OpenSSL

Introduction

This module uses the functions of » OpenSSL for generation and verification of signatures and for sealing (encrypting) and opening (decrypting) data. OpenSSL offers many features that this module currently doesn't support. Some of these may be added in the future.

Installing/Configuring

Requirements

In order to use the OpenSSL functions you need to install the » OpenSSL package. PHP between versions 4.0.5 and 4.3.1 will work with OpenSSL >= 0.9.5. Other versions (PHP <=4.0.4 and >= 4.3.2) require OpenSSL >= 0.9.6.

Warning

You are strongly encouraged to use the most recent OpenSSL version, otherwise your web server could be vulnerable to attack.

Installation

To use PHP's OpenSSL support you must also compile PHP *--with-openssl[=DIR]*.

Note

Note to Win32 Users

In order for this extension to work, there are DLL files that must be available to the Windows system *PATH*. For information on how to do this, see the FAQ entitled " How do I add my PHP directory to the PATH on Windows ". Although copying DLL files from the PHP folder into the Windows system directory also works (because the system directory is by default in the system's *PATH*), this is not recommended. *This extension requires the following files to be in the PATH: libeay32.dll*

Additionally, if you are planning to use the key generation and certificate signing functions, you will need to install a valid *openssl.cnf* file on your system. As of PHP 4.3.0, we include a sample configuration file in our win32 binary distributions. PHP 4.3.x and 4.4.x has the file in the *openssl* directory. PHP 5.x and 6.x has the file in the *extras/openssl* directory. If you are either using PHP 4.2.x or missing the file, you can obtain it from » the OpenSSL binaries page or by downloading a recent PHP release. Be aware that Windows Explorer hides the*.cnf* extension by default and says the file Type is *SpeedDial*.

PHP will search for the *openssl.cnf* using the following logic:

- the *OPENSSL_CONF* environmental variable, if set, will be used as the path (including filename) of the configuration file.
- the *SSLEAY_CONF* environmental variable, if set, will be used as the path (including filename) of the configuration file.

- The file *openssl.cnf* will be assumed to be found in the default certificate area, as configured at the time that the openssl DLL was compiled. This is usually means that the default filename is *c:\usr\local\ssl\openssl.cnf*.

In your installation, you need to decide whether to install the configuration file at *c:\usr\local\ssl\openssl.cnf* or whether to install it someplace else and use environmental variables (possibly on a per-virtual-host basis) to locate the configuration file. Note that it is possible to override the default path from the script using the `configargs` of the functions that require a configuration file.

Runtime Configuration

This extension has no configuration directives defined in *php.ini*.

Resource Types

This extension has no resource types defined.

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

Purpose checking flags

X509_PURPOSE_SSL_CLIENT (integer)

X509_PURPOSE_SSL_SERVER (integer)

X509_PURPOSE_NS_SSL_SERVER (integer)

X509_PURPOSE_SMIME_SIGN (integer)

X509_PURPOSE_SMIME_ENCRYPT (integer)

X509_PURPOSE_CRL_SIGN (integer)

X509_PURPOSE_ANY (integer)

Padding flags

OPENSSL_PKCS1_PADDING (integer)

OPENSSL_SSLV23_PADDING (integer)

OPENSSL_NO_PADDING (integer)

OPENSSL_PKCS1_OAEP_PADDING (integer)

Key types

OPENSSL_KEYTYPE_RSA (integer)

OPENSSL_KEYTYPE_DSA (integer)

OPENSSL_KEYTYPE_DH (integer)

PKCS7 Flags/Constants

The S/MIME functions make use of flags which are specified using a bitfield which can include one or more of the following values:

PKCS7 CONSTANTS

Constant	Description
PKCS7_TEXT	Adds text/plain content type headers to encrypted/signed message. If decrypting or verifying, it strips those headers from the output - if the decrypted or verified message is not of MIME type text/plain then an error will occur.
PKCS7_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 options is present, no translation occurs. This is useful when handling binary data which may not be in MIME format.
PKCS7_NOINTERN	When verifying a message, certificates (if any) included in the message are normally searched for the signing certificate. With this option only the certificates specified in the *extracerts* parameter of openssl_pkcs7_verify() are used. The supplied certificates can still be used as untrusted CAs however.
PKCS7_NOVERIFY	Do not verify the signers certificate of a signed message.
PKCS7_NOCHAIN	Do not chain verification of signers certificates: that is don't use the certificates in the signed message as untrusted CAs.
PKCS7_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 *extracerts* to openssl_pkcs7_verify() for example).
PKCS7_NOATTR	Normally when a message is signed, a set of attributes are included which include the signing time and the supported symmetric algorithms. With this option they are not included.
PKCS7_DETACHED	When signing a message, use cleartext signing with the MIME type multipart/signed. This is the default if you do not specify any *flags* to openssl_pkcs7_sign(). If you turn this option off, the message will be signed using opaque signing, which is more resistant to translation by mail relays but cannot be read by mail agents that do not support S/MIME.
PKCS7_NOSIGS	Don't try and verify the signatures on a message

Note
These constants were added in 4.0.6.

Signature Algorithms

OPENSSL_ALGO_SHA1 (integer)
Used as default algorithm by openssl_sign() and openssl_verify().

OPENSSL_ALGO_MD5 (integer)

OPENSSL_ALGO_MD4 (integer)

OPENSSL_ALGO_MD2 (integer)

Note
These constants were added in 5.0.0.

Ciphers

OPENSSL_CIPHER_RC2_40 (integer)

OPENSSL_CIPHER_RC2_128 (integer)

OPENSSL_CIPHER_RC2_64 (integer)

OPENSSL_CIPHER_DES (integer)

OPENSSL_CIPHER_3DES (integer)

Note
These constants were added in 4.3.0.

Version constants

OPENSSL_VERSION_TEXT (string)

OPENSSL_VERSION_NUMBER (integer)

Note
These constants were added in 5.2.0.

Key/Certificate parameters

Quite a few of the openssl functions require a key or a certificate parameter. PHP 4.0.5 and earlier have to use a key or certificate resource returned by one of the openssl_get_xxx functions. Later versions may use one of the following methods:

- Certificates
  - An X.509 resource returned from openssl_x509_read()
  - A string having the format *file://path/to/cert.pem*; the named file must contain a PEM encoded certificate
  - A string containing the content of a certificate, PEM encoded

- Public/Private Keys
  - A key resource returned from openssl_get_publickey() or openssl_get_privatekey()
  - For public keys only: an X.509 resource
  - A string having the format *file://path/to/file.pem* - the named file must contain a PEM encoded certificate/private key (it may contain both)
  - A string containing the content of a certificate/key, PEM encoded
  - For private keys, you may also use the syntax *array($key, $passphrase)* where $key represents a key specified using the file:// or textual content notation above, and $passphrase represents a string containing the passphrase for that private key

Certificate Verification

When calling a function that will verify a signature/certificate, the *cainfo* parameter is an array containing file and directory names that specify the locations of trusted CA files. If a directory is specified, then it must be a correctly formed hashed directory as the *openssl* command would use.

OpenSSL Functions

openssl_csr_export_to_file

openssl_csr_export_to_file -- Exports a CSR to a file

Description

bool **openssl_csr_export_to_file** (resource `$csr`, string `$outfilename` [, bool `$notext`])

openssl_csr_export_to_file() takes the Certificate Signing Request represented by *`csr`* and saves it as ascii-armoured text into the file named by *`outfilename`*.

Parameters

`csr`

`outfilename`
: Path to the output file.

`notext`
: The optional parameter *`notext`* affects the verbosity of the output; if it is **FALSE**, then additional human-readable information is included in the output. The default value of *`notext`* is **TRUE**.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- openssl_csr_export()
- openssl_csr_new()
- openssl_csr_sign()

openssl_csr_export

openssl_csr_export -- Exports a CSR as a string

Description

bool **openssl_csr_export** (resource `$csr`, string `&$out` [, bool `$notext`])

openssl_csr_export() takes the Certificate Signing Request represented by *`csr`* and stores it as ascii-armoured text into *`out`*, which is passed by reference.

Parameters

`csr`

`out`

`notext`
The optional parameter *`notext`* affects the verbosity of the output; if it is **FALSE**, then additional human-readable information is included in the output. The default value of *`notext`* is **TRUE**.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- openssl_csr_export_to_file()
- openssl_csr_new()
- openssl_csr_sign()

openssl_csr_get_public_key

openssl_csr_get_public_key -- Returns the public key of a CERT

Description

resource **openssl_csr_get_public_key** (mixed `$csr` [, bool `$use_shortnames`])

Warning
This function is currently not documented; only its argument list is available.

openssl_csr_get_subject

openssl_csr_get_subject -- Returns the subject of a CERT

Description

array **openssl_csr_get_subject** (mixed `$csr` [, bool `$use_shortnames`])

Warning
This function is currently not documented; only its argument list is available.

openssl_csr_new

openssl_csr_new -- Generates a CSR

Description

mixed **openssl_csr_new** (array `$dn`, resource `&$privkey` [, array `$configargs` [, array `$extraattribs`]])

openssl_csr_new() generates a new CSR (Certificate Signing Request) based on the information provided by *dn*, which represents the Distinguished Name to be used in the certificate.

Note
You need to have a valid *openssl.cnf* installed for this function to operate correctly. See the notes under the installation section for more information.

Parameters

dn
: The Distinguished Name to be used in the certificate.

privkey
: *privkey* should be set to a private key that was previously generated by openssl_pkey_new() (or otherwise obtained from the other openssl_pkey family of functions). The corresponding public portion of the key will be used to sign the CSR.

configargs
: By default, the information in your system *openssl.conf* is used to initialize the request; you can specify a configuration file section by setting the *config_section_section* key of *configargs*. You can also specify an alternative openssl configuration file by setting the value of the *config* key to the path of the file you want to use. The following keys, if present in *configargs* behave as their equivalents in the *openssl.conf*, as listed in the table below.

Configuration overrides

configargs **key**	**type**	*openssl.conf* **equivalent**	**descriptio**
digest_alg	string	default_md	Selects wh to use
x509_extensions	string	x509_extensions	Selects wh should be creating a

req_extensions	string	req_extensions	Selects wh should be creating a
private_key_bits	integer	default_bits	Specifies h should be private key
private_key_type	integer	none	Specifies t key to crea one of **OPENSSL** **OPENSSL** or **OPENSSL** The defau **OPENSSL** which is cu supported
encrypt_key	boolean	encrypt_key	Should an passphras

extraattribs
: *extraattribs* is used to specify additional configuration options for the CSR. Both *dn* and *extraattribs* are associative arrays whose keys are converted to OIDs and applied to the relevant part of the request.

Return Values

Returns the CSR.

Examples

Example #1 - Creating a self-signed-certificate

```
<?php
// Fill in data for the distinguished name to be used in the cert
// You must change the values of these keys to match your name and
// company, or more precisely, the name and company of the person/site
// that you are generating the certificate for.
// For SSL certificates, the commonName is usually the domain name of
// that will be using the certificate, but for S/MIME certificates,
// the commonName will be the name of the individual who will use the
// certificate.
$dn = array(
   "countryName" => "UK",
   "stateOrProvinceName" => "Somerset",
```

```
   "localityName" => "Glastonbury",
   "organizationName" => "The Brain Room Limited",
   "organizationalUnitName" => "PHP Documentation Team",
   "commonName" => "Wez Furlong",
   "emailAddress" => "wez@example.com"
);

// Generate a new private (and public) key pair
$privkey = openssl_pkey_new();

// Generate a certificate signing request
$csr = openssl_csr_new($dn, $privkey);

// You will usually want to create a self-signed certificate at this
// point until your CA fulfills your request.
// This creates a self-signed cert that is valid for 365 days
$sscert = openssl_csr_sign($csr, null, $privkey, 365);

// Now you will want to preserve your private key, CSR and self-signed
// cert so that they can be installed into your web server, mail server
// or mail client (depending on the intended use of the certificate).
// This example shows how to get those things into variables, but you
// can also store them directly into files.
// Typically, you will send the CSR on to your CA who will then issue
// you with the "real" certificate.
openssl_csr_export($csr, $csrout) and var_dump($csrout);
openssl_x509_export($sscert, $certout) and var_dump($certout);
openssl_pkey_export($privkey, $pkeyout, "mypassword") and var_dump($pkeyout);

// Show any errors that occurred here
while (($e = openssl_error_string()) !== false) {
   echo $e . "\n";
}
?>
```

openssl_csr_sign

openssl_csr_sign -- Sign a CSR with another certificate (or itself) and generate a certificate

Description

resource **openssl_csr_sign** (mixed `$csr`, mixed `$cacert`, mixed `$priv_key`, int `$days` [, array `$configargs` [, int `$serial`]])

openssl_csr_sign() generates an x509 certificate resource from the given CSR.

Note
You need to have a valid *openssl.cnf* installed for this function to operate correctly. See the notes under the installation section for more information.

Parameters

`csr`
A CSR previously generated by openssl_csr_new(). It can also be the path to a PEM encoded CSR when specified as *file://path/to/csr* or an exported string generated by openssl_csr_export().

`cacert`
The generated certificate will be signed by *`cacert`*. If *`cacert`* is **NULL**, the generated certificate will be a self-signed certificate.

`priv_key`
`priv_key` is the private key that corresponds to *`cacert`*.

`days`
`days` specifies the length of time for which the generated certificate will be valid, in days.

`configargs`
You can finetune the CSR signing by *`configargs`*. See openssl_csr_new() for more information about *`configargs`*.

`serial`
An optional the serial number of issued certificate. If not specified it will default to 0.

Return Values

Returns an x509 certificate resource on success, **FALSE** on failure.

ChangeLog

Version	Description
4.3.3	The *serial* parameter was added.

Examples

Example #2 - openssl_csr_sign() example - signing a CSR (how to implement your own CA)

```
<?php
// Let's assume that this script is set to receive a CSR that has
// been pasted into a textarea from another page
$csrdata = $_POST["CSR"];

// We will sign the request using our own "certificate authority"
// certificate.  You can use any certificate to sign another, but
// the process is worthless unless the signing certificate is trusted
// by the software/users that will deal with the newly signed certificate

// We need our CA cert and its private key
$cacert = "file://path/to/ca.crt";
$privkey = array("file://path/to/ca.key", "your_ca_key_passphrase");

$userscert = openssl_csr_sign($csrdata, $cacert, $privkey, 365);

// Now display the generated certificate so that the user can
// copy and paste it into their local configuration (such as a file
// to hold the certificate for their SSL server)
openssl_x509_export($usercert, $certout);
echo $certout;

// Show any errors that occurred here
while (($e = openssl_error_string()) !== false) {
   echo $e . "\n";
}
?>
```

openssl_error_string

openssl_error_string -- Return openSSL error message

Description

string **openssl_error_string** (void)

openssl_error_string() returns the last error from the openSSL library. Error messages are stacked, so this function should be called multiple times to collect all of the information.

Return Values

Returns an error message string, or **FALSE** if there are no more error messages to return.

Examples

Example #3 - openssl_error_string() example

```
<?php
// lets assume you just called an openssl function that failed
while ($msg = openssl_error_string())
   echo $msg . "<br />\n";
?>
```

openssl_free_key

openssl_free_key -- Free key resource

Description

void **openssl_free_key** (resource `$key_identifier`)

openssl_free_key() frees the key associated with the specified *`key_identifier`* from memory.

Parameters

`key_identifier`

Return Values

No value is returned.

openssl_get_privatekey

openssl_get_privatekey -- Alias of openssl_pkey_get_private()

Description

This function is an alias of: openssl_pkey_get_private().

openssl_get_publickey

openssl_get_publickey -- Alias of openssl_pkey_get_public()

Description

This function is an alias of: openssl_pkey_get_public().

openssl_open

openssl_open -- Open sealed data

Description

bool **openssl_open** (string `$sealed_data`, string `&$open_data`, string `$env_key`, mixed `$priv_key_id`)

openssl_open() opens (decrypts) *`sealed_data`* using the private key associated with the key identifier *`priv_key_id`* and the envelope key *`env_key`*, and fills *`open_data`* with the decrypted data. The envelope key is generated when the data are sealed and can only be used by one specific private key. See openssl_seal() for more information.

Parameters

`sealed_data`

`open_data`
If the call is successful the opened data is returned in this parameter.

`env_key`

`priv_key_id`

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #4 - openssl_open() example

```
<?php
// $sealed and $env_key are assumed to contain the sealed data
// and our envelope key, both given to us by the sealer.

// fetch private key from file and ready it
$fp = fopen("/src/openssl-0.9.6/demos/sign/key.pem", "r");
$priv_key = fread($fp, 8192);
fclose($fp);
$pkeyid = openssl_get_privatekey($priv_key);
```

```
// decrypt the data and store it in $open
if (openssl_open($sealed, $open, $env_key, $pkeyid)) {
   echo "here is the opened data: ", $open;
} else {
   echo "failed to open data";
}

// free the private key from memory
openssl_free_key($pkeyid);
?>
```

See Also

- openssl_seal()

openssl_pkcs12_export_to_file

openssl_pkcs12_export_to_file -- Exports a PKCS#12 Compatible Certificate Store File

Description

bool **openssl_pkcs12_export_to_file** (mixed `$x509`, string `$filename`, mixed `$priv_key`, string `$pass` [, array `$args`])

openssl_pkcs12_export_to_file() stores *`x509`* into a file named by *`filename`* in a PKCS#12 file format.

Parameters

`x509`

`filename`
Path to the output file.

`priv_key`
Private key component of PKCS#12 file.

`pass`
Encryption password for unlocking the PKCS#12 file.

`args`

Return Values

Returns **TRUE** on success or **FALSE** on failure.

openssl_pkcs12_export

openssl_pkcs12_export -- Exports a PKCS#12 Compatible Certificate Store File to variable.

Description

bool **openssl_pkcs12_export** (mixed `$x509`, string `&$out`, mixed `$priv_key`, string `$pass` [, array `$args`])

openssl_pkcs12_export() stores *`x509`* into a string named by *`out`* in a PKCS#12 file format.

Parameters

`x509`

`out`
: On success, this will hold the PKCS#12.

`priv_key`
: Private key component of PKCS#12 file.

`pass`
: Encryption password for unlocking the PKCS#12 file.

`args`

Return Values

Returns **TRUE** on success or **FALSE** on failure.

openssl_pkcs12_read

openssl_pkcs12_read -- Parse a PKCS#12 Certificate Store into an array

Description

bool **openssl_pkcs12_read** (mixed `$PKCS12`, array `&$certs`, string `$pass`)

openssl_pkcs12_read() parses the PKCS#12 certificate store supplied by *`PKCS12`* into a array named *`certs`*.

Parameters

`PKCS12`

`certs`
: On success, this will hold the Certificate Store Data.

`pass`
: Encryption password for unlocking the PKCS#12 file.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

openssl_pkcs7_decrypt

openssl_pkcs7_decrypt -- Decrypts an S/MIME encrypted message

Description

bool **openssl_pkcs7_decrypt** (string `$infilename`, string `$outfilename`, mixed `$recipcert` [, mixed `$recipkey`])

Decrypts the S/MIME encrypted message contained in the file specified by *`infilename`* using the certificate and its associated private key specified by *`recipcert`* and *`recipkey`*.

Parameters

`infilename`

`outfilename`
The decrypted message is written to the file specified by *`outfilename`*.

`recipcert`

`recipkey`

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #5 - openssl_pkcs7_decrypt() example

```
<?php
// $cert and $key are assumed to contain your personal certificate and
private
// key pair, and that you are the recipient of an S/MIME message
$infilename = "encrypted.msg";  // this file holds your encrypted message
$outfilename = "decrypted.msg"; // make sure you can write to this file

if (openssl_pkcs7_decrypt($infilename, $outfilename, $cert, $key)) {
   echo "decrypted!";
} else {
   echo "failed to decrypt!";
}
```

```
?>
```

openssl_pkcs7_encrypt

openssl_pkcs7_encrypt -- Encrypt an S/MIME message

Description

bool **openssl_pkcs7_encrypt** (string `$infile`, string `$outfile`, mixed `$recipcerts`, array `$headers` [, int `$flags` [, int `$cipherid`]])

openssl_pkcs7_encrypt() takes the contents of the file named *`infile`* and encrypts them using an RC2 40-bit cipher so that they can only be read by the intended recipients specified by *`recipcerts`*.

Parameters

`infile`

`outfile`

`recipcerts`
: Either a lone X.509 certificate, or an array of X.509 certificates.

`headers`
: *`headers`* is an array of headers that will be prepended to the data after it has been encrypted. *`headers`* can be either an associative array keyed by header name, or an indexed array, where each element contains a single header line.

`flags`
: *`flags`* can be used to specify options that affect the encoding process - see PKCS7 constants.

`cipherid`
: Cipher can be selected with *`cipherid`*.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

ChangeLog

Version	Description

5.0.0	The *cipherid* parameter was added.

Examples

Example #6 - openssl_pkcs7_encrypt() example

```
<?php
// the message you want to encrypt and send to your secret agent
// in the field, known as nighthawk.  You have his certificate
// in the file nighthawk.pem
$data = <<<EOD
Nighthawk,

Top secret, for your eyes only!

The enemy is closing in! Meet me at the cafe at 8.30am
to collect your forged passport!

HQ
EOD;

// load key
$key = file_get_contents("nighthawk.pem");

// save message to file
$fp = fopen("msg.txt", "w");
fwrite($fp, $data);
fclose($fp);

// encrypt it
if (openssl_pkcs7_encrypt("msg.txt", "enc.txt", $key,
   array("To" => "nighthawk@example.com", // keyed syntax
         "From: HQ <hq@example.com>", // indexed syntax
         "Subject" => "Eyes only"))) {
   // message encrypted - send it!
   exec(ini_get("sendmail_path") . " < enc.txt");
}
?>
```

openssl_pkcs7_sign

openssl_pkcs7_sign -- Sign an S/MIME message

Description

bool **openssl_pkcs7_sign** (string `$infilename`, string `$outfilename`, mixed `$signcert`, mixed `$privkey`, array `$headers` [, int `$flags` [, string `$extracerts`]])

openssl_pkcs7_sign() takes the contents of the file named *`infilename`* and signs them using the certificate and its matching private key specified by *`signcert`* and *`privkey`* parameters.

Parameters

`infilename`

`outfilename`

`signcert`

`privkey`

`headers`
: *`headers`* is an array of headers that will be prepended to the data after it has been signed (see openssl_pkcs7_encrypt() for more information about the format of this parameter.

`flags`
: *`flags`* can be used to alter the output - see PKCS7 constants - if not specified, it defaults to PKCS7_DETACHED.

`extracerts`
: *`extracerts`* specifies the name of a file containing a bunch of extra certificates to include in the signature which can for example be used to help the recipient to verify the certificate that you used.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #7 - openssl_pkcs7_sign() example

```
<?php
// the message you want to sign so that recipient can be sure it was you
that
// sent it
$data = <<<EOD

You have my authorization to spend $10,000 on dinner expenses.

The CEO
EOD;
// save message to file
$fp = fopen("msg.txt", "w");
fwrite($fp, $data);
fclose($fp);
// encrypt it
if (openssl_pkcs7_sign("msg.txt", "signed.txt", "mycert.pem",
   array("file://mycert.pem", "mypassphrase"),
   array("To" => "joes@example.com", // keyed syntax
         "From: HQ <ceo@example.com>", // indexed syntax
         "Subject" => "Eyes only")
   )) {
   // message signed - send it!
   exec(ini_get("sendmail_path") . " < signed.txt");
}
?>
```

openssl_pkcs7_verify

openssl_pkcs7_verify -- Verifies the signature of an S/MIME signed message

Description

mixed **openssl_pkcs7_verify** (string `$filename`, int `$flags` [, string `$outfilename` [, array `$cainfo` [, string `$extracerts` [, string `$content`]]]])

openssl_pkcs7_verify() reads the S/MIME message contained in the given file and examines the digital signature.

Parameters

`filename`
: Path to the message.

`flags`
: *`flags`* can be used to affect how the signature is verified - see PKCS7 constants for more information.

`outfilename`
: If the *`outfilename`* is specified, it should be a string holding the name of a file into which the certificates of the persons that signed the messages will be stored in PEM format.

`cainfo`
: If the *`cainfo`* is specified, it should hold information about the trusted CA certificates to use in the verification process - see certificate verification for more information about this parameter.

`extracerts`
: If the *`extracerts`* is specified, it is the filename of a file containing a bunch of certificates to use as untrusted CAs.

`content`
: You can specify a filename with *`content`* that will be filled with the verified data, but with the signature information stripped.

Return Values

Returns **TRUE** if the signature is verified, **FALSE** if it is not correct (the message has been tampered with, or the signing certificate is invalid), or -1 on error.

ChangeLog

Version	Description
5.1.0	The `content` parameter was added.

openssl_pkey_export_to_file

openssl_pkey_export_to_file -- Gets an exportable representation of a key into a file

Description

bool **openssl_pkey_export_to_file** (mixed `$key`, string `$outfilename` [, string `$passphrase` [, array `$configargs`]])

openssl_pkey_export_to_file() saves an ascii-armoured (PEM encoded) rendition of *`key`* into the file named by *`outfilename`*.

Note
You need to have a valid *openssl.cnf* installed for this function to operate correctly. See the notes under the installation section for more information.

Parameters

`key`

`outfilename`
: Path to the output file.

`passphrase`
: The key can be optionally protected by a *`passphrase`*.

`configargs`
: *`configargs`* can be used to fine-tune the export process by specifying and/or overriding options for the openssl configuration file. See openssl_csr_new() for more information about *`configargs`*.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

openssl_pkey_export

openssl_pkey_export -- Gets an exportable representation of a key into a string

Description

bool **openssl_pkey_export** (mixed `$key`, string `&$out` [, string `$passphrase` [, array `$configargs`]])

openssl_pkey_export() exports *`key`* as a PEM encoded string and stores it into *`out`* (which is passed by reference).

Note
You need to have a valid *openssl.cnf* installed for this function to operate correctly. See the notes under the installation section for more information.

Parameters

`key`

`out`

`passphrase`
The key is optionally protected by *`passphrase`*.

`configargs`
`configargs` can be used to fine-tune the export process by specifying and/or overriding options for the openssl configuration file. See openssl_csr_new() for more information about *`configargs`*.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

openssl_pkey_free

openssl_pkey_free -- Frees a private key

Description

void **openssl_pkey_free** (resource `$key`)

This function frees a private key created by openssl_pkey_new().

Parameters

key
: Resource holding the key.

Return Values

No value is returned.

openssl_pkey_get_details

openssl_pkey_get_details -- Returns an array with the key details

Description

array **openssl_pkey_get_details** (resource `$key`)

This function returns the key details (bits, key, type).

Parameters

key
: Resource holding the key.

Return Values

Returns an array with the key details in success or **FALSE** in failure. Returned array has indexes *bits* (number of bits), *key* (string representation of the public key) and *type* (type of the key which is one of **OPENSSL_KEYTYPE_RSA**, **OPENSSL_KEYTYPE_DSA**, **OPENSSL_KEYTYPE_DH**, **OPENSSL_KEYTYPE_EC** or -1 meaning unknown).

openssl_pkey_get_private

openssl_pkey_get_private -- Get a private key

Description

resource **openssl_pkey_get_private** (mixed `$key` [, string `$passphrase`])

openssl_get_privatekey() parses *`key`* and prepares it for use by other functions.

Parameters

`key`
: *`key`* can be one of the following:

  - a string having the format *file://path/to/file.pem*. The named file must contain a PEM encoded certificate/private key (it may contain both).
  - A PEM formatted private key.

`passphrase`
: The optional parameter *`passphrase`* must be used if the specified key is encrypted (protected by a passphrase).

Return Values

Returns a positive key resource identifier on success, or **FALSE** on error.

openssl_pkey_get_public

openssl_pkey_get_public -- Extract public key from certificate and prepare it for use

Description

resource **openssl_pkey_get_public** (mixed `$certificate`)

openssl_get_publickey() extracts the public key from *`certificate`* and prepares it for use by other functions.

Parameters

`certificate`
`certificate` can be one of the following:

- an X.509 certificate resource
- a string having the format *file://path/to/file.pem*. The named file must contain a PEM encoded certificate/private key (it may contain both).
- A PEM formatted private key.

Return Values

Returns a positive key resource identifier on success, or **FALSE** on error.

openssl_pkey_new

openssl_pkey_new -- Generates a new private key

Description

resource **openssl_pkey_new** ([array `$configargs`])

openssl_pkey_new() generates a new private and public key pair. The public component of the key can be obtained using openssl_pkey_get_public().

Note
You need to have a valid *openssl.cnf* installed for this function to operate correctly. See the notes under the installation section for more information.

Parameters

`configargs`
: You can finetune the key generation (such as specifying the number of bits) using *`configargs`*. See openssl_csr_new() for more information about *`configargs`*.

Return Values

Returns a resource identifier for the pkey on success, or **FALSE** on error.

openssl_private_decrypt

openssl_private_decrypt -- Decrypts data with private key

Description

bool **openssl_private_decrypt** (string `$data`, string `&$decrypted`, mixed `$key` [, int `$padding`])

openssl_private_decrypt() decrypts *`data`* that was previous encrypted via openssl_public_encrypt() and stores the result into *`decrypted`*.

You can use this function e.g. to decrypt data which were supposed only to you.

Parameters

`data`

`decrypted`

`key`
: *`key`* must be the private key corresponding that was used to encrypt the data.

`padding`
: *`padding`* defaults to **OPENSSL_PKCS1_PADDING**, but can also be one of **OPENSSL_SSLV23_PADDING**, **OPENSSL_PKCS1_OAEP_PADDING**, **OPENSSL_NO_PADDING**.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- openssl_public_encrypt()
- openssl_public_decrypt()

openssl_private_encrypt

openssl_private_encrypt -- Encrypts data with private key

Description

bool **openssl_private_encrypt** (string `$data`, string `&$crypted`, mixed `$key` [, int `$padding`])

openssl_private_encrypt() encrypts *`data`* with private *`key`* and stores the result into *`crypted`*. Encrypted data can be decrypted via openssl_public_decrypt().

This function can be used e.g. to sign data (or its hash) to prove that it is not written by someone else.

Parameters

`data`

`crypted`

`key`

`padding`
: *`padding`* defaults to **OPENSSL_PKCS1_PADDING**, but can also be **OPENSSL_NO_PADDING**.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- openssl_public_encrypt()
- openssl_public_decrypt()

openssl_public_decrypt

openssl_public_decrypt -- Decrypts data with public key

Description

bool **openssl_public_decrypt** (string `$data`, string `&$decrypted`, mixed `$key` [, int `$padding`])

openssl_public_decrypt() decrypts *`data`* that was previous encrypted via openssl_private_encrypt() and stores the result into *`decrypted`*.

You can use this function e.g. to check if the message was written by the owner of the private key.

Parameters

`data`

`decrypted`

`key`
: *`key`* must be the public key corresponding that was used to encrypt the data.

`padding`
: *`padding`* defaults to **OPENSSL_PKCS1_PADDING**, but can also be **OPENSSL_NO_PADDING**.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- openssl_private_encrypt()
- openssl_private_decrypt()

openssl_public_encrypt

openssl_public_encrypt -- Encrypts data with public key

Description

bool **openssl_public_encrypt** (string `$data`, string `&$crypted`, mixed `$key` [, int `$padding`])

openssl_public_encrypt() encrypts *`data`* with public *`key`* and stores the result into *`crypted`*. Encrypted data can be decrypted via openssl_private_decrypt().

This function can be used e.g. to encrypt message which can be then read only by owner of the private key. It can be also used to store secure data in database.

Parameters

`data`

`crypted`
: This will hold the result of the encryption.

`key`
: The public key.

`padding`
: *`padding`* defaults to **OPENSSL_PKCS1_PADDING**, but can also be one of **OPENSSL_SSLV23_PADDING**, **OPENSSL_PKCS1_OAEP_PADDING**, **OPENSSL_NO_PADDING**.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- openssl_private_encrypt()
- openssl_private_decrypt()

openssl_seal

openssl_seal -- Seal (encrypt) data

Description

int **openssl_seal** (string `$data`, string `&$sealed_data`, array `&$env_keys`, array `$pub_key_ids`)

openssl_seal() seals (encrypts) *`data`* by using RC4 with a randomly generated secret key. The key is encrypted with each of the public keys associated with the identifiers in *`pub_key_ids`* and each encrypted key is returned in *`env_keys`*. This means that one can send sealed data to multiple recipients (provided one has obtained their public keys). Each recipient must receive both the sealed data and the envelope key that was encrypted with the recipient's public key.

Parameters

`data`

`sealed_data`

`env_keys`

`pub_key_ids`

Return Values

Returns the length of the sealed data on success, or **FALSE** on error. If successful the sealed data is returned in *`sealed_data`*, and the envelope keys in *`env_keys`*.

Examples

Example #8 - openssl_seal() example

```
<?php
// $data is assumed to contain the data to be sealed

// fetch public keys for our recipients, and ready them
$fp = fopen("/src/openssl-0.9.6/demos/maurice/cert.pem", "r");
$cert = fread($fp, 8192);
fclose($fp);
```

```
$pk1 = openssl_get_publickey($cert);
// Repeat for second recipient
$fp = fopen("/src/openssl-0.9.6/demos/sign/cert.pem", "r");
$cert = fread($fp, 8192);
fclose($fp);
$pk2 = openssl_get_publickey($cert);

// seal message, only owners of $pk1 and $pk2 can decrypt $sealed with keys
// $ekeys[0] and $ekeys[1] respectively.
openssl_seal($data, $sealed, $ekeys, array($pk1, $pk2));

// free the keys from memory
openssl_free_key($pk1);
openssl_free_key($pk2);
?>
```

See Also

- openssl_open()

openssl_sign

openssl_sign -- Generate signature

Description

bool **openssl_sign** (string `$data`, string `&$signature`, mixed `$priv_key_id` [, int `$signature_alg`])

openssl_sign() computes a signature for the specified *`data`* by using SHA1 for hashing followed by encryption using the private key associated with *`priv_key_id`*. Note that the data itself is not encrypted.

Parameters

`data`

`signature`
If the call was successful the signature is returned in *`signature`*.

`priv_key_id`

`signature_alg`
Defaults to **OPENSSL_ALGO_SHA1**. For more information see the list of Signature Algorithms.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

ChangeLog

Version	Description
5.0.0	The *`signature_alg`* parameter was added.

Examples

Example #9 - openssl_sign() example

```
<?php
// $data is assumed to contain the data to be signed

// fetch private key from file and ready it
$fp = fopen("/src/openssl-0.9.6/demos/sign/key.pem", "r");
$priv_key = fread($fp, 8192);
fclose($fp);
$pkeyid = openssl_get_privatekey($priv_key);

// compute signature
openssl_sign($data, $signature, $pkeyid);

// free the key from memory
openssl_free_key($pkeyid);
?>
```

See Also

- openssl_verify()

openssl_verify

openssl_verify -- Verify signature

Description

int **openssl_verify** (string `$data`, string `$signature`, mixed `$pub_key_id` [, int `$signature_alg`])

openssl_verify() verifies that the *`signature`* is correct for the specified *`data`* using the public key associated with *`pub_key_id`*. This must be the public key corresponding to the private key used for signing.

Parameters

`data`

`signature`

`pub_key_id`

`signature_alg`
Defaults to **OPENSSL_ALGO_SHA1**. For more information see the list of Signature Algorithms.

Return Values

Returns 1 if the signature is correct, 0 if it is incorrect, and -1 on error.

ChangeLog

Version	Description
5.0.0	The *`signature_alg`* parameter was added.

Examples

Example #10 - openssl_verify() example

```
<?php
// $data and $signature are assumed to contain the data and the signature

// fetch public key from certificate and ready it
$fp = fopen("/src/openssl-0.9.6/demos/sign/cert.pem", "r");
$cert = fread($fp, 8192);
fclose($fp);
$pubkeyid = openssl_get_publickey($cert);

// state whether signature is okay or not
$ok = openssl_verify($data, $signature, $pubkeyid);
if ($ok == 1) {
   echo "good";
} elseif ($ok == 0) {
   echo "bad";
} else {
   echo "ugly, error checking signature";
}
// free the key from memory
openssl_free_key($pubkeyid);
?>
```

See Also

- openssl_sign()

openssl_x509_check_private_key

openssl_x509_check_private_key -- Checks if a private key corresponds to a certificate

Description

bool **openssl_x509_check_private_key** (mixed `$cert`, mixed `$key`)

Checks whether the given *`key`* is the private key that corresponds to *`cert`*.

Parameters

`cert`
The certificate.

`key`
The private key.

Return Values

Returns **TRUE** if *`key`* is the private key that corresponds to *`cert`*, or **FALSE** otherwise.

openssl_x509_checkpurpose

openssl_x509_checkpurpose -- Verifies if a certificate can be used for a particular purpose

Description

int **openssl_x509_checkpurpose** (mixed `$x509cert`, int `$purpose` [, array `$cainfo` [, string `$untrustedfile`]])

openssl_x509_checkpurpose() examines a certificate to see if it can be used for the specified *`purpose`*.

Parameters

`x509cert`
The examined certificate.

`purpose`

openssl_x509_checkpurpose() **purposes**

Constant	Description
X509_PURPOSE_SSL_CLIENT	Can the certificate be used for the client side of an SSL connection?
X509_PURPOSE_SSL_SERVER	Can the certificate be used for the server side of an SSL connection?
X509_PURPOSE_NS_SSL_SERVER	Can the cert be used for Netscape SSL server?
X509_PURPOSE_SMIME_SIGN	Can the cert be used to sign S/MIME email?
X509_PURPOSE_SMIME_ENCRYPT	Can the cert be used to encrypt S/MIME email?
X509_PURPOSE_CRL_SIGN	Can the cert be used to sign a certificate revocation list (CRL)?
X509_PURPOSE_ANY	Can the cert be used for Any/All purposes?

These options are not bitfields - you may specify one only!

`cainfo`
`cainfo` should be an array of trusted CA files/dirs as described in Certificate Verification. It defaults to an empty array.

untrustedfile
: If specified, this should be the name of a PEM encoded file holding certificates that can be used to help verify the certificate, although no trust in placed in the certificates that come from that file.

Return Values

Returns **TRUE** if the certificate can be used for the intended purpose, **FALSE** if it cannot, or -1 on error.

openssl_x509_export_to_file

openssl_x509_export_to_file -- Exports a certificate to file

Description

bool **openssl_x509_export_to_file** (mixed `$x509`, string `$outfilename` [, bool `$notext`])

openssl_x509_export_to_file() stores *`x509`* into a file named by *`outfilename`* in a PEM encoded format.

Parameters

`x509`

`outfilename`
: Path to the output file.

`notext`
: The optional parameter *`notext`* affects the verbosity of the output; if it is **FALSE**, then additional human-readable information is included in the output. The default value of *`notext`* is **TRUE**.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

openssl_x509_export

openssl_x509_export -- Exports a certificate as a string

Description

bool **openssl_x509_export** (mixed `$x509`, string `&$output` [, bool `$notext`])

openssl_x509_export() stores *`x509`* into a string named by *`output`* in a PEM encoded format.

Parameters

`x509`

`output`
On success, this will hold the PEM.

`notext`
The optional parameter *`notext`* affects the verbosity of the output; if it is **FALSE**, then additional human-readable information is included in the output. The default value of *`notext`* is **TRUE**.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

openssl_x509_free

openssl_x509_free -- Free certificate resource

Description

void **openssl_x509_free** (resource `$x509cert`)

openssl_x509_free() frees the certificate associated with the specified *`x509cert`* resource from memory.

Parameters

`x509cert`

Return Values

No value is returned.

openssl_x509_parse

openssl_x509_parse -- Parse an X509 certificate and return the information as an array

Description

array **openssl_x509_parse** (mixed `$x509cert` [, bool `$shortnames`])

openssl_x509_parse() returns information about the supplied *`x509cert`*, including fields such as subject name, issuer name, purposes, valid from and valid to dates etc.

Parameters

`x509cert`

`shortnames`

`shortnames` controls how the data is indexed in the array - if *`shortnames`* is **TRUE** (the default) then fields will be indexed with the short name form, otherwise, the long name form will be used - e.g.: CN is the shortname form of commonName.

Return Values

The structure of the returned data is (deliberately) not yet documented, as it is still subject to change.

openssl_x509_read

openssl_x509_read -- Parse an X.509 certificate and return a resource identifier for it

Description

resource **openssl_x509_read** (mixed `$x509certdata`)

openssl_x509_read() parses the certificate supplied by *`x509certdata`* and returns a resource identifier for it.

Parameters

`x509certdata`

Return Values

Returns a resource identifier on success, or **FALSE** on failure.