

NAME

IO::Socket::SSL::Utils -- loading, storing, creating certificates and keys

SYNOPSIS

```

use IO::Socket::SSL::Utils;
my $cert = PEM_file2cert('cert.pem'); # load certificate from file
my $string = PEM_cert2string($cert); # convert certificate to PEM string
CERT_free($cert); # free memory within OpenSSL

my $key = KEY_create_rsa(2048); # create new 2048-bit RSA key
PEM_string2file($key,"key.pem"); # and write it to file
KEY_free($key); # free memory within OpenSSL

```

DESCRIPTION

This module provides various utility functions to work with certificates and private keys, shielding some of the complexity of the underlying [Net::SSLeay](#) and OpenSSL.

FUNCTIONS

- Functions converting between string or file and certificates and keys. They croak if the operation cannot be completed.

```

PEM_file2cert(file) -> cert
PEM_cert2file(cert,file)
PEM_string2cert(string) -> cert
PEM_cert2string(cert) -> string
PEM_file2key(file) -> key
PEM_key2file(key,file)
PEM_string2key(string) -> key
PEM_key2string(key) -> string

```

- Functions for cleaning up. Each loaded or created cert and key must be freed to not leak memory.

```

CERT_free(cert)
KEY_free(key)

```

- KEY_create_rsa(bits) -> key

Creates an RSA key pair, bits defaults to 2048.

- CERT_asHash(cert,[digest_algo]) -> hash

Extracts the information from the certificate into a hash and uses the given digest_algo (default: SHA-256) to determine digest of pubkey and cert. The resulting hash contains:

subject Hash with the parts of the subject, e.g. commonName, countryName, organizationName, stateOrProvinceName, localityName.

subjectAltNames

Array with list of alternative names. Each entry in the list is of [type,value], where type can be OTHERNAME, EMAIL, DNS, X400, DIRNAME, EDIPARTY, URI, IP or RID.

issuer Hash with the parts of the issuer, e.g. commonName, countryName, organizationName, stateOrProvinceName, localityName.

not_before, not_after

The time frame, where the certificate is valid, as time_t, e.g. can be converted with localtime or similar functions.

serial The serial number

crl_uri List of URIs for CRL distribution.

- `ocsp_uri`
List of URIs for revocation checking using OCSP.
- `keyusage`
List of keyUsage information in the certificate.
- `extkeyusage`
List of extended key usage information from the certificate. Each entry in this list consists of a hash with `oid`, `nid`, `ln` and `sn`.
- `pubkey_digest_xxx`
Binary digest of the pubkey using the given digest algorithm, e.g. `pubkey_digest_sha256` if (the default) SHA-256 was used.
- `x509_digest_xxx`
Binary digest of the X.509 certificate using the given digest algorithm, e.g. `x509_digest_sha256` if (the default) SHA-256 was used.
- `fingerprint_xxx`
Fingerprint of the certificate using the given digest algorithm, e.g. `fingerprint_sha256` if (the default) SHA-256 was used. Contrary to `digest_*` this is an ASCII string with a list of hexadecimal numbers, e.g. `"73:59:75:5C:6D..."`.
- `ext`
List of extensions. Each entry in the list is a hash with `oid`, `nid`, `sn`, `critical` (boolean) and `data` (string representation given by `X509V3_EXT_print`).
- `version`
Certificate version, usually 2 (x509v3)
- `CERT_create(hash) -> (cert,key)`
Creates a certificate based on the given hash. If the issuer is not specified the certificate will be self-signed. The following keys can be given:

`subject` Hash with the parts of the subject, e.g. `commonName`, `countryName`, ... as described in `CERT_asHash`. Default points to `IO::Socket::SSL`.

`not_before`
A `time_t` value when the certificate starts to be valid. Defaults to current time.

`not_after`
A `time_t` value when the certificate ends to be valid. Defaults to current time plus one 365 days.

`serial`
The serial number. If not given a random number will be used.

`version`
The version of the certificate, default 2 (x509v3).

`CA true|false`
If true declare certificate as CA, defaults to false.

`purpose string|array|hash`
Set the purpose of the certificate. The different purposes can be given as a string separated by non-word character, as array or hash. With string or array each purpose can be prefixed with `'+'` (enable) or `'-'` (disable) and same can be done with the value when given as a hash. By default enabling the purpose is assumed.

If the CA option is given and true the defaults `"ca,sslca,emailca,objca"` are assumed, but can be overridden with explicit purpose. If the CA option is given and false the defaults `"server,client"` are assumed. If no CA option and no purpose is given it defaults to `"server,client"`.

Purpose affects `basicConstraints`, `keyUsage`, `extKeyUsage` and `netscapeCertType`. The following purposes are defined (case is not important):

```

client
server
email
objsign

```

```

CA
sslCA
emailCA
objCA

```

```

emailProtection
codeSigning
timeStamping

```

```

digitalSignature
nonRepudiation
keyEncipherment
dataEncipherment
keyAgreement
keyCertSign
cRLSign
encipherOnly
decipherOnly

```

Examples:

```

# root-CA for SSL certificates
purpose => 'sslCA' # or CA => 1

# server certificate and CA (typically self-signed)
purpose => 'sslCA,server'

# client certificate
purpose => 'client',

```

```
ext [{ sn => .., data => ... }, ... ]
```

List of extensions. The type of the extension can be specified as name with `sn` or as NID with `nid` and the data with `data`. These data must be in the same syntax as expected within `openssl.cnf`, e.g. something like `OCSP;URI=http://....`. Additionally the critical flag can be set with `critical = 1`.

`key key` use given key as key for certificate, otherwise a new one will be generated and returned

`issuer_cert cert`
set issuer for new certificate

`issuer_key key`
sign new certificate with given key

`issuer [cert, key]`
Instead of giving `issuer_key` and `issuer_cert` as separate arguments they can be given both together.

`digest algorithm`
specify the algorithm used to sign the certificate, default SHA-256.

AUTHOR

Steffen Ullrich