQUENCE
 33:d=3  hl=2 l=  11 cons:    SET
 35:d=4  hl=2 l=   9 cons:     SEQUENCE
 37:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
 42:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :AU
....
599:d=1  hl=2 l=  13 cons:  SEQUENCE
601:d=2  hl=2 l=   9 prim:   OBJECT            :md5WithRSAEncryption
612:d=2  hl=2 l=   0 prim:   NULL
614:d=1  hl=3 l= 129 prim:  BIT STRING
```

The final BIT STRING contains the actual signature. It can be extracted with:

```
openssl asn1parse -in pca-cert.pem -out sig -noout -strparse 614
```

The certificate public key can be extracted with:

```
openssl x509 -in test/testx509.pem -pubout -noout >pubkey.pem
```

The signature can be analysed with:

```
openssl rsautl -in sig -verify -asn1parse -inkey pubkey.pem -pubin
```

```
  0:d=0  hl=2 l=  32 cons: SEQUENCE
  2:d=1  hl=2 l=  12 cons:  SEQUENCE
  4:d=2  hl=2 l=   8 prim:   OBJECT            :md5
 14:d=2  hl=2 l=   0 prim:   NULL
 16:d=1  hl=2 l=  16 prim:  OCTET STRING
    0000 - f3 46 9e aa 1a 4a 73 c9-37 ea 93 00 48 25 08 b5   .F...Js.7...H%..
```

This is the parsed version of an ASN1 DigestInfo structure. It can be seen that the digest used was md5.
The actual part of the certificate that was signed can be extracted with:

```
openssl asn1parse -in pca-cert.pem -out tbs -noout -strparse 4
```

and its digest computed with:

```
openssl md5 -c tbs
MD5(tbs)= f3:46:9e:aa:1a:4a:73:c9:37:ea:93:00:48:25:08:b5
```

which it can be seen agrees with the recovered value above.

SEE ALSO

dgst(1)|dgst(1), *rsa(1)|rsa(1)*, *genrsa(1)|genrsa(1)*

SESS_ID

sess_id - SSL/TLS session handling utility

SYNOPSIS

openssl sess_id [**-inform PEM|DER**] [**-outform PEM|DER**] [**-in filename**] [**-out filename**] [**-text**] [**-noout**] [**-context ID**]

DESCRIPTION

The **sess_id** process the encoded version of the SSL session structure and optionally prints out SSL session details (for example the SSL session master key) in human readable format. Since this is a diagnostic tool that needs some knowledge of the SSL protocol to use properly, most users will not need to use it.

-inform DER|PEM
: This specifies the input format. The **DER** option uses an ASN1 DER encoded format containing session details. The precise format can vary from one version to the next. The **PEM** form is the default format: it consists of the **DER** format base64 encoded with additional header and footer lines.

-outform DER|PEM
: This specifies the output format, the options have the same meaning as the **-inform** option.

-in filename
: This specifies the input filename to read session information from or standard input by default.

-out filename
: This specifies the output filename to write session information to or standard output if this option is not specified.

-text
: prints out the various public or private key components in plain text in addition to the encoded version.

-cert
: if a certificate is present in the session it will be output using this option, if the **-text** option is also present then it will be printed out in text form.

-noout
: this option prevents output of the encoded version of the session.

-context ID
: this option can set the session id so the output session information uses the supplied ID. The ID can be any string of characters. This option wont normally be used.

OUTPUT

Typical output:

```
SSL-Session:
    Protocol  : TLSv1
    Cipher    : 0016
    Session-ID: 871E62626C554CE95488823752CBD5F3673A3EF3DCE9C67BD916C809914B40ED
    Session-ID-ctx: 01000000
    Master-Key: A7CEFC571974BE02CAC305269DC59F76EA9F0B180CB6642697A68251F2D2BB57E
    Key-Arg   : None
    Start Time: 948459261
    Timeout   : 300 (sec)
    Verify return code 0 (ok)
```

Theses are described below in more detail.

Protocol
: this is the protocol in use TLSv1, SSLv3 or SSLv2.

Cipher
: the cipher used this is the actual raw SSL or TLS cipher code, see the SSL or TLS specifications for more information.

Session-ID
the SSL session ID in hex format.

Session-ID-ctx
the session ID context in hex format.

Master-Key
this is the SSL session master key.

Key-Arg
the key argument, this is only used in SSL v2.

Start Time
this is the session start time represented as an integer in standard Unix format.

Timeout
the timeout in seconds.

Verify return code
this is the return code when an SSL client certificate is verified.

NOTES

The PEM encoded session format uses the header and footer lines:

```
-----BEGIN SSL SESSION PARAMETERS-----
-----END SSL SESSION PARAMETERS-----
```

Since the SSL session output contains the master key it is possible to read the contents of an encrypted session using this information. Therefore appropriate security precautions should be taken if the information is being output by a "real" application. This is however strongly discouraged and should only be used for debugging purposes.

BUGS

The cipher and start time should be printed out in human readable form.

SEE ALSO

ciphers(1)|ciphers(1), *s_server(1)|s_server(1)*

SMIME

smime - S/MIME utility

SYNOPSIS

openssl smime [**-encrypt**] [**-decrypt**] [**-sign**] [**-verify**] [**-pk7out**] [**-des**] [**-des3**] [**-rc2-40**] [**-rc2-64**] [**-rc2-128**] [**-in file**] [**-certfile file**] [**-signer file**] [**-recip file**] [**-inform SMIME|PEM|DER**] [**-passin arg**] [**-inkey file**] [**-out file**] [**-outform SMIME|PEM|DER**] [**-content file**] [**-to addr**] [**-from ad**] [**-subject s**] [**-text**] [**-rand file(s)**] [cert.pem]...

DESCRIPTION

The **smime** command handles S/MIME mail. It can encrypt, decrypt, sign and verify S/MIME messages.

COMMAND OPTIONS

There are five operation options that set the type of operation to be performed. The meaning of the other options varies according to the operation type.

-encrypt

encrypt mail for the given recipient certificates. Input file is the message to be encrypted. The output file is the encrypted mail in MIME format.

-decrypt

decrypt mail using the supplied certificate and private key. Expects an encrypted mail message in MIME format for the input file. The decrypted mail is written to the output file.

-sign

sign mail using the supplied certificate and private key. Input file is the message to be signed. The signed message in MIME format is written to the output file.

-verify

verify signed mail. Expects a signed mail message on input and outputs the signed data. Both clear text and opaque signing is supported.

-pk7out

takes an input message and writes out a PEM encoded PKCS#7 structure.

-in filename

the input message to be encrypted or signed or the MIME message to be decrypted or verified.

-inform SMIME|PEM|DER

this specifies the input format for the PKCS#7 structure. The default is **SMIME** which reads an S/MIME format message. **PEM** and **DER** format change this to expect PEM and DER format PKCS#7 structures instead. This currently only affects the input format of the PKCS#7 structure, if no PKCS#7 structure is being input (for example with **-encrypt** or **-sign**) this option has no effect.

-out filename

the message text that has been decrypted or verified or the output MIME format message that has been signed or verified.

-outform SMIME|PEM|DER

this specifies the output format for the PKCS#7 structure. The default is **SMIME** which write an S/MIME format message. **PEM** and **DER** format change this to write PEM and DER format PKCS#7 structures instead. This currently only affects the output format of the PKCS#7 structure, if no PKCS#7 structure is being output (for example with **-verify** or **-decrypt**) this option has no effect.

-content filename

This specifies a file containing the detached content, this is only useful with the **-verify** command. This is only usable if the PKCS#7 structure is using the detached signature form where the content is not included. This option will override any content if the input format is S/MIME and it uses the multipart/signed MIME content type.

-text

this option adds plain text (text/plain) MIME headers to the supplied message if encrypting or signing. If decrypting or verifying it strips off text headers: if the decrypted or verified message is not of MIME type text/plain then an error occurs.

-CAfile file

a file containing trusted CA certificates, only used with **-verify**.

-CApath dir

a directory containing trusted CA certificates, only used with **-verify**. This directory must be a standard certificate directory: that is a hash of each subject name (using **x509 -hash**) should be linked to each certificate.

-des -des3 -rc2-40 -rc2-64 -rc2-128

the encryption algorithm to use. DES (56 bits), triple DES (168 bits) or 40, 64 or 128 bit RC2 respectively if not specified 40 bit RC2 is used. Only used with **-encrypt**.

-nointern

when verifying a message normally certificates (if any) included in the message are searched for the signing certificate. With this option only the certificates specified in the **-certfile** option are used. The supplied certificates can still be used as untrusted CAs however.

-noverify

do not verify the signers certificate of a signed message.

-nochain

do not do chain verification of signers certificates: that is don‘t use the certificates in the signed message as untrusted CAs.

-nosigs

don‘t try to verify the signatures on the message.

-nocerts

when signing a message the signer‘s certificate is normally included with this option it is excluded. This will reduce the size of the signed message but the verifier must have a copy of the signers certificate available locally (passed using the **-certfile** option for example).

-noattr

normally when a message is signed a set of attributes are included which include the signing time and supported symmetric algorithms. With this option they are not included.

-binary

normally the input message is converted to "canonical" format which is effectively using CR and LF as end of line: as required by the S/MIME specification. When this option is present no translation occurs. This is useful when handling binary data which may not be in MIME format.

-nodetach

when signing a message use opaque signing: this form is more resistant to translation by mail relays but it cannot be read by mail agents that do not support S/MIME. Without this option cleartext signing with the MIME type multipart/signed is used.

-certfile file

allows additional certificates to be specified. When signing these will be included with the message. When verifying these will be searched for the signers certificates. The certificates should be in PEM format.

-signer file

the signers certificate when signing a message. If a message is being verified then the signers certificates will be written to this file if the verification was successful.

-recip file

the recipients certificate when decrypting a message. This certificate must match one of the recipients of the message or an error occurs.

-inkey file

the private key to use when signing or decrypting. This must match the corresponding certificate. If this option is not specified then the private key must be included in the certificate file specified with the **-recip** or **-signer** file.

-passin arg

the private key password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-rand file(s)

a file or files containing random data used to seed the random number generator, or an EGD socket (see *RAND_egd(3)|RAND_egd(3)*). Multiple files can be specified separated by a OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

cert.pem...

one or more certificates of message recipients: used when encrypting a message.

-to, -from, -subject

the relevant mail headers. These are included outside the signed portion of a message so they may be included manually. If signing then many S/MIME mail clients check the signers certificate's email address matches that specified in the From: address.

NOTES

The MIME message must be sent without any blank lines between the headers and the output. Some mail programs will automatically add a blank line. Piping the mail directly to sendmail is one way to achieve the correct format.

The supplied message to be signed or encrypted must include the necessary MIME headers or many S/MIME clients wont display it properly (if at all). You can use the **-text** option to automatically add plain text headers.

A "signed and encrypted" message is one where a signed message is then encrypted. This can be produced by encrypting an already signed message: see the examples section.

This version of the program only allows one signer per message but it will verify multiple signers on received messages. Some S/MIME clients choke if a message contains multiple signers. It is possible to sign messages "in parallel" by signing an already signed message.

The options **-encrypt** and **-decrypt** reflect common usage in S/MIME clients. Strictly speaking these process PKCS#7 enveloped data: PKCS#7 encrypted data is used for other purposes.

EXIT CODES

0 the operation was completely successfully.

1 an error occurred parsing the command options.

2 one of the input files could not be read.

3 an error occurred creating the PKCS#7 file or when reading the MIME message.

4 an error occurred decrypting or verifying the message.

5 the message was verified correctly but an error occurred writing out the signers certificates.

EXAMPLES

Create a cleartext signed message:

```
openssl smime -sign -in message.txt -text -out mail.msg \
       -signer mycert.pem
```

Create and opaque signed message

```
openssl smime -sign -in message.txt -text -out mail.msg -nodetach \
       -signer mycert.pem
```

Create a signed message, include some additional certificates and read the private key from another file:

```
openssl smime -sign -in in.txt -text -out mail.msg \
       -signer mycert.pem -inkey mykey.pem -certfile mycerts.pem
```

Send a signed message under Unix directly to sendmail, including headers:

```
openssl smime -sign -in in.txt -text -signer mycert.pem \
       -from steve@openssl.org -to someone@somewhere \
       -subject "Signed message" | sendmail someone@somewhere
```

Verify a message and extract the signer's certificate if successful:

```
openssl smime -verify -in mail.msg -signer user.pem -out signedtext.txt
```

Send encrypted mail using triple DES:

```
openssl smime -encrypt -in in.txt -from steve@openssl.org \
       -to someone@somewhere -subject "Encrypted message" \
       -des3 user.pem -out mail.msg
```

Sign and encrypt mail:

```
openssl smime -sign -in ml.txt -signer my.pem -text \
        | openssl smime -encrypt -out mail.msg \
        -from steve@openssl.org -to someone@somewhere \
        -subject "Signed and Encrypted message" -des3 user.pem
```

Note: the encryption command does not include the **-text** option because the message being encrypted already has MIME headers.
Decrypt mail:

```
openssl smime -decrypt -in mail.msg -recip mycert.pem -inkey key.pem
```

The output from Netscape form signing is a PKCS#7 structure with the detached signature format. You can use this program to verify the signature by line wrapping the base64 encoded structure and surrounding it with:

```
-----BEGIN PKCS7-----
-----END PKCS7-----
```

and using the command,

```
openssl smime -verify -inform PEM -in signature.pem -content content.txt
```

alternatively you can base64 decode the signature and use

```
openssl smime -verify -inform DER -in signature.der -content content.txt
```

BUGS

The MIME parser isn‘t very clever: it seems to handle most messages that I‘ve thrown at it but it may choke on others.
The code currently will only write out the signer‘s certificate to a file: if the signer has a separate encryption certificate this must be manually extracted. There should be some heuristic that determines the correct encryption certificate.
Ideally a database should be maintained of a certificates for each email address.
The code doesn‘t currently take note of the permitted symmetric encryption algorithms as supplied in the SMIMECapabilities signed attribute. this means the user has to manually include the correct encryption algorithm. It should store the list of permitted ciphers in a database and only use those.
No revocation checking is done on the signer‘s certificate.
The current code can only handle S/MIME v2 messages, the more complex S/MIME v3 structures may cause parsing errors.

SPEED

speed - test library performance

SYNOPSIS

openssl speed [**-engine id**] [**md2**] [**mdc2**] [**md5**] [**hmac**] [**sha1**] [**rmd160**] [**idea-cbc**] [**rc2-cbc**] [**rc5-cbc**] [**bf-cbc**] [**des-cbc**] [**des-ede3**] [**rc4**] [**rsa512**] [**rsa1024**] [**rsa2048**] [**rsa4096**] [**dsa512**] [**dsa1024**] [**dsa2048**] [**idea**] [**rc2**] [**des**] [**rsa**] [**blowfish**]

DESCRIPTION

This command is used to test the performance of cryptographic algorithms.

OPTIONS

-engine id

specifying an engine (by it‘s unique **id** string) will cause **speed** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

[zero or more test algorithms]

If any options are given, **speed** tests those algorithms, otherwise all of the above are tested.

SPKAC

spkac - SPKAC printing and generating utility

SYNOPSIS

openssl spkac [**-in filename**] [**-out filename**] [**-key keyfile**] [**-passin arg**] [**-challenge string**] [**-pubkey**] [**-spkac spkacname**] [**-spksect section**] [**-noout**] [**-verify**] [**-engine id**]

DESCRIPTION

The **spkac** command processes Netscape signed public key and challenge (SPKAC) files. It can print out their contents, verify the signature and produce its own SPKACs from a supplied private key.

COMMAND OPTIONS

-in filename
: This specifies the input filename to read from or standard input if this option is not specified. Ignored if the **-key** option is used.

-out filename
: specifies the output filename to write to or standard output by default.

-key keyfile
: create an SPKAC file using the private key in **keyfile**. The **-in**, **-noout**, **-spksect** and **-verify** options are ignored if present.

-passin password
: the input file password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in *openssl(1)|openssl(1)*.

-challenge string
: specifies the challenge string if an SPKAC is being created.

-spkac spkacname
: allows an alternative name form the variable containing the SPKAC. The default is "SPKAC". This option affects both generated and input SPKAC files.

-spksect section
: allows an alternative name form the section containing the SPKAC. The default is the default section.

-noout
: don‘t output the text version of the SPKAC (not used if an SPKAC is being created).

-pubkey
: output the public key of an SPKAC (not used if an SPKAC is being created).

-verify
: verifies the digital signature on the supplied SPKAC.

-engine id
: specifying an engine (by it‘s unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

EXAMPLES

Print out the contents of an SPKAC:

```
openssl spkac -in spkac.cnf
```

Verify the signature of an SPKAC:

```
openssl spkac -in spkac.cnf -noout -verify
```

Create an SPKAC using the challenge string "hello":

```
openssl spkac -key key.pem -challenge hello -out spkac.cnf
```

Example of an SPKAC, (long lines split up for clarity):

```
SPKAC=MIG5MGUwXDANBgkqhkiG9w0BAQEFAANLADBIAkEA1cCoq2Wa3Ixs47uI7F\
PVwHVIPDx5yso105Y6zpozam135a8R0CpoRvkkigIyXfcCjiVi5oWk+6FfPaD03u\
PFoQIDAQABFgVoZWxsbzANBgkqhkiG9w0BAQQFAANBAFpQtY/FojdwkJh1bEIYuc\
2EeM2KHTWPEepWYeawvHD0gQ3DngSC75YCWnnDdq+NQ3F+X4deMx9AaEglZtULwV\
```

4=

NOTES

A created SPKAC with suitable DN components appended can be fed into the **ca** utility.

SPKACs are typically generated by Netscape when a form is submitted containing the **KEYGEN** tag as part of the certificate enrollment process.

The challenge string permits a primitive form of proof of possession of private key. By checking the SPKAC signature and a random challenge string some guarantee is given that the user knows the private key corresponding to the public key being certified. This is important in some applications. Without this it is possible for a previous SPKAC to be used in a "replay attack".

SEE ALSO

ca(1)|ca(1)

S_CLIENT

s_client - SSL/TLS client program

SYNOPSIS

openssl s_client [**-connect** host:port] [**-verify depth**] [**-cert filename**] [**-key filename**] [**-CApath directory**] [**-CAfile filename**] [**-reconnect**] [**-pause**] [**-showcerts**] [**-debug**] [**-msg**] [**-nbio_test**] [**-state**] [**-nbio**] [**-crlf**] [**-ign_eof**] [**-quiet**] [**-ssl2**] [**-ssl3**] [**-tls1**] [**-no_ssl2**] [**-no_ssl3**] [**-no_tls1**] [**-bugs**] [**-cipher cipherlist**] [**-engine id**] [**-rand file(s)**]

DESCRIPTION

The **s_client** command implements a generic SSL/TLS client which connects to a remote host using SSL/TLS. It is a *very* useful diagnostic tool for SSL servers.

OPTIONS

-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.

-cert certname

The certificate to use, if one is requested by the server. The default is not to use a certificate.

-key keyfile

The private key to use. If not specified then the certificate file will be used.

-verify depth

The verify depth to use. This specifies the maximum length of the server certificate chain and turns on server certificate verification. Currently the verify operation continues after errors so all the problems with a certificate chain can be seen. As a side effect the connection will never fail due to a server certificate verify failure.

-CApath directory

The directory to use for server certificate verification. This directory must be in "hash format", see **verify** for more information. These are also used when building the client certificate chain.

-CAfile file

A file containing trusted certificates to use during server authentication and to use when attempting to build the client certificate chain.

-reconnect

reconnects to the same server 5 times using the same session ID, this can be used as a test that session caching is working.

-pause

pauses 1 second between each read and write call.

-showcerts

display the whole server certificate chain: normally only the server certificate itself is displayed.

-prexit

print session information when the program exits. This will always attempt to print out information even if the connection fails. Normally information will only be printed out once if the connection succeeds. This option is useful because the cipher in use may be renegotiated or the connection may fail because a client certificate is required or is requested only after an attempt is made to access a certain URL. Note: the output produced by this option is not always accurate because a connection might never have been established.

-state

prints out the SSL session states.

-debug

print extensive debugging information including a hex dump of all traffic.

-msg

show all protocol messages with hex dump.

-nbio_test

tests non-blocking I/O

-nbio

turns on non-blocking I/O

-crlf

this option translated a line feed from the terminal into CR+LF as required by some servers.

-ign_eof
inhibit shutting down the connection when end of file is reached in the input.

-quiet
inhibit printing of session and certificate information. This implicitly turns on **-ign_eof** as well.

-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.

-bugs
there are several known bug in SSL and TLS implementations. Adding this option enables various workarounds.

-cipher cipherlist
this allows the cipher list sent by the client to be modified. Although the server determines which cipher suite is used it should take the first supported cipher in the list sent by the client. See the **ciphers** command for more information.

-engine id
specifying an engine (by it‘s unique **id** string) will cause **s_client** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

-rand file(s)
a file or files containing random data used to seed the random number generator, or an EGD socket (see *RAND_egd(3)|RAND_egd(3)*). Multiple files can be specified separated by a OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

CONNECTED COMMANDS

If a connection is established with an SSL server then any data received from the server is displayed and any key presses will be sent to the server. When used interactively (which means neither **-quiet** nor **-ign_eof** have been given), the session will be renegotiated if the line begins with an **R**, and if the line begins with a **Q** or if end of file is reached, the connection will be closed down.

NOTES

s_client can be used to debug SSL servers. To connect to an SSL HTTP server the command:

```
openssl s_client -connect servername:443
```

would typically be used (https uses port 443). If the connection succeeds then an HTTP command can be given such as "GET /" to retrieve a web page.
If the handshake fails then there are several possible causes, if it is nothing obvious like no client certificate then the **-bugs**, **-ssl2**, **-ssl3**, **-tls1**, **-no_ssl2**, **-no_ssl3**, **-no_tls1** can be tried in case it is a buggy server. In particular you should play with these options **before** submitting a bug report to an OpenSSL mailing list.
A frequent problem when attempting to get client certificates working is that a web client complains it has no certificates or gives an empty list to choose from. This is normally because the server is not sending the clients certificate authority in its "acceptable CA list" when it requests a certificate. By using **s_client** the CA list can be viewed and checked. However some servers only request client authentication after a specific URL is requested. To obtain the list in this case it is necessary to use the **-prexit** command and send an HTTP request for an appropriate page.
If a certificate is specified on the command line using the **-cert** option it will not be used unless the server specifically requests a client certificate. Therefor merely including a client certificate on the command line is no guarantee that the certificate works.
If there are problems verifying a server certificate then the **-showcerts** option can be used to show the whole chain.

BUGS

Because this program has a lot of options and also because some of the techniques used are rather old, the C source of s_client is rather hard to read and not a model of how things should be done. A typical SSL client program would be much simpler.

The **-verify** option should really exit if the server verification fails.

The **-prexit** option is a bit of a hack. We should really report information whenever a session is renegotiated.

SEE ALSO

sess_id(1)|sess_id(1), *s_server(1)|s_server(1)*, *ciphers(1)|ciphers(1)*

S_SERVER

s_server - SSL/TLS server program

SYNOPSIS

openssl s_server [**-accept port**] [**-context id**] [**-verify depth**] [**-Verify depth**] [**-cert filename**] [**-key keyfile**] [**-dcert filename**] [**-dkey keyfile**] [**-dhparam filename**] [**-nbio**] [**-nbio_test**] [**-crlf**] [**-debug**] [**-msg**] [**-state**] [**-CApath directory**] [**-CAfile filename**] [**-nocert**] [**-cipher cipherlist**] [**-quiet**] [**-no_tmp_rsa**] [**-ssl2**] [**-ssl3**] [**-tls1**] [**-no_ssl2**] [**-no_ssl3**] [**-no_tls1**] [**-no_dhe**] [**-bugs**] [**-hack**] [**-www**] [**-WWW**] [**-HTTP**] [**-engine id**] [**-rand file(s)**]

DESCRIPTION

The **s_server** command implements a generic SSL/TLS server which listens for connections on a given port using SSL/TLS.

OPTIONS

-accept port

the TCP port to listen on for connections. If not specified 4433 is used.

-context id

sets the SSL context id. It can be given any string value. If this option is not present a default value will be used.

-cert certname

The certificate to use, most servers cipher suites require the use of a certificate and some require a certificate with a certain public key type: for example the DSS cipher suites require a certificate containing a DSS (DSA) key. If not specified then the filename "server.pem" will be used.

-key keyfile

The private key to use. If not specified then the certificate file will be used.

-dcert filename, **-dkey keyname**

specify an additional certificate and private key, these behave in the same manner as the **-cert** and **-key** options except there is no default if they are not specified (no additional certificate and key is used). As noted above some cipher suites require a certificate containing a key of a certain type. Some cipher suites need a certificate carrying an RSA key and some a DSS (DSA) key. By using RSA and DSS certificates and keys a server can support clients which only support RSA or DSS cipher suites by using an appropriate certificate.

-nocert

if this option is set then no certificate is used. This restricts the cipher suites available to the anonymous ones (currently just anonymous DH).

-dhparam filename

the DH parameter file to use. The ephemeral DH cipher suites generate keys using a set of DH parameters. If not specified then an attempt is made to load the parameters from the server certificate file. If this fails then a static set of parameters hard coded into the s_server program will be used.

-no_dhe

if this option is set then no DH parameters will be loaded effectively disabling the ephemeral DH cipher suites.

-no_tmp_rsa

certain export cipher suites sometimes use a temporary RSA key, this option disables temporary RSA key generation.

-verify depth, **-Verify depth**

The verify depth to use. This specifies the maximum length of the client certificate chain and makes the server request a certificate from the client. With the **-verify** option a certificate is requested but the client does not have to send one, with the **-Verify** option the client must supply a certificate or an error occurs.

-CApath directory

The directory to use for client certificate verification. This directory must be in "hash format", see **verify** for more information. These are also used when building the server certificate chain.

-CAfile file

A file containing trusted certificates to use during client authentication and to use when attempting to build the server certificate chain. The list is also used in the list of acceptable client CAs passed to the client when a certificate is requested.

-state
prints out the SSL session states.

-debug
print extensive debugging information including a hex dump of all traffic.

-msg
show all protocol messages with hex dump.

-nbio_test
tests non blocking I/O

-nbio
turns on non blocking I/O

-crlf
this option translated a line feed from the terminal into CR+LF.

-quiet
inhibit printing of session and certificate information.

-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.

-bugs
there are several known bug in SSL and TLS implementations. Adding this option enables various workarounds.

-hack
this option enables a further workaround for some some early Netscape SSL code (?).

-cipher cipherlist
this allows the cipher list used by the server to be modified. When the client sends a list of supported ciphers the first client cipher also included in the server list is used. Because the client specifies the preference order, the order of the server cipherlist irrelevant. See the **ciphers** command for more information.

-www
sends a status message back to the client when it connects. This includes lots of information about the ciphers used and various session parameters. The output is in HTML format so this option will normally be used with a web browser.

-WWW
emulates a simple web server. Pages will be resolved relative to the current directory, for example if the URL https://myhost/page.html is requested the file ./page.html will be loaded.

-HTTP
emulates a simple web server. Pages will be resolved relative to the current directory, for example if the URL https://myhost/page.html is requested the file ./page.html will be loaded. The files loaded are assumed to contain a complete and correct HTTP response (lines that are part of the HTTP response line and headers must end with CRLF).

-engine id
specifying an engine (by it's unique **id** string) will cause **s_server** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

-rand file(s)
a file or files containing random data used to seed the random number generator, or an EGD socket (see *RAND_egd(3)|RAND_egd(3)*). Multiple files can be specified separated by a OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

CONNECTED COMMANDS

If a connection request is established with an SSL client and neither the **-www** nor the **-WWW** option has been used then normally any data received from the client is displayed and any key presses will be sent to the client.

Certain single letter commands are also recognized which perform special operations: these are listed below.

q
end the current SSL connection but still accept new connections.

Q
end the current SSL connection and exit.

r
renegotiate the SSL session.

R
renegotiate the SSL session and request a client certificate.

P
send some plain text down the underlying TCP connection: this should cause the client to disconnect due to a protocol violation.

S
print out some session cache status information.

NOTES

s_server can be used to debug SSL clients. To accept connections from a web browser the command:

```
openssl s_server -accept 443 -www
```

can be used for example.

Most web browsers (in particular Netscape and MSIE) only support RSA cipher suites, so they cannot connect to servers which don‘t use a certificate carrying an RSA key or a version of OpenSSL with RSA disabled.

Although specifying an empty list of CAs when requesting a client certificate is strictly speaking a protocol violation, some SSL clients interpret this to mean any CA is acceptable. This is useful for debugging purposes.

The session parameters can printed out using the **sess_id** program.

BUGS

Because this program has a lot of options and also because some of the techniques used are rather old, the C source of s_server is rather hard to read and not a model of how things should be done. A typical SSL server program would be much simpler.

The output of common ciphers is wrong: it just gives the list of ciphers that OpenSSL recognizes and the client supports.

There should be a way for the **s_server** program to print out details of any unknown cipher suites a client says it supports.

SEE ALSO

sess_id(1)|sess_id(1), *s_client(1)|s_client(1)*, *ciphers(1)|ciphers(1)*

TS

ts - Time Stamping Authority tool (client/server)

SYNOPSIS

openssl ts -query [**-rand** file:file...] [**-config** configfile] [**-data** file_to_hash] [**-digest** digest_bytes] [**-md2**|**-md4**|**-md5**|**-sha**|**-sha1**|**-mdc2**|**-ripemd160**] [**-policy** object_id] [**-no_nonce**] [**-cert**] [**-in** request.tsq] [**-out** request.tsq] [**-text**]
openssl ts -reply [**-config** configfile] [**-section** tsa_section] [**-queryfile** request.tsq] [**-passin** password_src] [**-signer** tsa_cert.pem] [**-inkey** private.pem] [**-chain** certs_file.pem] [**-policy** object_id] [**-in** response.tsr] [**-token_in**] [**-out** response.tsr] [**-token_out**] [**-text**]
openssl ts -verify [**-data** file_to_hash] [**-digest** digest_bytes] [**-queryfile** request.tsq] [**-in** response.tsr] [**-token_in**] [**-CApath** trusted_cert_path] [**-CAfile** trusted_certs.pem] [**-untrusted** cert_file.pem]

DESCRIPTION

The **ts** command is a basic Time Stamping Authority (TSA) client and server application as specified in RFC 3161 (Time-Stamp Protocol, TSP). A TSA can be part of a PKI deployment and its role is to provide long term proof of the existence of a certain datum before a particular time. Here is a brief description of the protocol:

1. The TSA client computes a one-way hash value for a data file and sends the hash to the TSA.
2. The TSA attaches the current date and time to the received hash value, signs them and sends the time stamp token back to the client. By creating this token the TSA certifies the existence of the original data file at the time of response generation.
3. The TSA client receives the time stamp token and verifies the signature on it. It also checks if the token contains the same hash value that it had sent to the TSA.

There is one DER encoded protocol data unit defined for transporting a time stamp request to the TSA and one for sending the time stamp response back to the client. The **ts** command has three main functions: creating a time stamp request based on a data file, creating a time stamp response based on a request, verifying if a response corresponds to a particular request or a data file.
There is no support for sending the requests/responses automatically over HTTP or TCP yet as suggested in RFC 3161. The users must send the requests either by ftp or e-mail.

OPTIONS

Time Stamp Request generation

The **-query** switch can be used for creating and printing a time stamp request with the following options:

-rand file:file...
: The files containing random data for seeding the random number generator. Multiple files can be specified, the separator is **;** for MS-Windows, **,** for VMS and **:** for all other platforms. (Optional)

-config configfile
: The configuration file to use, this option overrides the **OPENSSL_CONF** environment variable. Only the OID section of the config file is used with the **-query** command. (Optional)

-data file_to_hash
: The data file for which the time stamp request needs to be created. stdin is the default if neither the **-data** nor the **-digest** parameter is specified. (Optional)

[**-digest** digest_bytes]
: It is possible to specify the message imprint explicitly without the data file. The imprint must be specified in a hexadecimal format, two characters per byte, the bytes optionally separated by colons (e.g. 1A:F6:01:... or 1AF601...). The number of bytes must match the message digest algorithm in use. (Optional)

-md2|-md4|-md5|-sha|-sha1|-mdc2|-ripemd160
: The message digest to apply to the data file. The default is SHA-1. (Optional)

-policy object_id
: The policy that the client expects the TSA to use for creating the time stamp token. Either the dotted OID notation or OID names defined in the config file can be used. If no policy is requested

the TSA will use its own default policy. (Optional)

-no_nonce
No nonce is specified in the request if this option is given. Otherwise a 64 bit long pseudo-random none is included in the request. It is recommended to use nonce to protect against replay-attacks. (Optional)

-cert
The TSA is expected to include its signing certificate in the response. (Optional)

-in request.tsq
This option specifies a previously created time stamp request in DER format that will be printed into the output file. Useful when you need to examine the content of a request in human-readable format. (Optional)

-out request.tsq
Name of the output file to which the request will be written. Default is stdout. (Optional)

-text
If this option is specified the output is human-readable text format instead of DER. (Optional)

Time Stamp Response generation

A time stamp response (TimeStampResp) consists of a response status and the time stamp token itself (ContentInfo), if the token generation was successful. The **-reply** command is for creating a time stamp response or time stamp token based on a request and printing the response/token in human-readable format. If **-token_out** is not specified the output is always a time stamp response (TimeStampResp), otherwise it is a time stamp token (ContentInfo).

[**-config** configfile]
The configuration file to use, this option overrides the **OPENSSL_CONF** environment variable. See **CONFIGURATION FILE OPTIONS** for configurable variables. (Optional)

[**-section** tsa_section]
The name of the config file section conatining the settings for the response generation. If not specified the default TSA section is used, see **CONFIGURATION FILE OPTIONS** for details. (Optional)

[**-queryfile** request.tsq]
The name of the file containing a DER encoded time stamp request. (Optional)

[**-passin** password_src]
Specifies the password source for the private key of the TSA. See **PASS PHRASE ARGUMENTS** in *openssl(1)|openssl(1)*. (Optional)

[**-signer** tsa_cert.pem]
The signer certificate of the TSA in PEM format. The TSA signing certificate must have exactly one extended key usage assigned to it: timeStamping. Overrides the **signer_cert** variable of the config file. (Optional)

[**-inkey** private.pem]
The signer private key of the TSA in PEM format. Overrides the **signer_key** config file option. (Optional)

[**-chain** certs_file.pem]
The collection of certificates in PEM format that will all be included in the response in addition to the signer certificate if the **-cert** option was used for the request. This file is supposed to contain the certificate chain for the signer certificate from its issuer upwards. The **-reply** command does not build a certificate chain automatically. (Optional)

[**-policy** object_id]
The default policy to use for the response unless the client explicitly requires a particular TSA policy. The OID can be specified either in dotted notation or with its name. Overrides the **default_policy** config file option. (Optional)

[**-in** response.tsr]
Specifies a previously created time stamp response or time stamp token (if **-token_in** is also specified) in DER format that will be written to the output file. This option does not require a request, it is useful e.g. when you need to examine the content of a response or token or you want to extract the time stamp token from a response. If the input is a token and the output is a time

stamp response a default 'granted' status info is added to the token. (Optional)

[**-token_in**]
This flag can be used together with the **-in** option and indicates that the input is a DER encoded time stamp token (ContentInfo) instead of a time stamp response (TimeStampResp). (Optional)

[**-out** response.tsr]
The response is written to this file. The format and content of the file depends on other options (see **-text**, **-token_out**). The default is stdout. (Optional)

[**-token_out**]
The output is a time stamp token (ContentInfo) instead of time stamp response (TimeStampResp). (Optional)

[**-text**]
If this option is specified the output is human-readable text format instead of DER. (Optional)

Time Stamp Response verification

The **-verify** command is for verifying if a time stamp response or time stamp token is valid and matches a particular time stamp request or data file. The **-verify** command does not use the configuration file.

[**-data** file_to_hash]
The response or token must be verified against file_to_hash. The file is hashed with the message digest algorithm specified in the token. The **-digest** and **-queryfile** options must not be specified with this one. (Optional)

[**-digest** digest_bytes]
The response or token must be verified against the message digest specified with this option. The number of bytes must match the message digest algorithm specified in the token. The **-data** and **-queryfile** options must not be specified with this one. (Optional)

[**-queryfile** request.tsq]
The original time stamp request in DER format. The **-data** and **-digest** options must not be specified with this one. (Optional)

[**-in** response.tsr]
The time stamp response that needs to be verified in DER format. (Mandatory)

[**-token_in**]
This flag can be used together with the **-in** option and indicates that the input is a DER encoded time stamp token (ContentInfo) instead of a time stamp response (TimeStampResp). (Optional)

[**-CApath** trusted_cert_path]
The name of the directory containing the trused CA certificates of the client. See the similar option of *verify(1)|verify(1)* for additional details. Either this option or **-CAfile** must be specified. (Optional)

[**-CAfile** trusted_certs.pem]
The name of the file containing a set of trusted self-signed CA certificates in PEM format. See the similar option of *verify(1)|verify(1)* for additional details. Either this option or **-CApath** must be specified. (Optional)

[**-untrusted** cert_file.pem]
Set of additional untrusted certificates in PEM format which may be needed when building the certificate chain for the TSA's signing certificate. This file must contain the TSA signing certificate and all intermediate CA certificates unless the response includes them. (Optional)

CONFIGURATION FILE OPTIONS

The **-query** and **-reply** commands make use of a configuration file defined by the **OPENSSL_CONF** environment variable. See *config(5)|config(5* for a general description of the syntax of the config file. The **-query** command uses only the symbolic OID names section and it can work without it. However, the **-reply** command needs the config file for its operation.
When there is a command line switch equivalent of a variable the switch always overrides the settings in the config file.

tsa section, **default_tsa**

This is the main section and it specifies the name of another section that contains all the options for the **-reply** command. This default section can be overriden with the **-section** command line switch. (Optional)

oid_file
See *ca(1)|ca(1)* for description. (Optional)

oid_section
See *ca(1)|ca(1)* for description. (Optional)

RANDFILE
See *ca(1)|ca(1)* for description. (Optional)

serial
The name of the file containing the hexadecimal serial number of the last time stamp response created. This number is incremented by 1 for each response. If the file does not exit at the time of response generation a new file is created with serial number 1. (Mandatory)

signer_cert
TSA signing certificate in PEM format. The same as the **-signer** command line option. (Optional)

certs
A file containing a set of PEM encoded certificates that need to be included in the response. The same as the **-chain** command line option. (Optional)

signer_key
The private key of the TSA in PEM format. The same as the **-inkey** command line option. (Optional)

default_policy
The default policy to use when the request does not mandate any policy. The same as the **-policy** command line option. (Optional)

other_policies
Comma separated list of policies that are also acceptable by the TSA and used only if the request explicitly specifies one of them. (Optional)

digests
The list of message digest algorithms that the TSA accepts. At least one algorithm must be specified. (Mandatory)

accuracy
The accuracy of the time source of the TSA in seconds, milliseconds and microseconds. E.g. secs:1, millisecs:500, microsecs:100. If any of the components is missing zero is assumed for that field. (Optional)

ordering
If this option is yes the responses generated by this TSA can always be ordered, even if the time difference between two responses is less than the sum of their accuracies. Default is no. (Optional)

tsa_name
Set this option to yes if the subject name of the TSA must be included in the TSA name field of the response. Default is no. (Optional)

ess_cert_id_chain
The SignedData objects created by the TSA always contain the certificate identifier of the signing certificate in a signed attribute (see RFC 2634, Enhanced Security Services). If this option is set to yes and either the **certs** variable or the **-chain** option is specified then the certificate identifiers of the chain will also be included in the SigningCertificate signed attribute. If this variable is set to no, only the signing certificate identifier is included. Default is no. (Optional)

ENVIRONMENT VARIABLES

OPENSSL_CONF contains the path of the configuration file and can be overriden by the **-config** command line option.

EXAMPLES

All the examples below presume that **OPENSSL_CONF** is set to a proper configuration file, e.g. the example configuration file openssl/apps/openssl.cnf will do.

Time Stamp Request

To create a time stamp request for design1.txt with SHA-1 without nonce and policy and no certificate is required in the response:

```
openssl ts -query -data design1.txt -no_nonce \
      -out design1.tsq
```

To create a similar time stamp request with specifying the message imprint explicitly:

```
openssl ts -query -digest b7e5d3f93198b38379852f2c04e78d73abdd0f4b \
       -no_nonce -out design1.tsq
```

To print the content of the previous request in human readable format:

```
openssl ts -query -in design1.tsq -text
```

To create a time stamp request which includes the MD-5 digest of design2.txt, requests the signer certificate and nonce, specifies a policy id (assuming the tsa_policy1 name is defined in the OID section of the config file):

```
openssl ts -query -data design2.txt -md5 \
      -policy tsa_policy1 -cert -out design2.tsq
```

Time Stamp Response

Before generating a response a signing certificate must be created for the TSA that contains the **timeStamping** extended key usage extension without any other key usage extensions. You can add the 'extendedKeyUsage = timeStamping' line to the user certificate section of the config file to generate a proper certificate. See *req(1)|req(1)*, *ca(1)|ca(1)*, *x509(1)|x509(1)* for instructions. The examples below assume that cacert.pem contains the certificate of the CA, tsacert.pem is the signing certificate issued by cacert.pem and tsakey.pem is the private key of the TSA.

To create a time stamp response for a request:

```
openssl ts -reply -queryfile design1.tsq -inkey tsakey.pem \
      -signer tsacert.pem -out design1.tsr
```

If you want to use the settings in the config file you could just write:

```
openssl ts -reply -queryfile design1.tsq -out design1.tsr
```

To print a time stamp reply to stdout in human readable format:

```
openssl ts -reply -in design1.tsr -text
```

To create a time stamp token instead of time stamp response:

```
openssl ts -reply -queryfile design1.tsq -out design1_token.der -token_out
```

To print a time stamp token to stdout in human readable format:

```
openssl ts -reply -in design1_token.der -token_in -text -token_out
```

To extract the time stamp token from a response:

```
openssl ts -reply -in design1.tsr -out design1_token.der -token_out
```

To add 'granted' status info to a time stamp token thereby creating a valid response:

```
openssl ts -reply -in design1_token.der -token_in -out design1.tsr
```

Time Stamp Verification

To verify a time stamp reply against a request:

```
openssl ts -verify -queryfile design1.tsq -in design1.tsr \
      -CAfile cacert.pem -untrusted tsacert.pem
```

To verify a time stamp reply that includes the certificate chain:

```
openssl ts -verify -queryfile design2.tsq -in design2.tsr \
      -CAfile cacert.pem
```

To verify a time stamp token against the original data file:

```
openssl ts -verify -data design2.txt -in design2.tsr \
      -CAfile cacert.pem
```

To verify a time stamp token against a message imprint:

```
openssl ts -verify -digest b7e5d3f93198b38379852f2c04e78d73abdd0f4b \
        -in design2.tsr -CAfile cacert.pem
```

You could also look at the 'test' directory for more examples.

BUGS

If you find any bugs or you have suggestions please write to Zoltan Glozik <zglozik@stones.com>. Known issues:

- No support for TCP or HTTP transport between client and server. It is quite easy to implement an automatic e-mail based TSA with *procmail(1)|procmail(1)* and *perl(1)|perl(1)*. HTTP support will come soon in the form of an apache module.
- The file containing the last serial number of the TSA is not locked when being read or written. This is a problem if more than one instance of *openssl(1)|openssl(1)* is trying to create a time stamp response at the same time.
- Look for the FIXME word in the source files.
- The source code should really be reviewed by somebody else, too.
- More testing is needed, I have done only some basic tests (see test/testtsa).

AUTHOR

- This patch kit was written by Zoltan Glozik <zglozik@stones.com> for the OpenSSL project. See http://www.opentsa.org/.
- This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.OpenSSL.org/).
- This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).
- This product includes software written by Tim Hudson (tjh@cryptsoft.com).

SEE ALSO

openssl(1)|openssl(1), *req(1)|req(1)*, *x509(1)|x509(1)*, *ca(1)|ca(1)*, *genrsa(1)|genrsa(1)*, *config(5)|config(5)*

TSGET

tsget - Time Stamping HTTP/HTTPS client

SYNOPSIS

tsget **-h** server_url [**-e** extension] [**-o** output] [**-v**] [**-d**] [**-k** private_key.pem] [**-p** key_password] [**-c** client_cert.pem] [**-C** CA_certs.pem] [**-P** CA_path] [**-r** file:file...] [**-g** EGD_socket] [request]...

DESCRIPTION

The **tsget** command can be used for sending a time stamp request, as specified in **RFC 3161**, to a time stamp server over HTTP or HTTPS and storing the time stamp response in a file. This tool cannot be used for creating the requests and verifying responses, you can use the OpenSSL **ts(1)** command to do that. **tsget** can send several requests to the server without closing the TCP connection if more than one requests are specified on the command line.

The tool sends the following HTTP request for each time stamp request:

```
POST url HTTP/1.1
User-Agent: OpenTSA tsget.pl/<version>
Host: <host>:<port>
Pragma: no-cache
Content-Type: application/timestamp-query
Accept: application/timestamp-reply
Content-Length: length of body

...binary request specified by the user...
```

tsget expects a response of type application/timestamp-reply, which is written to a file without any interpretation.

OPTIONS

-h server_url

The URL of the HTTP/HTTPS server listening for time stamp requests.

-e extension

If the **-o** option is not given this argument specifies the extension of the output files. The base name of the output file will be the same as those of the input files. Default extension is '.tsr'. (Optional)

-o output

This option can be specified only when just one request is sent to the server. The time stamp response will be written to the given output file. '-' means standard output. In case of multiple time stamp requests or the absence of this argument the names of the output files will be derived from the names of the input files and the default or specified extension argument. (Optional)

-v

The name of the currently processed request is printed on standard error. (Optional)

-d

Switches on verbose mode for the underlying **curl** library. You can see detailed debug messages for the connection. (Optional)

-k private_key.pem

(HTTPS) In case of certificate-based client authentication over HTTPS <private_key.pem> must contain the private key of the user. The private key file can optionally be protected by a passphrase. The **-c** option must also be specified. (Optional)

-p key_password

(HTTPS) Specifies the passphrase for the private key specified by the **-k** argument. If this option is omitted and the key is passphrase protected **tsget** will ask for it. (Optional)

-c client_cert.pem

(HTTPS) In case of certificate-based client authentication over HTTPS <client_cert.pem> must contain the X.509 certificate of the user. The **-k** option must also be specified. If this option is not specified no certificate-based client authentication will take place. (Optional)

-C CA_certs.pem

(HTTPS) The trusted CA certificate store. The certificate chain of the peer's certificate must include one of the CA certificates specified in this file. Either option **-C** or option **-P** must be given

in case of HTTPS. (Optional)

-P CA_path
: (HTTPS) The path containing the trusted CA certificates to verify the peer‘s certificate. The directory must be prepared with the **c_rehash** OpenSSL utility. Either option **-C** or option **-P** must be given in case of HTTPS. (Optional)

-rand file:file...
: The files containing random data for seeding the random number generator. Multiple files can be specified, the separator is **;** for MS-Windows, **,** for VMS and **:** for all other platforms. (Optional)

-g EGD_socket
: The name of an EGD socket to get random data from. (Optional)

[request]...
: List of files containing **RFC 3161** DER-encoded time stamp requests. If no requests are specifed only one request will be sent to the server and it will be read from the standard input. (Optional)

ENVIRONMENT VARIABLES

The **TSGET** environment variable can optionally contain default arguments. The content of this variable is added to the list of command line arguments.

EXAMPLES

The examples below presume that **file1.tsq** and **file2.tsq** contain valid time stamp requests, tsa.opentsa.org listens at port 8080 for HTTP requests and at port 8443 for HTTPS requests, the TSA service is available at the /tsa absolute path.

Get a time stamp response for file1.tsq over HTTP, output is written to file1.tsr:

```
tsget -h http://tsa.opentsa.org:8080/tsa file1.tsq
```

Get a time stamp response for file1.tsq and file2.tsq over HTTP showing progress, output is written to file1.reply and file2.reply respectively:

```
tsget -h http://tsa.opentsa.org:8080/tsa -v -e .reply \
      file1.tsq file2.tsq
```

Create a time stamp request, write it to file3.tsq, send it to the server and write the response to file3.tsr:

```
openssl ts -query -data file3.txt -cert | tee file3.tsq \
      | tsget -h http://tsa.opentsa.org:8080/tsa \
      -o file3.tsr
```

Get a time stamp response for file1.tsq over HTTPS without client authentication:

```
tsget -h https://tsa.opentsa.org:8443/tsa \
      -C cacerts.pem file1.tsq
```

Get a time stamp response for file1.tsq over HTTPS with certificate-based client authentication (it will ask for the passphrase if client_key.pem is protected):

```
tsget -h https://tsa.opentsa.org:8443/tsa -C cacerts.pem \
      -k client_key.pem -c client_cert.pem file1.tsq
```

You can shorten the previous command line if you make use of the **TSGET** environment variable. The following commands do the same as the previous example:

```
TSGET='-h https://tsa.opentsa.org:8443/tsa -C cacerts.pem \
      -k client_key.pem -c client_cert.pem'
export TSGET
tsget file1.tsq
```

BUGS

If you find any bugs or you have suggestions please write to info@opentsa.org.

AUTHOR

This tool was written by Zoltan Glozik <zglozik@stones.com> for the OpenTSA project. See http://www.opentsa.org/.

SEE ALSO

openssl(1)|openssl(1), *ts(1)|ts(1)*, *curl(1)|curl(1)*, **RFC 3161**

VERIFY

verify - Utility to verify certificates.

SYNOPSIS

openssl verify [**-CApath directory**] [**-CAfile file**] [**-purpose purpose**] [**-untrusted file**] [**-help**] [**-issuer_checks**] [**-verbose**] [**-**] [certificates]

DESCRIPTION

The **verify** command verifies certificate chains.

COMMAND OPTIONS

-CApath directory

A directory of trusted certificates. The certificates should have names of the form: hash.0 or have symbolic links to them of this form ("hash" is the hashed certificate subject name: see the **-hash** option of the **x509** utility). Under Unix the **c_rehash** script will automatically create symbolic links to a directory of certificates.

-CAfile file

A file of trusted certificates. The file should contain multiple certificates in PEM format concatenated together.

-untrusted file

A file of untrusted certificates. The file should contain multiple certificates

-purpose purpose

the intended use for the certificate. Without this option no chain verification will be done. Currently accepted uses are **sslclient**, **sslserver**, **nssslserver**, **smimesign**, **smimeencrypt**. See the **VERIFY OPERATION** section for more information.

-help

prints out a usage message.

-verbose

print extra information about the operations being performed.

-issuer_checks

print out diagnostics relating to searches for the issuer certificate of the current certificate. This shows why each candidate issuer certificate was rejected. However the presence of rejection messages does not itself imply that anything is wrong: during the normal verify process several rejections may take place.

-

marks the last option. All arguments following this are assumed to be certificate files. This is useful if the first certificate filename begins with a **-**.

certificates

one or more certificates to verify. If no certificate filenames are included then an attempt is made to read a certificate from standard input. They should all be in PEM format.

VERIFY OPERATION

The **verify** program uses the same functions as the internal SSL and S/MIME verification, therefore this description applies to these verify operations too.

There is one crucial difference between the verify operations performed by the **verify** program: wherever possible an attempt is made to continue after an error whereas normally the verify operation would halt on the first error. This allows all the problems with a certificate chain to be determined.

The verify operation consists of a number of separate steps.

Firstly a certificate chain is built up starting from the supplied certificate and ending in the root CA. It is an error if the whole chain cannot be built up. The chain is built up by looking up the issuers certificate of the current certificate. If a certificate is found which is its own issuer it is assumed to be the root CA.

The process of 'looking up the issuers certificate' itself involves a number of steps. In versions of OpenSSL before 0.9.5a the first certificate whose subject name matched the issuer of the current certificate was assumed to be the issuers certificate. In OpenSSL 0.9.6 and later all certificates whose subject name matches the issuer name of the current certificate are subject to further tests. The relevant authority key identifier components of the current certificate (if present) must match the subject key

identifier (if present) and issuer and serial number of the candidate issuer, in addition the keyUsage extension of the candidate issuer (if present) must permit certificate signing.
The lookup first looks in the list of untrusted certificates and if no match is found the remaining lookups are from the trusted certificates. The root CA is always looked up in the trusted certificate list: if the certificate to verify is a root certificate then an exact match must be found in the trusted list.
The second operation is to check every untrusted certificate‘s extensions for consistency with the supplied purpose. If the **-purpose** option is not included then no checks are done. The supplied or "leaf" certificate must have extensions compatible with the supplied purpose and all other certificates must also be valid CA certificates. The precise extensions required are described in more detail in the **CERTIFICATE EXTENSIONS** section of the **x509** utility.
The third operation is to check the trust settings on the root CA. The root CA should be trusted for the supplied purpose. For compatibility with previous versions of SSLeay and OpenSSL a certificate with no trust settings is considered to be valid for all purposes.
The final operation is to check the validity of the certificate chain. The validity period is checked against the current system time and the notBefore and notAfter dates in the certificate. The certificate signatures are also checked at this point.
If all operations complete successfully then certificate is considered valid. If any operation fails then the certificate is not valid.

DIAGNOSTICS

When a verify operation fails the output messages can be somewhat cryptic. The general form of the error message is:

```
server.pem: /C=AU/ST=Queensland/O=CryptSoft Pty Ltd/CN=Test CA (1024 bit)
error 24 at 1 depth lookup:invalid CA certificate
```

The first line contains the name of the certificate being verified followed by the subject name of the certificate. The second line contains the error number and the depth. The depth is number of the certificate being verified when a problem was detected starting with zero for the certificate being verified itself then 1 for the CA that signed the certificate and so on. Finally a text version of the error number is presented.
An exhaustive list of the error codes and messages is shown below, this also includes the name of the error code as defined in the header file x509_vfy.h Some of the error codes are defined but never returned: these are described as "unused".

0 X509_V_OK: ok
the operation was successful.

2 X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT: unable to get issuer certificate
the issuer certificate could not be found: this occurs if the issuer certificate of an untrusted certificate cannot be found.

3 X509_V_ERR_UNABLE_TO_GET_CRL unable to get certificate CRL
the CRL of a certificate could not be found. Unused.

4 X509_V_ERR_UNABLE_TO_DECRYPT_CERT_SIGNATURE: unable to decrypt certificate‘s signature
the certificate signature could not be decrypted. This means that the actual signature value could not be determined rather than it not matching the expected value, this is only meaningful for RSA keys.

5 X509_V_ERR_UNABLE_TO_DECRYPT_CRL_SIGNATURE: unable to decrypt CRL‘s signature
the CRL signature could not be decrypted: this means that the actual signature value could not be determined rather than it not matching the expected value. Unused.

6 X509_V_ERR_UNABLE_TO_DECODE_ISSUER_PUBLIC_KEY: unable to decode issuer public key
the public key in the certificate SubjectPublicKeyInfo could not be read.

7 X509_V_ERR_CERT_SIGNATURE_FAILURE: certificate signature failure
the signature of the certificate is invalid.

8 X509_V_ERR_CRL_SIGNATURE_FAILURE: CRL signature failure
the signature of the certificate is invalid. Unused.

9 X509_V_ERR_CERT_NOT_YET_VALID: certificate is not yet valid

the certificate is not yet valid: the notBefore date is after the current time.

10 X509_V_ERR_CERT_HAS_EXPIRED: certificate has expired

the certificate has expired: that is the notAfter date is before the current time.

11 X509_V_ERR_CRL_NOT_YET_VALID: CRL is not yet valid

the CRL is not yet valid. Unused.

12 X509_V_ERR_CRL_HAS_EXPIRED: CRL has expired

the CRL has expired. Unused.

13 X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD: format error in certificate's notBefore field

the certificate notBefore field contains an invalid time.

14 X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD: format error in certificate's notAfter field

the certificate notAfter field contains an invalid time.

15 X509_V_ERR_ERROR_IN_CRL_LAST_UPDATE_FIELD: format error in CRL's lastUpdate field

the CRL lastUpdate field contains an invalid time. Unused.

16 X509_V_ERR_ERROR_IN_CRL_NEXT_UPDATE_FIELD: format error in CRL's nextUpdate field

the CRL nextUpdate field contains an invalid time. Unused.

17 X509_V_ERR_OUT_OF_MEM: out of memory

an error occurred trying to allocate memory. This should never happen.

18 X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT: self signed certificate

the passed certificate is self signed and the same certificate cannot be found in the list of trusted certificates.

19 X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN: self signed certificate in certificate chain

the certificate chain could be built up using the untrusted certificates but the root could not be found locally.

20 X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY: unable to get local issuer certificate

the issuer certificate of a locally looked up certificate could not be found. This normally means the list of trusted certificates is not complete.

21 X509_V_ERR_UNABLE_TO_VERIFY_LEAF_SIGNATURE: unable to verify the first certificate

no signatures could be verified because the chain contains only one certificate and it is not self signed.

22 X509_V_ERR_CERT_CHAIN_TOO_LONG: certificate chain too long

the certificate chain length is greater than the supplied maximum depth. Unused.

23 X509_V_ERR_CERT_REVOKED: certificate revoked

the certificate has been revoked. Unused.

24 X509_V_ERR_INVALID_CA: invalid CA certificate

a CA certificate is invalid. Either it is not a CA or its extensions are not consistent with the supplied purpose.

25 X509_V_ERR_PATH_LENGTH_EXCEEDED: path length constraint exceeded

the basicConstraints pathlength parameter has been exceeded.

26 X509_V_ERR_INVALID_PURPOSE: unsupported certificate purpose

the supplied certificate cannot be used for the specified purpose.

27 X509_V_ERR_CERT_UNTRUSTED: certificate not trusted

the root CA is not marked as trusted for the specified purpose.

28 X509_V_ERR_CERT_REJECTED: certificate rejected

the root CA is marked to reject the specified purpose.

29 X509_V_ERR_SUBJECT_ISSUER_MISMATCH: subject issuer mismatch

the current candidate issuer certificate was rejected because its subject name did not match the issuer name of the current certificate. Only displayed when the **-issuer_checks** option is set.

30 X509_V_ERR_AKID_SKID_MISMATCH: authority and subject key identifier mismatch

the current candidate issuer certificate was rejected because its subject key identifier was present and did not match the authority key identifier current certificate. Only displayed when the **-issuer_checks** option is set.

31 X509_V_ERR_AKID_ISSUER_SERIAL_MISMATCH: authority and issuer serial number mismatch

the current candidate issuer certificate was rejected because its issuer name and serial number was present and did not match the authority key identifier of the current certificate. Only displayed when the **-issuer_checks** option is set.

32 X509_V_ERR_KEYUSAGE_NO_CERTSIGN:key usage does not include certificate signing

the current candidate issuer certificate was rejected because its keyUsage extension does not permit certificate signing.

50 X509_V_ERR_APPLICATION_VERIFICATION: application verification failure

an application specific error. Unused.

BUGS

Although the issuer checks are a considerably improvement over the old technique they still suffer from limitations in the underlying X509_LOOKUP API. One consequence of this is that trusted certificates with matching subject name must either appear in a file (as specified by the **-CAfile** option) or a directory (as specified by **-CApath**. If they occur in both then only the certificates in the file will be recognised.

Previous versions of OpenSSL assume certificates with matching subject name are identical and mishandled them.

SEE ALSO

x509(1)|x509(1)

VERSION

version - print OpenSSL version information

SYNOPSIS

openssl version [**-a**] [**-v**] [**-b**] [**-o**] [**-f**] [**-p**]

DESCRIPTION

This command is used to print out version information about OpenSSL.

OPTIONS

-a
all information, this is the same as setting all the other flags.

-v
the current OpenSSL version.

-b
the date the current version of OpenSSL was built.

-o
option information: various options set when the library was built.

-c
compilation flags.

-p
platform setting.

-d
OPENSSLDIR setting.

NOTES

The output of **openssl version -a** would typically be used when sending in a bug report.

HISTORY

The **-d** option was added in OpenSSL 0.9.7.

X509

x509 - Certificate display and signing utility

SYNOPSIS

openssl x509 [**-inform DER|PEM|NET**] [**-outform DER|PEM|NET**] [**-keyform DER|PEM**] [**-CAform DER|PEM**] [**-CAkeyform DER|PEM**] [**-in filename**] [**-out filename**] [**-serial**] [**-hash**] [**-subject**] [**-issuer**] [**-nameopt option**] [**-email**] [**-startdate**] [**-enddate**] [**-purpose**] [**-dates**] [**-modulus**] [**-fingerprint**] [**-alias**] [**-noout**] [**-trustout**] [**-clrtrust**] [**-clrreject**] [**-addtrust arg**] [**-addreject arg**] [**-setalias arg**] [**-days arg**] [**-set_serial n**] [**-signkey filename**] [**-x509toreq**] [**-req**] [**-CA filename**] [**-CAkey filename**] [**-CAcreateserial**] [**-CAserial filename**] [**-text**] [**-C**] [**-md2|-md5|-sha1|-mdc2**] [**-clrext**] [**-extfile filename**] [**-extensions section**] [**-engine id**]

DESCRIPTION

The **x509** command is a multi purpose certificate utility. It can be used to display certificate information, convert certificates to various forms, sign certificate requests like a "mini CA" or edit certificate trust settings.

Since there are a large number of options they will split up into various sections.

OPTIONS

INPUT, OUTPUT AND GENERAL PURPOSE OPTIONS

-inform DER|PEM|NET

This specifies the input format normally the command will expect an X509 certificate but this can change if other options such as **-req** are present. The DER format is the DER encoding of the certificate and PEM is the base64 encoding of the DER encoding with header and footer lines added. The NET option is an obscure Netscape server format that is now obsolete.

-outform DER|PEM|NET

This specifies the output format, the options have the same meaning as the **-inform** option.

-in filename

This specifies the input filename to read a certificate from or standard input if this option is not specified.

-out filename

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

-md2|-md5|-sha1|-mdc2

the digest to use. This affects any signing or display option that uses a message digest, such as the **-fingerprint**, **-signkey** and **-CA** options. If not specified then MD5 is used. If the key being used to sign with is a DSA key then this option has no effect: SHA1 is always used with DSA keys.

-engine id

specifying an engine (by it's unique **id** string) will cause **req** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

DISPLAY OPTIONS

Note: the **-alias** and **-purpose** options are also display options but are described in the **TRUST SETTINGS** section.

-text

prints out the certificate in text form. Full details are output including the public key, signature algorithms, issuer and subject names, serial number any extensions present and any trust settings.

-certopt option

customise the output format used with **-text**. The **option** argument can be a single option or multiple options separated by commas. The **-certopt** switch may be also be used more than once to set multiple options. See the **TEXT OPTIONS** section for more information.

-noout

this option prevents output of the encoded version of the request.

-modulus

this option prints out the value of the modulus of the public key contained in the certificate.

-serial

outputs the certificate serial number.

-hash

outputs the "hash" of the certificate subject name. This is used in OpenSSL to form an index to allow certificates in a directory to be looked up by subject name.

-subject

outputs the subject name.

-issuer

outputs the issuer name.

-nameopt option

option which determines how the subject or issuer names are displayed. The **option** argument can be a single option or multiple options separated by commas. Alternatively the **-nameopt** switch may be used more than once to set multiple options. See the **NAME OPTIONS** section for more information.

-email

outputs the email address(es) if any.

-startdate

prints out the start date of the certificate, that is the notBefore date.

-enddate

prints out the expiry date of the certificate, that is the notAfter date.

-dates

prints out the start and expiry dates of a certificate.

-fingerprint

prints out the digest of the DER encoded version of the whole certificate (see digest options).

-C

this outputs the certificate in the form of a C source file.

TRUST SETTINGS

Please note these options are currently experimental and may well change.

A **trusted certificate** is an ordinary certificate which has several additional pieces of information attached to it such as the permitted and prohibited uses of the certificate and an "alias".

Normally when a certificate is being verified at least one certificate must be "trusted". By default a trusted certificate must be stored locally and must be a root CA: any certificate chain ending in this CA is then usable for any purpose.

Trust settings currently are only used with a root CA. They allow a finer control over the purposes the root CA can be used for. For example a CA may be trusted for SSL client but not SSL server use.

See the description of the **verify** utility for more information on the meaning of trust settings.

Future versions of OpenSSL will recognize trust settings on any certificate: not just root CAs.

-trustout

this causes **x509** to output a **trusted** certificate. An ordinary or trusted certificate can be input but by default an ordinary certificate is output and any trust settings are discarded. With the **-trustout** option a trusted certificate is output. A trusted certificate is automatically output if any trust settings are modified.

-setalias arg

sets the alias of the certificate. This will allow the certificate to be referred to using a nickname for example "Steve's Certificate".

-alias

outputs the certificate alias, if any.

-clrtrust

clears all the permitted or trusted uses of the certificate.

-clrreject

clears all the prohibited or rejected uses of the certificate.

-addtrust arg

adds a trusted certificate use. Any object name can be used here but currently only **clientAuth** (SSL client use), **serverAuth** (SSL server use) and **emailProtection** (S/MIME email) are used. Other OpenSSL applications may define additional uses.

-addreject arg

adds a prohibited use. It accepts the same values as the **-addtrust** option.

-purpose

this option performs tests on the certificate extensions and outputs the results. For a more complete description see the **CERTIFICATE EXTENSIONS** section.

SIGNING OPTIONS

The **x509** utility can be used to sign certificates and requests: it can thus behave like a "mini CA".

-signkey filename

this option causes the input file to be self signed using the supplied private key.

If the input file is a certificate it sets the issuer name to the subject name (i.e. makes it self signed) changes the public key to the supplied value and changes the start and end dates. The start date is set to the current time and the end date is set to a value determined by the **-days** option. Any certificate extensions are retained unless the **-clrext** option is supplied.

If the input is a certificate request then a self signed certificate is created using the supplied private key using the subject name in the request.

-clrext

delete any extensions from a certificate. This option is used when a certificate is being created from another certificate (for example with the **-signkey** or the **-CA** options). Normally all extensions are retained.

-keyform PEM|DER

specifies the format (DER or PEM) of the private key file used in the **-signkey** option.

-days arg

specifies the number of days to make a certificate valid for. The default is 30 days.

-x509toreq

converts a certificate into a certificate request. The **-signkey** option is used to pass the required private key.

-req

by default a certificate is expected on input. With this option a certificate request is expected instead.

-set_serial n

specifies the serial number to use. This option can be used with either the **-signkey** or **-CA** options. If used in conjunction with the **-CA** option the serial number file (as specified by the **-CAserial** or **-CAcreateserial** options) is not used.

The serial number can be decimal or hex (if preceded by **0x**). Negative serial numbers can also be specified but their use is not recommended.

-CA filename

specifies the CA certificate to be used for signing. When this option is present **x509** behaves like a "mini CA". The input file is signed by this CA using this option: that is its issuer name is set to the subject name of the CA and it is digitally signed using the CAs private key.

This option is normally combined with the **-req** option. Without the **-req** option the input is a certificate which must be self signed.

-CAkey filename

sets the CA private key to sign a certificate with. If this option is not specified then it is assumed that the CA private key is present in the CA certificate file.

-CAserial filename

sets the CA serial number file to use.

When the **-CA** option is used to sign a certificate it uses a serial number specified in a file. This file consist of one line containing an even number of hex digits with the serial number to use. After each use the serial number is incremented and written out to the file again.

The default filename consists of the CA certificate file base name with ".srl" appended. For example if the CA certificate file is called "mycacert.pem" it expects to find a serial number file called "mycacert.srl".

-CAcreateserial

with this option the CA serial number file is created if it does not exist: it will contain the serial number "02" and the certificate being signed will have the 1 as its serial number. Normally if the

-CA option is specified and the serial number file does not exist it is an error.

-extfile filename

file containing certificate extensions to use. If not specified then no extensions are added to the certificate.

-extensions section

the section to add certificate extensions from. If this option is not specified then the extensions should either be contained in the unnamed (default) section or the default section should contain a variable called "extensions" which contains the section to use.

NAME OPTIONS

The **nameopt** command line switch determines how the subject and issuer names are displayed. If no **nameopt** switch is present the default "oneline" format is used which is compatible with previous versions of OpenSSL. Each option is described in detail below, all options can be preceded by a **-** to turn the option off. Only the first four will normally be used.

compat

use the old format. This is equivalent to specifying no name options at all.

RFC2253

displays names compatible with RFC2253 equivalent to **esc_2253**, **esc_ctrl**, **esc_msb**, **utf8**, **dump_nostr**, **dump_unknown**, **dump_der**, **sep_comma_plus**, **dn_rev** and **sname**.

oneline

a oneline format which is more readable than RFC2253. It is equivalent to specifying the **esc_2253**, **esc_ctrl**, **esc_msb**, **utf8**, **dump_nostr**, **dump_der**, **use_quote**, **sep_comma_plus_spc**, **spc_eq** and **sname** options.

multiline

a multiline format. It is equivalent **esc_ctrl**, **esc_msb**, **sep_multiline**, **spc_eq**, **lname** and **align**.

esc_2253

escape the "special" characters required by RFC2253 in a field That is **,+"<>;**. Additionally **#** is escaped at the beginning of a string and a space character at the beginning or end of a string.

esc_ctrl

escape control characters. That is those with ASCII values less than 0x20 (space) and the delete (0x7f) character. They are escaped using the RFC2253 \XX notation (where XX are two hex digits representing the character value).

esc_msb

escape characters with the MSB set, that is with ASCII values larger than 127.

use_quote

escapes some characters by surrounding the whole string with **"** characters, without the option all escaping is done with the \ character.

utf8 convert all strings to UTF8 format first. This is required by RFC2253. If you are lucky enough to have a UTF8 compatible terminal then the use of this option (and **not** setting **esc_msb**) may result in the correct display of multibyte (international) characters. Is this option is not present then multibyte characters larger than 0xff will be represented using the format \UXXXX for 16 bits and \WXXXXXXXX for 32 bits. Also if this option is off any UTF8Strings will be converted to their character form first.

no_type

this option does not attempt to interpret multibyte characters in any way. That is their content octets are merely dumped as though one octet represents each character. This is useful for diagnostic purposes but will result in rather odd looking output.

show_type

show the type of the ASN1 character string. The type precedes the field contents. For example "BMPSTRING: Hello World".

dump_der

when this option is set any fields that need to be hexdumped will be dumped using the DER encoding of the field. Otherwise just the content octets will be displayed. Both options use the RFC2253 **#XXXX...** format.

dump_nostr

dump non character string types (for example OCTET STRING) if this option is not set then non character string types will be displayed as though each content octet represents a single character.

dump_all
dump all fields. This option when used with **dump_der** allows the DER encoding of the structure to be unambiguously determined.

dump_unknown
dump any field whose OID is not recognised by OpenSSL.

sep_comma_plus, **sep_comma_plus_space**, **sep_semi_plus_space**, **sep_multiline**
these options determine the field separators. The first character is between RDNs and the second between multiple AVAs (multiple AVAs are very rare and their use is discouraged). The options ending in "space" additionally place a space after the separator to make it more readable. The **sep_multiline** uses a linefeed character for the RDN separator and a spaced + for the AVA separator. It also indents the fields by four characters.

dn_rev
reverse the fields of the DN. This is required by RFC2253. As a side effect this also reverses the order of multiple AVAs but this is permissible.

nofname, **sname**, **lname**, **oid**
these options alter how the field name is displayed. **nofname** does not display the field at all. **sname** uses the "short name" form (CN for commonName for example). **lname** uses the long form. **oid** represents the OID in numerical form and is useful for diagnostic purpose.

align
align field values for a more readable output. Only usable with **sep_multiline**.

spc_eq
places spaces round the = character which follows the field name.

TEXT OPTIONS

As well as customising the name output format, it is also possible to customise the actual fields printed using the **certopt** options when the **text** option is present. The default behaviour is to print all fields.

compatible
use the old format. This is equivalent to specifying no output options at all.

no_header
don‘t print header information: that is the lines saying "Certificate" and "Data".

no_version
don‘t print out the version number.

no_serial
don‘t print out the serial number.

no_signame
don‘t print out the signature algorithm used.

no_validity
don‘t print the validity, that is the **notBefore** and **notAfter** fields.

no_subject
don‘t print out the subject name.

no_issuer
don‘t print out the issuer name.

no_pubkey
don‘t print out the public key.

no_sigdump
don‘t give a hexadecimal dump of the certificate signature.

no_aux
don‘t print out certificate trust information.

no_extensions
don‘t print out any X509V3 extensions.

ext_default

retain default extension behaviour: attempt to print out unsupported certificate extensions.

ext_error
print an error message for unsupported certificate extensions.

ext_parse
ASN1 parse unsupported extensions.

ext_dump
hex dump unsupported extensions.

ca_default
the value used by the **ca** utility, equivalent to **no_issuer**, **no_pubkey**, **no_header**, **no_version**, **no_sigdump** and **no_signame**.

EXAMPLES

Note: in these examples the '\' means the example should be all on one line.

Display the contents of a certificate:

```
openssl x509 -in cert.pem -noout -text
```

Display the certificate serial number:

```
openssl x509 -in cert.pem -noout -serial
```

Display the certificate subject name:

```
openssl x509 -in cert.pem -noout -subject
```

Display the certificate subject name in RFC2253 form:

```
openssl x509 -in cert.pem -noout -subject -nameopt RFC2253
```

Display the certificate subject name in oneline form on a terminal supporting UTF8:

```
openssl x509 -in cert.pem -noout -subject -nameopt oneline,-escmsb
```

Display the certificate MD5 fingerprint:

```
openssl x509 -in cert.pem -noout -fingerprint
```

Display the certificate SHA1 fingerprint:

```
openssl x509 -sha1 -in cert.pem -noout -fingerprint
```

Convert a certificate from PEM to DER format:

```
openssl x509 -in cert.pem -inform PEM -out cert.der -outform DER
```

Convert a certificate to a certificate request:

```
openssl x509 -x509toreq -in cert.pem -out req.pem -signkey key.pem
```

Convert a certificate request into a self signed certificate using extensions for a CA:

```
openssl x509 -req -in careq.pem -extfile openssl.cnf -extensions v3_ca \
       -signkey key.pem -out cacert.pem
```

Sign a certificate request using the CA certificate above and add user certificate extensions:

```
openssl x509 -req -in req.pem -extfile openssl.cnf -extensions v3_usr \
       -CA cacert.pem -CAkey key.pem -CAcreateserial
```

Set a certificate to be trusted for SSL client use and change set its alias to "Steve's Class 1 CA"

```
openssl x509 -in cert.pem -addtrust clientAuth \
       -setalias "Steve's Class 1 CA" -out trust.pem
```

NOTES

The PEM format uses the header and footer lines:

```
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
```

it will also handle files containing:

```
-----BEGIN X509 CERTIFICATE-----
-----END X509 CERTIFICATE-----
```

Trusted certificates have the lines

```
-----BEGIN TRUSTED CERTIFICATE-----
-----END TRUSTED CERTIFICATE-----
```

The conversion to UTF8 format used with the name options assumes that T61Strings use the ISO8859-1 character set. This is wrong but Netscape and MSIE do this as do many certificates. So although this is incorrect it is more likely to display the majority of certificates correctly.

The **-fingerprint** option takes the digest of the DER encoded certificate. This is commonly called a "fingerprint". Because of the nature of message digests the fingerprint of a certificate is unique to that certificate and two certificates with the same fingerprint can be considered to be the same.

The Netscape fingerprint uses MD5 whereas MSIE uses SHA1.

The **-email** option searches the subject name and the subject alternative name extension. Only unique email addresses will be printed out: it will not print the same address more than once.

CERTIFICATE EXTENSIONS

The **-purpose** option checks the certificate extensions and determines what the certificate can be used for. The actual checks done are rather complex and include various hacks and workarounds to handle broken certificates and software.

The same code is used when verifying untrusted certificates in chains so this section is useful if a chain is rejected by the verify code.

The basicConstraints extension CA flag is used to determine whether the certificate can be used as a CA. If the CA flag is true then it is a CA, if the CA flag is false then it is not a CA. **All** CAs should have the CA flag set to true.

If the basicConstraints extension is absent then the certificate is considered to be a "possible CA" other extensions are checked according to the intended use of the certificate. A warning is given in this case because the certificate should really not be regarded as a CA: however it is allowed to be a CA to work around some broken software.

If the certificate is a V1 certificate (and thus has no extensions) and it is self signed it is also assumed to be a CA but a warning is again given: this is to work around the problem of Verisign roots which are V1 self signed certificates.

If the keyUsage extension is present then additional restraints are made on the uses of the certificate. A CA certificate **must** have the keyCertSign bit set if the keyUsage extension is present.

The extended key usage extension places additional restrictions on the certificate uses. If this extension is present (whether critical or not) the key can only be used for the purposes specified.

A complete description of each test is given below. The comments about basicConstraints and keyUsage and V1 certificates above apply to **all** CA certificates.

SSL Client

The extended key usage extension must be absent or include the "web client authentication" OID. keyUsage must be absent or it must have the digitalSignature bit set. Netscape certificate type must be absent or it must have the SSL client bit set.

SSL Client CA

The extended key usage extension must be absent or include the "web client authentication" OID. Netscape certificate type must be absent or it must have the SSL CA bit set: this is used as a work around if the basicConstraints extension is absent.

SSL Server

The extended key usage extension must be absent or include the "web server authentication" and/or one of the SGC OIDs. keyUsage must be absent or it must have the digitalSignature, the keyEncipherment set or both bits set. Netscape certificate type must be absent or have the SSL server bit set.

SSL Server CA

The extended key usage extension must be absent or include the "web server authentication" and/or one of the SGC OIDs. Netscape certificate type must be absent or the SSL CA bit must be set: this is used as a work around if the basicConstraints extension is absent.

Netscape SSL Server

For Netscape SSL clients to connect to an SSL server it must have the keyEncipherment bit set if the keyUsage extension is present. This isn‘t always valid because some cipher suites use the key for digital signing. Otherwise it is the same as a normal SSL server.

Common S/MIME Client Tests

The extended key usage extension must be absent or include the "email protection" OID. Netscape certificate type must be absent or should have the S/MIME bit set. If the S/MIME bit is not set in netscape certificate type then the SSL client bit is tolerated as an alternative but a warning is shown: this is because some Verisign certificates don‘t set the S/MIME bit.

S/MIME Signing

In addition to the common S/MIME client tests the digitalSignature bit must be set if the keyUsage extension is present.

S/MIME Encryption

In addition to the common S/MIME tests the keyEncipherment bit must be set if the keyUsage extension is present.

S/MIME CA

The extended key usage extension must be absent or include the "email protection" OID. Netscape certificate type must be absent or must have the S/MIME CA bit set: this is used as a work around if the basicConstraints extension is absent.

CRL Signing

The keyUsage extension must be absent or it must have the CRL signing bit set.

CRL Signing CA

The normal CA tests apply. Except in this case the basicConstraints extension must be present.

BUGS

Extensions in certificates are not transferred to certificate requests and vice versa.

It is possible to produce invalid certificates or requests by specifying the wrong private key or using inconsistent options in some cases: these should be checked.

There should be options to explicitly set such things as start and end dates rather than an offset from the current time.

The code to implement the verify behaviour described in the **TRUST SETTINGS** is currently being developed. It thus describes the intended behaviour rather than the current behaviour. It is hoped that it will represent reality in OpenSSL 0.9.5 and later.

SEE ALSO

req(1)|req(1), *ca(1)|ca(1)*, *genrsa(1)|genrsa(1)*, *gendsa(1)|gendsa(1)*, *verify(1)|verify(1)*