NAME

Crypt::OpenSSL::RSA − RSA encoding and decoding, using the openSSL libraries

SYNOPSIS

```
use Crypt::OpenSSL::Random;
use Crypt::OpenSSL::RSA;

# not necessary if we have /dev/random:
Crypt::OpenSSL::Random::random_seed($good_entropy);
Crypt::OpenSSL::RSA->import_random_seed();
$rsa_pub = Crypt::OpenSSL::RSA->new_public_key($key_string);
$rsa_pub->use_sslv23_padding(); # use_pkcs1_oaep_padding is the default
$ciphertext = $rsa->encrypt($plaintext);

$rsa_priv = Crypt::OpenSSL::RSA->new_private_key($key_string);
$plaintext = $rsa->encrypt($ciphertext);

$rsa = Crypt::OpenSSL::RSA->generate_key(1024); # or
$rsa = Crypt::OpenSSL::RSA->generate_key(1024, $prime);

print "private key is:\n", $rsa->get_private_key_string();
print "public key (in PKCS1 format) is:\n",
      $rsa->get_public_key_string();
print "public key (in X509 format) is:\n",
      $rsa->get_public_key_x509_string();

$rsa_priv->use_md5_hash(); # insecure. use_sha256_hash or use_sha1_hash are the
$signature = $rsa_priv->sign($plaintext);
print "Signed correctly\n" if ($rsa->verify($plaintext, $signature));
```

DESCRIPTION

`Crypt::OpenSSL::RSA` provides the ability to RSA encrypt strings which are somewhat shorter than the block size of a key. It also allows for decryption, signatures and signature verification.

NOTE: Many of the methods in this package can croak, so use `eval`, or Error.pm's try/catch mechanism to capture errors. Also, while some methods from earlier versions of this package return true on success, this (never documented) behavior is no longer the case.

Class Methods

new_public_key

Create a new `Crypt::OpenSSL::RSA` object by loading a public key in from a string containing Base64/DER−encoding of either the PKCS1 or X.509 representation of the key. The string should include the `-----BEGIN...-----` and `-----END...-----` lines.

The padding is set to PKCS1_OAEP, but can be changed with the `use_xxx_padding` methods.

new_private_key

Create a new `Crypt::OpenSSL::RSA` object by loading a private key in from an string containing the Base64/DER encoding of the PKCS1 representation of the key. The string should include the `-----BEGIN...-----` and `-----END...-----` lines. The padding is set to PKCS1_OAEP, but can be changed with `use_xxx_padding`.

generate_key

Create a new `Crypt::OpenSSL::RSA` object by constructing a private/public key pair. The first (mandatory) argument is the key size, while the second optional argument specifies the public exponent (the default public exponent is 65537). The padding is set to `PKCS1_OAEP`, but can be changed with use_xxx_padding methods.

new_key_from_parameters
: Given Crypt::OpenSSL::Bignum objects for n, e, and optionally d, p, and q, where p and q are the prime factors of n, e is the public exponent and d is the private exponent, create a new Crypt::OpenSSL::RSA object using these values. If p and q are provided and d is undef, d is computed. Note that while p and q are not necessary for a private key, their presence will speed up computation.

import_random_seed
: Import a random seed from Crypt::OpenSSL::Random, since the OpenSSL libraries won't allow sharing of random structures across perl XS modules.

Instance Methods

DESTROY
: Clean up after ourselves. In particular, erase and free the memory occupied by the RSA key structure.

get_public_key_string
: Return the Base64/DER−encoded PKCS1 representation of the public key. This string has header and footer lines:

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

get_public_key_x509_string
: Return the Base64/DER−encoded representation of the ‘‘subject public key’’, suitable for use in X509 certificates. This string has header and footer lines:

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

and is the format that is produced by running `openssl rsa −pubout`.

get_private_key_string
: Return the Base64/DER−encoded PKCS1 representation of the private key. This string has header and footer lines:

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

encrypt
: Encrypt a binary ‘‘string’’ using the public (portion of the) key.

decrypt
: Decrypt a binary ‘‘string’’. Croaks if the key is public only.

private_encrypt
: Encrypt a binary ‘‘string’’ using the private key. Croaks if the key is public only.

public_decrypt
: Decrypt a binary ‘‘string’’ using the public (portion of the) key.

sign
: Sign a string using the secret (portion of the) key.

verify
: Check the signature on a text.

use_no_padding
: Use raw RSA encryption. This mode should only be used to implement cryptographically sound padding modes in the application code. Encrypting user data directly with RSA is insecure.

use_pkcs1_padding
: Use PKCS #1 v1.5 padding. This currently is the most widely used mode of padding.

use_pkcs1_oaep_padding
: Use `EME-OAEP` padding as defined in PKCS #1 v2.0 with SHA-1, MGF1 and an empty encoding parameter. This mode of padding is recommended for all new applications. It is the default mode used by `Crypt::OpenSSL::RSA`.

use_sslv23_padding
: Use `PKCS #1 v1.5` padding with an SSL-specific modification that denotes that the server is SSL3 capable.

use_md5_hash
: Use the RFC 1321 MD5 hashing algorithm by Ron Rivest when signing and verifying messages.

  Note that this is considered **insecure**.

use_sha1_hash
: Use the RFC 3174 Secure Hashing Algorithm (FIPS 180-1) when signing and verifying messages. This is the default, when use_sha256_hash is not available.

use_sha224_hash, use_sha256_hash, use_sha384_hash, use_sha512_hash
: These FIPS 180-2 hash algorithms, for use when signing and verifying messages, are only available with newer openssl versions (>= 0.9.8).

  use_sha256_hash is the default hash mode when available.

use_ripemd160_hash
: Dobbertin, Bosselaers and Preneel's RIPEMD hashing algorithm when signing and verifying messages.

use_whirlpool_hash
: Vincent Rijmen und Paulo S. L. M. Barreto ISO/IEC 10118-3:2004 WHIRLPOOL hashing algorithm when signing and verifying messages.

size
: Returns the size, in bytes, of the key. All encrypted text will be of this size, and depending on the padding mode used, the length of the text to be encrypted should be:

  pkcs1_oaep_padding
  : at most 42 bytes less than this size.

  pkcs1_padding or sslv23_padding
  : at most 11 bytes less than this size.

  no_padding
  : exactly this size.

check_key
: This function validates the RSA key, returning a true value if the key is valid, and a false value otherwise. Croaks if the key is public only.

get_key_parameters
: Return `Crypt::OpenSSL::Bignum` objects representing the values of `n`, `e`, `d`, `p`, `q`, `d mod (p-1)`, `d mod (q-1)`, and `1/q mod p`, where `p` and `q` are the prime factors of `n`, `e` is the public exponent and `d` is the private exponent. Some of these values may return as `undef`; only `n` and `e` will be defined for a public key. The`Crypt::OpenSSL::Bignum` module must be installed for this to work.

is_private
: Return true if this is a private key, and false if it is private only.

BUGS

There is a small memory leak when generating new keys of more than 512 bits.

AUTHOR

Ian Robertson, `iroberts@cpan.org`. For support, please email `perl-openssl-users@lists.sourceforge.net`.

ACKNOWLEDGEMENTS

LICENSE

SEE ALSO

perl (1), Crypt::OpenSSL::Random, Crypt::OpenSSL::Bignum, **rsa** (3), **RSA_new** (3) <http://man.he.net/?topic=RSA_new&section=3>, **RSA_public_encrypt** (3) <http://man.he.net/?topic=RSA_public_encrypt&section=3>, **RSA_size** (3) <http://man.he.net/?topic=RSA_size&section=3>, **RSA_generate_key** (3) <http://man.he.net/?topic=RSA_generate_key&section=3>, **RSA_check_key** (3) <http://man.he.net/?topic=RSA_check_key&section=3>