NAME

certtool - GnuTLS certificate tool

SYNOPSIS

certtool [**-flags**] [**-flag** [*value*]] [**--option-name**[[=|]*value*]]

All arguments must be options.

DESCRIPTION

Tool to parse and generate X.509 certificates, requests and private keys. It can be used interactively or non interactively by specifying the template command line option.

The tool accepts files or supported URIs via the --infile option. In case PIN is required for URI access you can provide it using the environment variables GNUTLS_PIN and GNUTLS_SO_PIN.

OPTIONS

-d *num*, **--debug**=*num*

Enable debugging. This option takes an integer number as its argument. The value of *num* is constrained to being:

in the range 0 through 9999

Specifies the debug level.

-V, **--verbose**

More verbose output.

--infile=*file*

Input file.

--outfile=*str*

Output file.

Certificate related options

-i, **--certificate-info**

Print information on the given certificate.

--pubkey-info
Print information on a public key.

The option combined with --load-request, --load-pubkey, --load-privkey and --load-certificate will extract the public key of the object in question.

-s, **--generate-self-signed**
Generate a self-signed certificate.

-c, **--generate-certificate**
Generate a signed certificate.

--generate-proxy
Generates a proxy certificate.

-u, **--update-certificate**
Update a signed certificate.

--fingerprint
Print the fingerprint of the given certificate.

This is a simple hash of the DER encoding of the certificate. It can be combined with the --hash parameter. However, it is recommended for identification to use the key-id which depends only on the certificate’s key.

--key-id
Print the key ID of the given certificate.

This is a hash of the public key of the given certificate. It identifies the key uniquely, remains the same on a certificate renewal and depends only on signed fields of the certificate.

--certificate-pubkey
Print certificate’s public key.

This option is deprecated as a duplicate of --pubkey-info

NOTE: THIS OPTION IS DEPRECATED

--v1
Generate an X.509 version 1 certificate (with no extensions).

--sign-params=*str*
Sign a certificate with a specific signature algorithm.

This option can be combined with --generate-certificate, to sign the certificate with a specific signature algorithm variant. The only option supported is 'RSA-PSS', and should be specified when the signer does not have a certificate which is marked for RSA-PSS use only.

Certificate request related options

--crq-info
Print information on the given certificate request.

-q, **--generate-request**
Generate a PKCS #10 certificate request. This option must not appear in combination with any of the following options: infile.

Will generate a PKCS #10 certificate request. To specify a private key use --load-privkey.

--no-crq-extensions
Do not use extensions in certificate requests.

PKCS#12 file related options

--p12-info
Print information on a PKCS #12 structure.

This option will dump the contents and print the metadata of the provided PKCS #12 structure.

--p12-name=*str*
The PKCS #12 friendly name to use.

The name to be used for the primary certificate and private key in a PKCS #12 file.

--to-p12

Generate a PKCS #12 structure.

It requires a certificate, a private key and possibly a CA certificate to be specified.

Private key related options

-k, **--key-info**
Print information on a private key.

--p8-info
Print information on a PKCS #8 structure.

This option will print information about encrypted PKCS #8 structures. That option does not require the decryption of the structure.

--to-rsa
Convert an RSA-PSS key to raw RSA format.

It requires an RSA-PSS key as input and will output a raw RSA key. This command is necessary for compatibility with applications that cannot read RSA-PSS keys.

-p, **--generate-privkey**
Generate a private key.

When generating RSA-PSS private keys, the --hash option will restrict the allowed hash for the key; in the same keys the --salt-size option is also acceptable.

--key-type=*str*
Specify the key type to use on key generation.

This option can be combined with --generate-privkey, to specify the key type to be generated. Valid options are, 'rsa', 'rsa-pss', 'dsa', 'ecdsa', 'ed25519, 'ed448', 'x25519', and 'x448'.'. When combined with certificate generation it can be used to specify an RSA-PSS certificate when an RSA key is given.

--bits=*num*
Specify the number of bits for key generation. This option takes an integer number as its argument.

--curve=*str*
Specify the curve used for EC key generation.

Supported values are secp192r1, secp224r1, secp256r1, secp384r1 and secp521r1.

--sec-param=*security parameter*
Specify the security level [low, legacy, medium, high, ultra].

This is alternative to the bits option.

--to-p8
Convert a given key to a PKCS #8 structure.

This needs to be combined with --load-privkey.

-8, **--pkcs8**
Use PKCS #8 format for private keys.

--provable
Generate a private key or parameters from a seed using a provable method.

This will use the FIPS PUB186-4 algorithms (i.e., Shawe-Taylor) for provable key generation. When specified the private keys or parameters will be generated from a seed, and can be later validated with --verify-provable-privkey to be correctly generated from the seed. You may specify --seed or allow GnuTLS to generate one (recommended). This option can be combined with --generate-privkey or --generate-dh-params.

That option applies to RSA and DSA keys. On the DSA keys the PQG parameters are generated using the seed, and on RSA the two primes.

--verify-provable-privkey
Verify a private key generated from a seed using a provable method.

This will use the FIPS-186-4 algorithms for provable key generation. You may specify --seed or use the seed stored in the private key structure.

--seed=*str*
When generating a private key use the given hex-encoded seed.

The seed acts as a security parameter for the private key, and thus a seed size which corresponds to the security level of the private key should be provided (e.g., 256-bits seed).

CRL related options

-l, **--crl-info**

Print information on the given CRL structure.

--generate-crl

Generate a CRL.

This option generates a Certificate Revocation List. When combined with --load-crl it would use the loaded CRL as base for the generated (i.e., all revoked certificates in the base will be copied to the new CRL). To add new certificates to the CRL use --load-certificate.

--verify-crl

Verify a Certificate Revocation List using a trusted list. This option must appear in combination with the following options: load-ca-certificate.

The trusted certificate list must be loaded with --load-ca-certificate.

Certificate verification related options

-e, **--verify-chain**

Verify a PEM encoded certificate chain.

Verifies the validity of a certificate chain. That is, an ordered set of certificates where each one is the issuer of the previous, and the first is the end-certificate to be validated. In a proper chain the last certificate is a self signed one. It can be combined with --verify-purpose or --verify-hostname.

--verify

Verify a PEM encoded certificate (chain) against a trusted set.

The trusted certificate list can be loaded with --load-ca-certificate. If no certificate list is provided, then the system’s trusted certificate list is used. Note that during verification multiple paths may be explored. On a successful verification the successful path will be the last one. It can be combined with --verify-purpose or --verify-hostname.

--verify-hostname=*str*

Specify a hostname to be used for certificate chain verification.

This is to be combined with one of the verify certificate options.

--verify-email=*str*
Specify a email to be used for certificate chain verification. This option must not appear in combination with any of the following options: verify-hostname.

This is to be combined with one of the verify certificate options.

--verify-purpose=*str*
Specify a purpose OID to be used for certificate chain verification.

This object identifier restricts the purpose of the certificates to be verified. Example purposes are 1.3.6.1.5.5.7.3.1 (TLS WWW), 1.3.6.1.5.5.7.3.4 (EMAIL) etc. Note that a CA certificate without a purpose set (extended key usage) is valid for any purpose.

--verify-allow-broken
Allow broken algorithms, such as MD5 for verification.

This can be combined with --p7-verify, --verify or --verify-chain.

--verify-profile=*str*
Specify a security level profile to be used for verification.

This option can be used to specify a certificate verification profile. Certificate verification profiles correspond to the security level. This should be one of 'none', 'very weak', 'low', 'legacy', 'medium', 'high', 'ultra', 'future'. Note that by default no profile is applied, unless one is set as minimum in the gnutls configuration file.

PKCS#7 structure options

--p7-generate
Generate a PKCS #7 structure.

This option generates a PKCS #7 certificate container structure. To add certificates in the structure use --load-certificate and --load-crl.

--p7-sign
Signs using a PKCS #7 structure.

This option generates a PKCS #7 structure containing a signature for the provided data from infile.

The data are stored within the structure. The signer certificate has to be specified using --load-certificate and --load-privkey. The input to --load-certificate can be a list of certificates. In case of a list, the first certificate is used for signing and the other certificates are included in the structure.

--p7-detached-sign

Signs using a detached PKCS #7 structure.

This option generates a PKCS #7 structure containing a signature for the provided data from infile. The signer certificate has to be specified using --load-certificate and --load-privkey. The input to --load-certificate can be a list of certificates. In case of a list, the first certificate is used for signing and the other certificates are included in the structure.

--p7-include-cert, **--no-p7-include-cert**

The signer's certificate will be included in the cert list. The *no-p7-include-cert* form will disable the option. This option is enabled by default.

This options works with --p7-sign or --p7-detached-sign and will include or exclude the signer's certificate into the generated signature.

--p7-time, **--no-p7-time**

Will include a timestamp in the PKCS #7 structure. The *no-p7-time* form will disable the option.

This option will include a timestamp in the generated signature

--p7-show-data, **--no-p7-show-data**

Will show the embedded data in the PKCS #7 structure. The *no-p7-show-data* form will disable the option.

This option can be combined with --p7-verify or --p7-info and will display the embedded signed data in the PKCS #7 structure.

--p7-info

Print information on a PKCS #7 structure.

--p7-verify

Verify the provided PKCS #7 structure.

This option verifies the signed PKCS #7 structure. The certificate list to use for verification can be

specified with --load-ca-certificate. When no certificate list is provided, then the system's certificate list is used. Alternatively a direct signer can be provided using --load-certificate. A key purpose can be enforced with the --verify-purpose option, and the --load-data option will utilize detached data.

--smime-to-p7
Convert S/MIME to PKCS #7 structure.

Other options

--generate-dh-params
Generate PKCS #3 encoded Diffie-Hellman parameters.

The will generate random parameters to be used with Diffie-Hellman key exchange. The output parameters will be in PKCS #3 format. Note that it is recommended to use the --get-dh-params option instead.

NOTE: THIS OPTION IS DEPRECATED

--get-dh-params
List the included PKCS #3 encoded Diffie-Hellman parameters.

Returns stored DH parameters in GnuTLS. Those parameters returned are defined in RFC7919, and can be considered standard parameters for a TLS key exchange. This option is provided for old applications which require DH parameters to be specified; modern GnuTLS applications should not require them.

--dh-info
Print information PKCS #3 encoded Diffie-Hellman parameters.

--load-privkey=*str*
Loads a private key file.

This can be either a file or a PKCS #11 URL

--load-pubkey=*str*
Loads a public key file.

This can be either a file or a PKCS #11 URL

--load-request=*str*
Loads a certificate request file.

This option can be used with a file

--load-certificate=*str*
Loads a certificate file.

This option can be used with a file

--load-ca-privkey=*str*
Loads the certificate authority’s private key file.

This can be either a file or a PKCS #11 URL

--load-ca-certificate=*str*
Loads the certificate authority’s certificate file.

This can be either a file or a PKCS #11 URL

--load-crl=*str*
Loads the provided CRL.

This option can be used with a file

--load-data=*str*
Loads auxiliary data.

This option can be used with a file

--password=*str*
Password to use.

You can use this option to specify the password in the command line instead of reading it from the tty. Note, that the command line arguments are available for view in others in the system. Specifying password as ’’ is the same as specifying no password.

--null-password
Enforce a NULL password.

This option enforces a NULL password. This is different than the empty or no password in schemas like PKCS #8.

--empty-password
Enforce an empty password.

This option enforces an empty password. This is different than the NULL or no password in schemas like PKCS #8.

--hex-numbers
Print big number in an easier format to parse.

--cprint
In certain operations it prints the information in C-friendly format.

In certain operations it prints the information in C-friendly format, suitable for including into C programs.

--rsa
Generate RSA key.

When combined with --generate-privkey generates an RSA private key.

NOTE: THIS OPTION IS DEPRECATED

--dsa
Generate DSA key.

When combined with --generate-privkey generates a DSA private key.

NOTE: THIS OPTION IS DEPRECATED

--ecc
Generate ECC (ECDSA) key.

When combined with --generate-privkey generates an elliptic curve private key to be used with ECDSA.

NOTE: THIS OPTION IS DEPRECATED

--ecdsa
This is an alias for the --*ecc* option.

NOTE: THIS OPTION IS DEPRECATED

--hash=*str*
Hash algorithm to use for signing.

Available hash functions are SHA1, RMD160, SHA256, SHA384, SHA512, SHA3-224, SHA3-256, SHA3-384, SHA3-512.

--salt-size=*num*
Specify the RSA-PSS key default salt size. This option takes an integer number as its argument.

Typical keys shouldn’t set or restrict this option.

--inder, **--no-inder**
Use DER format for input certificates, private keys, and DH parameters . The *no-inder* form will disable the option.

The input files will be assumed to be in DER or RAW format. Unlike options that in PEM input would allow multiple input data (e.g. multiple certificates), when reading in DER format a single data structure is read.

--inraw
This is an alias for the --*inder* option.

--outder, **--no-outder**
Use DER format for output certificates, private keys, and DH parameters. The *no-outder* form will disable the option.

The output will be in DER or RAW format.

--outraw
This is an alias for the --*outder* option.

--disable-quick-random
No effect.

NOTE: THIS OPTION IS DEPRECATED

--template=*str*
Template file to use for non-interactive operation.

--stdout-info
Print information to stdout instead of stderr.

--ask-pass
Enable interaction for entering password when in batch mode.

This option will enable interaction to enter password when in batch mode. That is useful when the template option has been specified.

--pkcs-cipher=*cipher*
Cipher to use for PKCS #8 and #12 operations.

Cipher may be one of 3des, 3des-pkcs12, aes-128, aes-192, aes-256, rc2-40, arcfour.

--provider=*str*
Specify the PKCS #11 provider library.

This will override the default options in /usr/local/etc/gnutls/pkcs11.conf

--text, **--no-text**
Output textual information before PEM-encoded certificates, private keys, etc. The *no-text* form will disable the option. This option is enabled by default.

Output textual information before PEM-encoded data

-v *arg*, **--version**=*arg*
Output version of program and exit. The default mode is ‘v’, a simple version. The ‘c’ mode will print copyright information and ‘n’ will print the full copyright notice.

-h, **--help**
Display usage information and exit.

-!, **--more-help**

Pass the extended usage information through a pager.

FILES

Certtool's template file format

A template file can be used to avoid the interactive questions of certtool. Initially create a file named 'cert.cfg' that contains the information about the certificate. The template can be used as below:

```
$ certtool --generate-certificate --load-privkey key.pem     --template cert.cfg --outfile cert.pem    --load-ca-certific
```

An example certtool template file that can be used to generate a certificate request or a self signed certificate follows.

```
# X.509 Certificate options
#
# DN options

# The organization of the subject.
organization = "Koko inc."

# The organizational unit of the subject.
unit = "sleeping dept."

# The locality of the subject.
# locality =

# The state of the certificate owner.
state = "Attiki"

# The country of the subject. Two letter code.
country = GR

# The common name of the certificate owner.
cn = "Cindy Lauper"

# A user id of the certificate owner.
#uid = "clauper"

# Set domain components
#dc = "name"
```

```
#dc = "domain"

# If the supported DN OIDs are not adequate you can set
# any OID here.
# For example set the X.520 Title and the X.520 Pseudonym
# by using OID and string pairs.
#dn_oid = "2.5.4.12 Dr."
#dn_oid = "2.5.4.65 jackal"

# This is deprecated and should not be used in new
# certificates.
# pkcs9_email = "none@none.org"

# An alternative way to set the certificate's distinguished name directly
# is with the "dn" option. The attribute names allowed are:
# C (country), street, O (organization), OU (unit), title, CN (common name),
# L (locality), ST (state), placeOfBirth, gender, countryOfCitizenship,
# countryOfResidence, serialNumber, telephoneNumber, surName, initials,
# generationQualifier, givenName, pseudonym, dnQualifier, postalCode, name,
# businessCategory, DC, UID, jurisdictionOfIncorporationLocalityName,
# jurisdictionOfIncorporationStateOrProvinceName,
# jurisdictionOfIncorporationCountryName, XmppAddr, and numeric OIDs.

#dn = "cn = Nikos,st = New Something,C=GR,surName=Mavrogiannopoulos,2.5.4.9=Arkadias"

# The serial number of the certificate
# The value is in decimal (i.e. 1963) or hex (i.e. 0x07ab).
# Comment the field for a random serial number.
serial = 007

# In how many days, counting from today, this certificate will expire.
# Use -1 if there is no expiration date.
expiration_days = 700

# Alternatively you may set concrete dates and time. The GNU date string
# formats are accepted. See:
# https://www.gnu.org/software/tar/manual/html_node/Date-input-formats.html

#activation_date = "2004-02-29 16:21:42"
#expiration_date = "2025-02-29 16:24:41"
```

```
# X.509 v3 extensions

# A dnsname in case of a WWW server.
#dns_name = "www.none.org"
#dns_name = "www.morethanone.org"

# An othername defined by an OID and a hex encoded string
#other_name = "1.3.6.1.5.2.2 302ca00d1b0b56414e5245494e2e4f5247a11b3019a006020400000002a10f300d1b0
#other_name_utf8 = "1.2.4.5.6 A UTF8 string"
#other_name_octet = "1.2.4.5.6 A string that will be encoded as ASN.1 octet string"

# Allows writing an XmppAddr Identifier
#xmpp_name = juliet@im.example.com

# Names used in PKINIT
#krb5_principal = user@REALM.COM
#krb5_principal = HTTP/user@REALM.COM

# A subject alternative name URI
#uri = "https://www.example.com"

# An IP address in case of a server.
#ip_address = "192.168.1.1"

# An email in case of a person
email = "none@none.org"

# TLS feature (rfc7633) extension. That can is used to indicate mandatory TLS
# extension features to be provided by the server. In practice this is used
# to require the Status Request (extid: 5) extension from the server. That is,
# to require the server holding this certificate to provide a stapled OCSP response.
# You can have multiple lines for multiple TLS features.

# To ask for OCSP status request use:
#tls_feature = 5

# Challenge password used in certificate requests
challenge_password = 123456

# Password when encrypting a private key
```

```
#password = secret

# An URL that has CRLs (certificate revocation lists)
# available. Needed in CA certificates.
#crl_dist_points = "https://www.getcrl.crl/getcrl/"

# Whether this is a CA certificate or not
#ca

# Subject Unique ID (in hex)
#subject_unique_id = 00153224

# Issuer Unique ID (in hex)
#issuer_unique_id = 00153225

#### Key usage

# The following key usage flags are used by CAs and end certificates

# Whether this certificate will be used to sign data (needed
# in TLS DHE ciphersuites). This is the digitalSignature flag
# in RFC5280 terminology.
signing_key

# Whether this certificate will be used to encrypt data (needed
# in TLS RSA ciphersuites). Note that it is preferred to use different
# keys for encryption and signing. This is the keyEncipherment flag
# in RFC5280 terminology.
encryption_key

# Whether this key will be used to sign other certificates. The
# keyCertSign flag in RFC5280 terminology.
#cert_signing_key

# Whether this key will be used to sign CRLs. The
# cRLSign flag in RFC5280 terminology.
#crl_signing_key

# The keyAgreement flag of RFC5280. Its purpose is loosely
# defined. Not use it unless required by a protocol.
```

```
#key_agreement

# The dataEncipherment flag of RFC5280. Its purpose is loosely
# defined. Not use it unless required by a protocol.
#data_encipherment

# The nonRepudiation flag of RFC5280. Its purpose is loosely
# defined. Not use it unless required by a protocol.
#non_repudiation

#### Extended key usage (key purposes)

# The following extensions are used in an end certificate
# to clarify its purpose. Some CAs also use it to indicate
# the types of certificates they are purposed to sign.


# Whether this certificate will be used for a TLS client;
# this sets the id-kp-clientAuth (1.3.6.1.5.5.7.3.2) of
# extended key usage.
#tls_www_client

# Whether this certificate will be used for a TLS server;
# this sets the id-kp-serverAuth (1.3.6.1.5.5.7.3.1) of
# extended key usage.
#tls_www_server

# Whether this key will be used to sign code. This sets the
# id-kp-codeSigning (1.3.6.1.5.5.7.3.3) of extended key usage
# extension.
#code_signing_key

# Whether this key will be used to sign OCSP data. This sets the
# id-kp-OCSPSigning (1.3.6.1.5.5.7.3.9) of extended key usage extension.
#ocsp_signing_key

# Whether this key will be used for time stamping. This sets the
# id-kp-timeStamping (1.3.6.1.5.5.7.3.8) of extended key usage extension.
#time_stamping_key
```

```
# Whether this key will be used for email protection. This sets the
# id-kp-emailProtection (1.3.6.1.5.5.7.3.4) of extended key usage extension.
#email_protection_key

# Whether this key will be used for IPsec IKE operations (1.3.6.1.5.5.7.3.17).
#ipsec_ike_key

## adding custom key purpose OIDs

# for microsoft smart card logon
# key_purpose_oid = 1.3.6.1.4.1.311.20.2.2

# for email protection
# key_purpose_oid = 1.3.6.1.5.5.7.3.4

# for any purpose (must not be used in intermediate CA certificates)
# key_purpose_oid = 2.5.29.37.0

### end of key purpose OIDs

### Adding arbitrary extensions
# This requires to provide the extension OIDs, as well as the extension data in
# hex format. The following two options are available since GnuTLS 3.5.3.
#add_extension = "1.2.3.4 0x0AAB01ACFE"

# As above but encode the data as an octet string
#add_extension = "1.2.3.4 octet_string(0x0AAB01ACFE)"

# For portability critical extensions shouldn't be set to certificates.
#add_critical_extension = "5.6.7.8 0x1AAB01ACFE"

# When generating a certificate from a certificate
# request, then honor the extensions stored in the request
# and store them in the real certificate.
#honor_crq_extensions

# Alternatively only specific extensions can be copied.
#honor_crq_ext = 2.5.29.17
#honor_crq_ext = 2.5.29.15
```

```
# Path length constraint. Sets the maximum number of
# certificates that can be used to certify this certificate.
# (i.e. the certificate chain length)
#path_len = -1
#path_len = 2

# OCSP URI
# ocsp_uri = https://my.ocsp.server/ocsp

# CA issuers URI
# ca_issuers_uri = https://my.ca.issuer

# Certificate policies
#policy1 = 1.3.6.1.4.1.5484.1.10.99.1.0
#policy1_txt = "This is a long policy to summarize"
#policy1_url = https://www.example.com/a-policy-to-read

#policy2 = 1.3.6.1.4.1.5484.1.10.99.1.1
#policy2_txt = "This is a short policy"
#policy2_url = https://www.example.com/another-policy-to-read

# The number of additional certificates that may appear in a
# path before the anyPolicy is no longer acceptable.
#inhibit_anypolicy_skip_certs 1

# Name constraints

# DNS
#nc_permit_dns = example.com
#nc_exclude_dns = test.example.com

# EMAIL
#nc_permit_email = "nmav@ex.net"

# Exclude subdomains of example.com
#nc_exclude_email = .example.com

# Exclude all e-mail addresses of example.com
#nc_exclude_email = example.com
```

```
# IP
#nc_permit_ip = 192.168.0.0/16
#nc_exclude_ip = 192.168.5.0/24
#nc_permit_ip = fc0a:eef2:e7e7:a56e::/64


# Options for proxy certificates
#proxy_policy_language = 1.3.6.1.5.5.7.21.1


# Options for generating a CRL

# The number of days the next CRL update will be due.
# next CRL update will be in 43 days
#crl_next_update = 43

# this is the 5th CRL by this CA
# The value is in decimal (i.e. 1963) or hex (i.e. 0x07ab).
# Comment the field for a time-based number.
# Time-based CRL numbers generated in GnuTLS 3.6.3 and later
# are significantly larger than those generated in previous
# versions. Since CRL numbers need to be monotonic, you need
# to specify the CRL number here manually if you intend to
# downgrade to an earlier version than 3.6.3 after publishing
# the CRL as it is not possible to specify CRL numbers greater
# than 2**63-2 using hex notation in those versions.
#crl_number = 5

# Specify the update dates more precisely.
#crl_this_update_date = "2004-02-29 16:21:42"
#crl_next_update_date = "2025-02-29 16:24:41"

# The date that the certificates will be made seen as
# being revoked.
#crl_revocation_date = "2025-02-29 16:24:41"
```

EXAMPLES

Generating private keys

To create an RSA private key, run:

```
$ certtool --generate-privkey --outfile key.pem --rsa
```

To create a DSA or elliptic curves (ECDSA) private key use the above command combined with 'dsa' or 'ecc' options.

Generating certificate requests

To create a certificate request (needed when the certificate is issued by another party), run:

```
certtool --generate-request --load-privkey key.pem --outfile request.pem
```

If the private key is stored in a smart card you can generate a request by specifying the private key object URL.

```
$ ./certtool --generate-request --load-privkey "pkcs11:..." --load-pubkey "pkcs11:..." --outfile request.pem
```

Generating a self-signed certificate

To create a self signed certificate, use the command:

```
$ certtool --generate-privkey --outfile ca-key.pem
$ certtool --generate-self-signed --load-privkey ca-key.pem --outfile ca-cert.pem
```

Note that a self-signed certificate usually belongs to a certificate authority, that signs other certificates.

Generating a certificate

To generate a certificate using the previous request, use the command:

```
$ certtool --generate-certificate --load-request request.pem --outfile cert.pem --load-ca-certificate ca-cert.pem -
```

To generate a certificate using the private key only, use the command:

```
$ certtool --generate-certificate --load-privkey key.pem --outfile cert.pem --load-ca-certificate ca-cert.pem --lo
```

Certificate information

To view the certificate information, use:

```
$ certtool --certificate-info --infile cert.pem
```

Changing the certificate format

To convert the certificate from PEM to DER format, use:

```
$ certtool --certificate-info --infile cert.pem --outder --outfile cert.der
```

PKCS #12 structure generation

To generate a PKCS #12 structure using the previous key and certificate, use the command:

```
$ certtool --load-certificate cert.pem --load-privkey key.pem --to-p12 --outder --outfile key.p12
```

Some tools (reportedly web browsers) have problems with that file because it does not contain the CA certificate for the certificate. To work around that problem in the tool, you can use the --load-ca-certificate parameter as follows:

```
$ certtool --load-ca-certificate ca.pem   --load-certificate cert.pem --load-privkey key.pem   --to-p12 --outder --out
```

Obtaining Diffie-Hellman parameters

To obtain the RFC7919 parameters for Diffie-Hellman key exchange, use the command:

```
$ certtool --get-dh-params --outfile dh.pem --sec-param medium
```

Verifying a certificate

To verify a certificate in a file against the system's CA trust store use the following command:

```
$ certtool --verify --infile cert.pem
```

It is also possible to simulate hostname verification with the following options:

```
$ certtool --verify --verify-hostname www.example.com --infile cert.pem
```

Proxy certificate generation

Proxy certificate can be used to delegate your credential to a temporary, typically short-lived, certificate. To create one from the previously created certificate, first create a temporary key and then generate a proxy certificate for it, using the commands:

```
$ certtool --generate-privkey > proxy-key.pem
$ certtool --generate-proxy --load-ca-privkey key.pem   --load-privkey proxy-key.pem --load-certificate cert.pem
```

Certificate revocation list generation

To create an empty Certificate Revocation List (CRL) do:

```
$ certtool --generate-crl --load-ca-privkey x509-ca-key.pem          --load-ca-certificate x509-ca.pem
```

To create a CRL that contains some revoked certificates, place the certificates in a file and use **--load-certificate** as follows:

```
$ certtool --generate-crl --load-ca-privkey x509-ca-key.pem   --load-ca-certificate x509-ca.pem --load-certificate r
```

To verify a Certificate Revocation List (CRL) do:

```
$ certtool --verify-crl --load-ca-certificate x509-ca.pem < crl.pem
```

EXIT STATUS

One of the following exit values will be returned:

0 (EXIT_SUCCESS)
: Successful program execution.

1 (EXIT_FAILURE)
: The operation failed or the command syntax was not valid.

SEE ALSO

p11tool (1), psktool (1), srptool (1)

AUTHORS

COPYRIGHT

BUGS

Please send bug reports to: bugs@gnutls.org