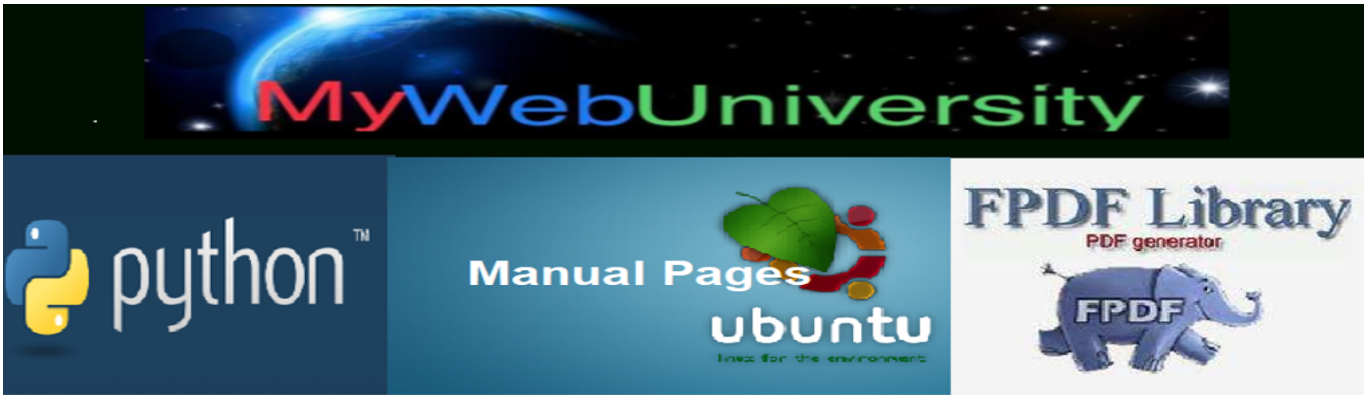

Rocky Enterprise Linux 9.2 Manual Pages on command 'Crypt::PK::RSA.3pm'

$ man Crypt::PK::RSA.3pm

Crypt::PK::RSA(3pm) User Contributed Perl Documentation Crypt::PK::RSA(3pm)

NAME

Crypt::PK::RSA - Public key cryptography based on RSA

SYNOPSIS

```
### OO interface
#Encryption: Alice
my $pub = Crypt::PK::RSA->new('Bob_pub_rsa1.der');
my $ct = $pub->encrypt("secret message");
#
#Encryption: Bob (received ciphertext $ct)
my $priv = Crypt::PK::RSA->new('Bob_priv_rsa1.der');
my $pt = $priv->decrypt($ct);
#Signature: Alice
my $priv = Crypt::PK::RSA->new('Alice_priv_rsa1.der');
my $sig = $priv->sign_message($message);
#
#Signature: Bob (received $message + $sig)
my $pub = Crypt::PK::RSA->new('Alice_pub_rsa1.der');
$pub->verify_message($sig, $message) or die "ERROR";
#Key generation
my $pk = Crypt::PK::RSA->new();
$pk->generate_key(256, 65537);
my $private_der = $pk->export_key_der('private');
```

```
my $public_der = $pk->export_key_der('public');
my $private_pem = $pk->export_key_pem('private');
my $public_pem = $pk->export_key_pem('public');
### Functional interface
#Encryption: Alice
my $ct = rsa_encrypt('Bob_pub_rsa1.der', "secret message");
#Encryption: Bob (received ciphertext $ct)
my $pt = rsa_decrypt('Bob_priv_rsa1.der', $ct);
#Signature: Alice
my $sig = rsa_sign_message('Alice_priv_rsa1.der', $message);
#Signature: Bob (received $message + $sig)
rsa_verify_message('Alice_pub_rsa1.der', $sig, $message) or die "ERROR";
```

DESCRIPTION

The module provides a full featured RSA implementation.

METHODS

new

```
my $pk = Crypt::PK::RSA->new();
#or
my $pk = Crypt::PK::RSA->new($priv_or_pub_key_filename);
#or
my $pk = Crypt::PK::RSA->new(\$buffer_containing_priv_or_pub_key);
```

Support for password protected PEM keys

```
my $pk = Crypt::PK::RSA->new($priv_pem_key_filename, $password);
#or
my $pk = Crypt::PK::RSA->new(\$buffer_containing_priv_pem_key, $password);
```

generate_key

Uses Yarrow-based cryptographically strong random number generator seeded with random data taken from "/dev/random" (UNIX) or "CryptGenRandom" (Win32).

```
$pk->generate_key($size, $e);
# $size .. key size: 128-512 bytes (DEFAULT is 256)
# $e ..... exponent: 3, 17, 257 or 65537 (DEFAULT is 65537)
```

import_key

Loads private or public key in DER or PEM format.

```
  $pk->import_key($priv_or_pub_key_filename);
  #or
  $pk->import_key(\$buffer_containing_priv_or_pub_key);
Support for password protected PEM keys
  $pk->import_key($pem_filename, $password);
  #or
  $pk->import_key(\$buffer_containing_pem_key, $password);
Loading private or public keys form perl hash:
 $pk->import_key($hashref);
 # the $hashref is either a key exported via key2hash
 $pk->import_key({
   e  => "10001", #public exponent
   d  => "9ED5C3D3F866E06957CA0E9478A273C39BBDA4EEAC5B...", #private exponent
   N  => "D0A5CCCAE03DF9C2F5C4C8C0CE840D62CDE279990DC6...", #modulus
   p  => "D3EF0028FFAB508E2773C659E428A80FB0E9211346B4...", #p factor of N
   q  => "FC07E46B163CAB6A83B8E467D169534B2077DCDEECAE...", #q factor of N
   qP => "88C6D406F833DF73C8B734548E0385261AD51F4187CF...", #1/q mod p CRT param
   dP => "486F142FEF0A1F53269AC43D2EE4D263E2841B60DA36...", #d mod (p - 1) CRT param
   dQ => "4597284B2968B72C4212DB7E8F24360B987B80514DA9...", #d mod (q - 1) CRT param
 });
 # or a hash with items corresponding to JWK (JSON Web Key)
 $pk->import_key({
 {
   kty => "RSA",
                                                                  n                    =>
"0vx7agoebGcQSuuPiLJXZpt...eZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqbw0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
   e   => "AQAB",
                                                                  d                    =>
"X4cTteJY_gn4FYPsXB8rdXi...FLN5EEaG6RoVH-HLKD9Mdx5ooGURknhnrRwUkC7h5fJLMWbFAKLWY2v7B6NqSzUvx0
_YSf",
                                                                  p                    =>
"83i-7IvMGXoMXCskv73TKr8...Z27zvoj6pbUQyLPBQxtPnwD20-60eTmD2ujMt5PoMrm8RmNhVWtjjMmMjOpSicFHjXOuVI"
,
```

q =>
"3dfOR9cuYq-0S-mkFLzgItg...q3hWeMuG0ouqnb3obLyuqjVZQ1dIrdgTnCdYzBcOW5r37AFXjift_NGiovonzhKpoVVS78",
dp =>
"G4sPXkc6Ya9y8oJW9_ILj4...zi_H7TkS8x5SdX3oE0oiYwxIiemTAu0UOa5pgFGyJ4c8t2VF40XRugKTP8akhFo5tA77Qe",
dq =>
"s9lAH9fggBsoFR8Oac2R_E...T2kGOhvIllTE1efA6huUvMfBcpn8lqW6vzzYY5SSF7pMd_agI3G8IbpBUb0JiraRNUfLhc",
qi =>
"GyM_p6JrXySiz1toFgKbWV...4ypu9bMWx3QJBfm0FoYzUIZEVEcOqwmRN81oDAaaBk0KWGDjJHDdDmFW3AN7I-pux_
mHZG",
});

Supported key formats:

```
# all formats can be loaded from a file
my $pk = Crypt::PK::RSA->new($filename);
# or from a buffer containing the key
my $pk = Crypt::PK::RSA->new(\$buffer_with_key);
```

? RSA public keys

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDHlYKg9DeHB3/dY1D9WCyJTnl5
vEzAXpUOL9tDtdPUl96brIbbdMLooO1hKjsq98kLs1q4vOn/pxvzk0BRwhiu7Vvb
VUjAn/2HHDDL0U1utqqlMJhaffeLI3HEq5o/lSMFY7sSkZU/E4YX1yqAN0SE7xfK
B2uzcNq60sMIfp6siQIDAQAB
-----END PUBLIC KEY-----
```

? RSA private keys

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDHlYKg9DeHB3/dY1D9WCyJTnl5vEzAXpUOL9tDtdPUl96brIbb
dMLooO1hKjsq98kLs1q4vOn/pxvzk0BRwhiu7VvbVUjAn/2HHDDL0U1utqqlMJha
ffeLI3HEq5o/lSMFY7sSkZU/E4YX1yqAN0SE7xfKB2uzcNq60sMIfp6siQIDAQAB
AoGBAI5+GgNcGQDYw9uF+t7FwxZM5sGZRJrbbEPyuvL+sDxKKW6voKCyHi4EJzaF
9jRZMDqgVJcsmUwjPPuMGBHHJ+MI5Zb3L0jbZkyx8u+U5gf88oy9eZmfGOjmHcMB
oCgzyoLmJETuyADg2onLanuY3jggFb3tq/jimKjO8xM2R6zhAkEA7uXWWyJI9cCN
zrVt5R5v6oosjZ4r5VILGMqBRLrzfTvH+WDMK6Rl/2MHE+YDeLajzunaM8qY2456
GTYEXQsIdQJBANXfMEtXocSdPtoVj3ME8Do/0r+ApgTdcDPCwXOzkmkEJW/UFMSn
b8CYF5G6sZQN9L5z3s2nvi55PaFV8Q0LMUUCQBh9GvIQm6YFbQPpeTBpZFOIgnSp
```

6BoDxPtvlryy5U7LF/6qO4OlwIbjYdBaXbS8FCKbujBg7jZjboSzEtNu1BkCQDGT

w0Yz0jQZn3A+fzpScr2N/fSWheWqz0+wXdfMUKw3YdZCe236wlUK7KvDc1a2xX1A

ru1NbTCoujikC3TSm2ECQQDKQshchJlZJmFv9vCFQlGCA/EX+4406xvOOiixbPYC

pIB4Ee2cmvEdAqSaOjrvgs5zvaCCFBO0MecPStCAxUX6

-----END RSA PRIVATE KEY-----

? RSA private keys in password protected PEM format

-----BEGIN RSA PRIVATE KEY-----

Proc-Type: 4,ENCRYPTED

DEK-Info: DES-EDE3-CBC,4D697440FF5AEF18

C09H49Gn99o8b8O2r4+Hqao4r3udvC+QSSfsk20sXatyuZSEmbhyqKAB+13NRj+3

KIsRTqnL9VkeibIGgLHuekOFKAqeSVZ0PmR4bGWEFxUPAYUvg9N9pIa6hGtNZG+y

TEpOAfFITb1pbHQhp3j8y7qmKc5kY5LrZSFE8WwA24NTG773E07wJgRxKDkXNGOl

kki6oYArNEps0DdtHFxzgdRg0+yaotXuFJRuC5V4YzKGG/oSRcgYyXKTwCndb3xt

aHgI2WprQAPg+qOpLABzoi7bEjCqbHWrwkvnAngylbim2Uyvw1e1xKnzlgIHU7pv

e/J+s00pTItfqW1IpY2mh4C9nkfkfVKBKaAv7jO0s6aPySATqsdlrzv2kpF6Ub4J

kgaZDOfZ4K3qkyAYVLWcQeDqg4glv9Ah2J05bTm4qrIMmthYnThyQlGvcjUfCMXs

0t+mEQbsRY7xKt0o6HzzvQlJ+JsFlLORoslAubJX9iLqpEdnlrj1lD9bo6uIClZ5

5+aoLcAyz1D4OsauuP5i8VFu+Is+QG4SN/vHVuArjkqi3VpLwSAjNDY+KWbq042l

CqlM2mwm6FIGUZQFxiLHJD7WDmk1xmae++m+XG9CEDTfrUQ5v+l0O6BTrl80XUfU

w3gzAWbSjz3UK0FpKeABVFPE9fjNP9fTcS6qL5YJWBPflwxCAbVgsBOW4bOMpDGK

BJDQTeShWn4BlYCe/vgThI9ERdgZhRz4NcFeDgVA/CqQzVqptvz4PSqH46fqUN2n

4PtJgKE5cASYUBuAjlD71FecSVVM/OTzL1uxYzXBilzvVn2vSHgo9g==

-----END RSA PRIVATE KEY-----

? PKCS#8 encoded private keys

-----BEGIN PRIVATE KEY-----

MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBANPN17xW4EkH5PXG

1i/i3rE1EXFcCHyxmz95VRBDs1p3MuYf9mxntbfYAmuzS3KrRWh3IyX/Eh80N/v9

OXPlwZbVqSTX+L3pCEJtRtsWn0zmswGThjMZiwle0oWuap63L35F1QN8EDaSPSBC

yGELNRr6rwVYq0w5b+LOcaCZ+/H1AgMBAAECgYEApfu3aGpww+rC3HUhX0+ckyTy

cXLdV9LbxidwqRlVEb0+DyfXNucjelp2sy5EHy3na9GJovo8mmWSxhCRGKliRkQ6

XgrEMZdCSaWI2AazuHAGlUJRFEVkvdla3AuBAn6y0YdDp/3kbg0yahmKyD8Gq74z

nUYbDL3R5JtR2Ad/KlUCQQDvSEICTHbO/BF7hVmlKRYZSNHKEPrv8X/OlppS14Kv

QRwc+CZ5+l6T1Y+l5cHJQUXrXZoWS1K741TXdUhjjUd7AkEA4pod804Ex8sttdWi

pHMfeyj+IbPAk5XnBc91jT7AYIeL8ccjtfl99xhMsGFaxrh3wA/4SGEvwzWkbxcq

H8G5TwJAKNG+0P2SVwURRm0dOdukdXPCtiHnbP9Zujhe4zr4hEUrMpXymmRntfh8

pORpBpgoAVraams3Fe5WDttnGfSD+QJAOOC6V9HjfUrQhG3FT0XeRwm5EDiQQ/tC

a8DxHqz7mL8tL1ju68ReC+G7jiJBqNOwqzLW/UP3uyYByiikWChGHQJAHUau7jIM

45ErO096n94Vh95p76ANxOroWszOt39TyvJOyklfoPwFagLrBWV9Jjos2/D54KE+

fyoy4t3yHT+/nw==

-----END PRIVATE KEY-----

? PKCS#8 encrypted private keys - password protected keys (supported since: CryptX-0.062)

-----BEGIN ENCRYPTED PRIVATE KEY-----

MIICojAcBgoqhkiG9w0BDAEDMA4ECCQk+Rr1yzzcAgIIAASCAoD/mgpUFjxxM/Ty

Yt+NeT0Fo4echgoGksqs6+rYhO16oshG664emZfkuNoFGGzJ38X6GVuqIXhlPnYQ

biKvL37dN/KnoGytFHq9Wnk8dDwjGHPtwajhW5WuIV3NuhW/AO1PF/cRZKFjWrPt

NWY5CrpfH6t6zojoe+5uyXpH29lQy4OqvSRdPIt/12UcB+tzV7XzSWEuXh8HAi8a

sYUu6tuCFnq4GrD2ffM4KWFmL5GqBAwN6m0KkyrNni9XT+RaA6zEhv/lVcwg2esa

4/EzRs0ixzzZDKaml8oCMl9RHtFAbQmdlfV7Ip4rGK9BwY6UFiDMIVru6HynOVQK

vvZ+j//bgO+3ubrv7psX+vC9Fy/MoH2Tc7MIwDN/QVTciPZlzjWBnBNxMfeFKtEn

d7NFiapgfLuRQIiDTMrW/clcqvO54NphxhrcgUEoxos4twKZARntqPZHtf8nEM2x

2sEF5kI65aEF/5Yy16qvP0vZAA2B1kcIdXZ8XLZCp4c3olhkIrmgUpo1gyFXdCoC

7dT5Cz7/YLkq5hkcFrtp4V9BZMR24fSttc4p24N5xuZ+JneGnGkLX6B+nJAtm9vw

bZA6P+23GI0qeMzL3HJXwCOTSsWfm/H9W5+2Zmw851aAmE+pZLni/pk3e3iNSWgs

946x/doA5O0uCFsU7oxme+WAIp2SjhxGoe808Lf1CCFMPboFi1O/E0NsX8SIEX+i

U+UHi4kxZqVkr3Q5SB/9kiSv8K1bE787yueQOT/dsTYYaMsjAbkEZo0o/47F32T6

A2ioXHOV/pr5zNHqE5tL+qKEcLYbAUF1O+WvmdqYz+vHQjRQBatAqTmncvLDYr/j

1HPwZX2d

-----END ENCRYPTED PRIVATE KEY-----

? RSA public key from X509 certificate

-----BEGIN CERTIFICATE-----

MIIC8zCCAdugAwIBAgIJAPi+LvMU3uGWMA0GCSqGSIb3DQEBCwUAMBAxDjAMBgNV

BAMMBXBva3VzMB4XDTE3MDcxNDE0MTAyMFoXDTIwMDQwOTE0MTAyMFowEDEOMAwG

A1UEAwwFcG9rdXMwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDCQima

SUIMIdz5uVevzcScbcj06xs1OLaFKUoPJ8v+xP6Ut61BQhAvc8GYuw2uRx223hZC

r3HYLfSdWIfmOIAtlL8cPYPVoSivJtpSGE6fBG1tlBjVgXWRmJGR/oxx6Y5QDwcB

Q4GZKga8TtHQoY5idZuatYOFZGfMIcIUC0Uoda+YSypnw7A90F/JvlpcTUh3Fnem

VinqEA6XOegU9dCZk/29sXqauBjbdGihh8DvpklOhY16eQoiR3909AywQ0KUmI+R

Sa9E8oIsmUDetFuXEvana+sD3y42tU+cd2nhBPRETbSXPcum0B3uF4yKgweuJy5D

cvtVQIFVkkh4+AWNAgMBAAGjUDBOMB0GA1UdDgQWBBSS6V5PVGyN92NoB0AVLcOb

pzR3SzAfBgNVHSMEGDAWgBSS6V5PVGyN92NoB0AVLcObpzR3SzAMBgNVHRMEBTAD

AQH/MA0GCSqGSIb3DQEBCwUAA4IBAQBIszrBjoJ39axsS6Btbvwvo8vAmgiSWsav

7AmjXOAwknHPaCcDmrdOys5POD0DNRwNeRsnxFiZ/UL8Vmj2JGDLgAw+/v32MwfX

Ig7m+oIbO8KqDzlYvS5kd3suJ5C21hHy1/JUtfofZLovZH7ZRzhTAoRvCYaodW90

2o8ZqmyCdcXPzjFmoJ2xYzs/Sf8/E1cHfb+4HjOpeRnKxDvG0gwWzcsXpUrw2pNO

Oztj6Rd0THNrf/anIeYVtAHX4aqZA8Kbv2TyJd+9g78usFw1cn+8vfmilm6Pn0DQ

a+I5GyGd7BJI8wYuWqIStzvrJHbQQaNrSk7hgjWYiYlcsPh6w2QP

-----END CERTIFICATE-----

? SSH public RSA keys

ssh-rsa AAAAB3NzaC1yc2EAAAADAQA...6mdYs5iJNGu/ltUdc=

? SSH public RSA keys (RFC-4716 format)

---- BEGIN SSH2 PUBLIC KEY ----

Comment: "768-bit RSA, converted from OpenSSH"

AAAAB3NzaC1yc2EAAAADAQABAAAAYQDYebeGQFCnlQiNRE7r9UEbjr+DQMTdw1ZHGB2w6x

D/DzKem8761GdCpqsLrGaw2D7aSIoP1B5Sz870YoVWHn6Ao7Hvm17V3Kxfn4B01GNQTM5+

L26mdYs5iJNGu/ltUdc=

---- END SSH2 PUBLIC KEY ----

? RSA private keys in JSON Web Key (JWK) format

See <http://tools.ietf.org/html/draft-ietf-jose-json-web-key>

{

"kty":"RSA",

"n":"0vx7agoebGcQSuuPiLJXZpt...eZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqbw0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw

",

"e":"AQAB",

"d":"X4cTteJY_gn4FYPsXB8rdXi...FLN5EEaG6RoVH-HLKD9Mdx5ooGURknhnrRwUkC7h5fJLMWbFAKLWY2v7B6NqSzUv

x0_YSf",

"p":"83i-7IvMGXoMXCskv73TKr8...Z27zvoj6pbUQyLPBQxtPnwD20-60eTmD2ujMt5PoMrm8RmNhVWtjjMmMjOpSicFHjXOuVI",

"q":"3dfOR9cuYq-0S-mkFLzgItg...q3hWeMuG0ouqnb3obLyuqjVZQ1dIrdgTnCdYzBcOW5r37AFXjift_NGiovonzhKpoVVS78",

"dp":"G4sPXkc6Ya9y8oJW9_ILj4...zi_H7TkS8x5SdX3oE0oiYwxIiemTAu0UOa5pgFGyJ4c8t2VF40XRugKTP8akhFo5tA77Qe",

"dq":"s9lAH9fggBsoFR8Oac2R_E...T2kGOhvIllTE1efA6huUvMfBcpn8lqW6vzzYY5SSF7pMd_agI3G8IbpBUb0JiraRNUfLhc",

"qi":"GyM_p6JrXySiz1toFgKbWV...4ypu9bMWx3QJBfm0FoYzUIZEVEcOqwmRN81oDAaaBk0KWGDjJHDdDmFW3AN7I-pux_mHZG",

}

BEWARE: For JWK support you need to have JSON module installed.

? RSA public keys in JSON Web Key (JWK) format

{

"kty":"RSA",

"n":"0vx7agoebGcQSuuPiLJXZp...tN9nndrQmbXEps2aiAFbWhM78LhWx4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECP",

"e":"AQAB",

}

BEWARE: For JWK support you need to have JSON module installed.

export_key_der

my $private_der = $pk->export_key_der('private');

#or

my $public_der = $pk->export_key_der('public');

export_key_pem

my $private_pem = $pk->export_key_pem('private');

#or

my $public_pem = $pk->export_key_pem('public');

```
 #or
 my $public_pem = $pk->export_key_pem('public_x509');
```

With parameter 'public' uses header and footer lines:

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

With parameter 'public_x509' uses header and footer lines:

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

Support for password protected PEM keys

```
 my $private_pem = $pk->export_key_pem('private', $password);
 #or
 my $private_pem = $pk->export_key_pem('private', $password, $cipher);
 # supported ciphers: 'DES-CBC'
 #                    'DES-EDE3-CBC'
 #                    'SEED-CBC'
 #                    'CAMELLIA-128-CBC'
 #                    'CAMELLIA-192-CBC'
 #                    'CAMELLIA-256-CBC'
 #                    'AES-128-CBC'
 #                    'AES-192-CBC'
 #                    'AES-256-CBC' (DEFAULT)
```

export_key_jwk

Since: CryptX-0.022

Exports public/private keys as a JSON Web Key (JWK).

```
 my $private_json_text = $pk->export_key_jwk('private');
 #or
 my $public_json_text = $pk->export_key_jwk('public');
```

Also exports public/private keys as a perl HASH with JWK structure.

```
 my $jwk_hash = $pk->export_key_jwk('private', 1);
 #or
 my $jwk_hash = $pk->export_key_jwk('public', 1);
```

BEWARE: For JWK support you need to have JSON module installed.

export_key_jwk_thumbprint

Since: CryptX-0.031

Exports the key's JSON Web Key Thumbprint as a string.

If you don't know what this is, see RFC 7638 <https://tools.ietf.org/html/rfc7638>.

```
 my $thumbprint = $pk->export_key_jwk_thumbprint('SHA256');
```

encrypt

```
 my $pk = Crypt::PK::RSA->new($pub_key_filename);
 my $ct = $pk->encrypt($message);
 #or
 my $ct = $pk->encrypt($message, $padding);
 #or
 my $ct = $pk->encrypt($message, 'oaep', $hash_name, $lparam);
 # $padding ..... 'oaep' (DEFAULT), 'v1.5' or 'none' (INSECURE)
 # $hash_name (only for oaep) .. 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
 # $lparam (only for oaep) ..... DEFAULT is empty string
```

decrypt

```
 my $pk = Crypt::PK::RSA->new($priv_key_filename);
 my $pt = $pk->decrypt($ciphertext);
 #or
 my $pt = $pk->decrypt($ciphertext, $padding);
 #or
 my $pt = $pk->decrypt($ciphertext, 'oaep', $hash_name, $lparam);
 # $padding ..... 'oaep' (DEFAULT), 'v1.5' or 'none' (INSECURE)
 # $hash_name (only for oaep) .. 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
 # $lparam (only for oaep) ..... DEFAULT is empty string
```

sign_message

```
 my $pk = Crypt::PK::RSA->new($priv_key_filename);
 my $signature = $priv->sign_message($message);
 #or
 my $signature = $priv->sign_message($message, $hash_name);
 #or
 my $signature = $priv->sign_message($message, $hash_name, $padding);
 #or
 my $signature = $priv->sign_message($message, $hash_name, 'pss', $saltlen);
```

```
    # $hash_name ..... 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
    # $padding ..... 'pss' (DEFAULT) or 'v1.5' or 'none' (INSECURE)
    # $saltlen (only for pss) .. DEFAULT is 12
verify_message
    my $pk = Crypt::PK::RSA->new($pub_key_filename);
    my $valid = $pub->verify_message($signature, $message);
    #or
    my $valid = $pub->verify_message($signature, $message, $hash_name);
    #or
    my $valid = $pub->verify_message($signature, $message, $hash_name, $padding);
    #or
    my $valid = $pub->verify_message($signature, $message, $hash_name, 'pss', $saltlen);
    # $hash_name ..... 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
    # $padding ..... 'pss' (DEFAULT) or 'v1.5' or 'none' (INSECURE)
    # $saltlen (only for pss) .. DEFAULT is 12
sign_hash
    my $pk = Crypt::PK::RSA->new($priv_key_filename);
    my $signature = $priv->sign_hash($message_hash);
    #or
    my $signature = $priv->sign_hash($message_hash, $hash_name);
    #or
    my $signature = $priv->sign_hash($message_hash, $hash_name, $padding);
    #or
    my $signature = $priv->sign_hash($message_hash, $hash_name, 'pss', $saltlen);
    # $hash_name ..... 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
    # $padding ..... 'pss' (DEFAULT) or 'v1.5' or 'none' (INSECURE)
    # $saltlen (only for pss) .. DEFAULT is 12
verify_hash
    my $pk = Crypt::PK::RSA->new($pub_key_filename);
    my $valid = $pub->verify_hash($signature, $message_hash);
    #or
    my $valid = $pub->verify_hash($signature, $message_hash, $hash_name);
    #or
```

```
    my $valid = $pub->verify_hash($signature, $message_hash, $hash_name, $padding);
    #or
    my $valid = $pub->verify_hash($signature, $message_hash, $hash_name, 'pss', $saltlen);
    # $hash_name ..... 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
    # $padding ..... 'pss' (DEFAULT) or 'v1.5' or 'none' (INSECURE)
    # $saltlen (only for pss) .. DEFAULT is 12
  is_private
    my $rv = $pk->is_private;
    # 1 .. private key loaded
    # 0 .. public key loaded
    # undef .. no key loaded
  size
    my $size = $pk->size;
    # returns key size in bytes or undef if no key loaded
  key2hash
    my $hash = $pk->key2hash;
    # returns hash like this (or undef if no key loaded):
    {
      type => 1,   # integer: 1 .. private, 0 .. public
      size => 256, # integer: key size in bytes
      # all the rest are hex strings
      e  => "10001", #public exponent
      d  => "9ED5C3D3F866E06957CA0E9478A273C39BBDA4EEAC5B...", #private exponent
      N  => "D0A5CCCAE03DF9C2F5C4C8C0CE840D62CDE279990DC6...", #modulus
      p  => "D3EF0028FFAB508E2773C659E428A80FB0E9211346B4...", #p factor of N
      q  => "FC07E46B163CAB6A83B8E467D169534B2077DCDEECAE...", #q factor of N
      qP => "88C6D406F833DF73C8B734548E0385261AD51F4187CF...", #1/q mod p CRT param
      dP => "486F142FEF0A1F53269AC43D2EE4D263E2841B60DA36...", #d mod (p - 1) CRT param
      dQ => "4597284B2968B72C4212DB7E8F24360B987B80514DA9...", #d mod (q - 1) CRT param
    }
FUNCTIONS
  rsa_encrypt
    RSA based encryption. See method "encrypt" below.
```

```
    my $ct = rsa_encrypt($pub_key_filename, $message);
    #or
    my $ct = rsa_encrypt(\$buffer_containing_pub_key, $message);
    #or
    my $ct = rsa_encrypt($pub_key, $message, $padding);
    #or
    my $ct = rsa_encrypt($pub_key, $message, 'oaep', $hash_name, $lparam);
    # $padding ..... 'oaep' (DEFAULT), 'v1.5' or 'none' (INSECURE)
    # $hash_name (only for oaep) .. 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
    # $lparam (only for oaep) ..... DEFAULT is empty string
rsa_decrypt
   RSA based decryption. See method "decrypt" below.
    my $pt = rsa_decrypt($priv_key_filename, $ciphertext);
    #or
    my $pt = rsa_decrypt(\$buffer_containing_priv_key, $ciphertext);
    #or
    my $pt = rsa_decrypt($priv_key, $ciphertext, $padding);
    #or
    my $pt = rsa_decrypt($priv_key, $ciphertext, 'oaep', $hash_name, $lparam);
    # $padding ..... 'oaep' (DEFAULT), 'v1.5' or 'none' (INSECURE)
    # $hash_name (only for oaep) .. 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
    # $lparam (only for oaep) ..... DEFAULT is empty string
rsa_sign_message
   Generate RSA signature. See method "sign_message" below.
    my $sig = rsa_sign_message($priv_key_filename, $message);
    #or
    my $sig = rsa_sign_message(\$buffer_containing_priv_key, $message);
    #or
    my $sig = rsa_sign_message($priv_key, $message, $hash_name);
    #or
    my $sig = rsa_sign_message($priv_key, $message, $hash_name, $padding);
    #or
    my $sig = rsa_sign_message($priv_key, $message, $hash_name, 'pss', $saltlen);
```

```
    # $hash_name ..... 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
    # $padding ..... 'pss' (DEFAULT) or 'v1.5' or 'none' (INSECURE)
    # $saltlen (only for pss) .. DEFAULT is 12
```

rsa_verify_message

Verify RSA signature. See method "verify_message" below.

```
    rsa_verify_message($pub_key_filename, $signature, $message) or die "ERROR";
    #or
    rsa_verify_message(\$buffer_containing_pub_key, $signature, $message) or die "ERROR";
    #or
    rsa_verify_message($pub_key, $signature, $message, $hash_name) or die "ERROR";
    #or
    rsa_verify_message($pub_key, $signature, $message, $hash_name, $padding) or die "ERROR";
    #or
    rsa_verify_message($pub_key, $signature, $message, $hash_name, 'pss', $saltlen) or die "ERROR";
    # $hash_name ..... 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
    # $padding ..... 'pss' (DEFAULT) or 'v1.5' or 'none' (INSECURE)
    # $saltlen (only for pss) .. DEFAULT is 12
```

rsa_sign_hash

Generate RSA signature. See method "sign_hash" below.

```
    my $sig = rsa_sign_hash($priv_key_filename, $message_hash);
    #or
    my $sig = rsa_sign_hash(\$buffer_containing_priv_key, $message_hash);
    #or
    my $sig = rsa_sign_hash($priv_key, $message_hash, $hash_name);
    #or
    my $sig = rsa_sign_hash($priv_key, $message_hash, $hash_name, $padding);
    #or
    my $sig = rsa_sign_hash($priv_key, $message_hash, $hash_name, 'pss', $saltlen);
    # $hash_name ..... 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
    # $padding ..... 'pss' (DEFAULT) or 'v1.5' or 'none' (INSECURE)
    # $saltlen (only for pss) .. DEFAULT is 12
```

rsa_verify_hash

Verify RSA signature. See method "verify_hash" below.

```
rsa_verify_hash($pub_key_filename, $signature, $message_hash) or die "ERROR";
#or
rsa_verify_hash(\$buffer_containing_pub_key, $signature, $message_hash) or die "ERROR";
#or
rsa_verify_hash($pub_key, $signature, $message_hash, $hash_name) or die "ERROR";
#or
rsa_verify_hash($pub_key, $signature, $message_hash, $hash_name, $padding) or die "ERROR";
#or
rsa_verify_hash($pub_key, $signature, $message_hash, $hash_name, 'pss', $saltlen) or die "ERROR";
# $hash_name ..... 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
# $padding ..... 'pss' (DEFAULT) or 'v1.5' or 'none' (INSECURE)
# $saltlen (only for pss) .. DEFAULT is 12
```

OpenSSL interoperability

```
### let's have:
# RSA private key in PEM format - rsakey.priv.pem
# RSA public key in PEM format  - rsakey.pub.pem
# data file to be signed or encrypted - input.data
```

Encrypt by OpenSSL, decrypt by Crypt::PK::RSA

Create encrypted file (from commandline):

```
openssl rsautl -encrypt -inkey rsakey.pub.pem -pubin -out input.encrypted.rsa -in input.data
```

Decrypt file (Perl code):

```
use Crypt::PK::RSA;
use Crypt::Misc 'read_rawfile';
my $pkrsa = Crypt::PK::RSA->new("rsakey.priv.pem");
my $encfile = read_rawfile("input.encrypted.rsa");
my $plaintext = $pkrsa->decrypt($encfile, 'v1.5');
print $plaintext;
```

Encrypt by Crypt::PK::RSA, decrypt by OpenSSL

Create encrypted file (Perl code):

```
use Crypt::PK::RSA;
use Crypt::Misc 'write_rawfile';
my $plaintext = 'secret message';
my $pkrsa = Crypt::PK::RSA->new("rsakey.pub.pem");
```

```
my $encrypted = $pkrsa->encrypt($plaintext, 'v1.5');
write_rawfile("input.encrypted.rsa", $encrypted);
```

Decrypt file (from commandline):

```
openssl rsautl -decrypt -inkey rsakey.priv.pem -in input.encrypted.rsa
```

Sign by OpenSSL, verify by Crypt::PK::RSA

Create signature (from commandline):

```
openssl dgst -sha1 -sign rsakey.priv.pem -out input.sha1-rsa.sig input.data
```

Verify signature (Perl code):

```
use Crypt::PK::RSA;
use Crypt::Digest 'digest_file';
use Crypt::Misc 'read_rawfile';
my $pkrsa = Crypt::PK::RSA->new("rsakey.pub.pem");
my $signature = read_rawfile("input.sha1-rsa.sig");
my $valid = $pkrsa->verify_hash($signature, digest_file("SHA1", "input.data"), "SHA1", "v1.5");
print $valid ? "SUCCESS" : "FAILURE";
```

Sign by Crypt::PK::RSA, verify by OpenSSL

Create signature (Perl code):

```
use Crypt::PK::RSA;
use Crypt::Digest 'digest_file';
use Crypt::Misc 'write_rawfile';
my $pkrsa = Crypt::PK::RSA->new("rsakey.priv.pem");
my $signature = $pkrsa->sign_hash(digest_file("SHA1", "input.data"), "SHA1", "v1.5");
write_rawfile("input.sha1-rsa.sig", $signature);
```

Verify signature (from commandline):

```
openssl dgst -sha1 -verify rsakey.pub.pem -signature input.sha1-rsa.sig input.data
```

Keys generated by Crypt::PK::RSA

Generate keys (Perl code):

```
use Crypt::PK::RSA;
use Crypt::Misc 'write_rawfile';
my $pkrsa = Crypt::PK::RSA->new;
$pkrsa->generate_key(256, 65537);
write_rawfile("rsakey.pub.der",  $pkrsa->export_key_der('public'));
write_rawfile("rsakey.priv.der", $pkrsa->export_key_der('private'));
```

```
write_rawfile("rsakey.pub.pem",  $pkrsa->export_key_pem('public_x509'));
write_rawfile("rsakey.priv.pem", $pkrsa->export_key_pem('private'));
write_rawfile("rsakey-passwd.priv.pem", $pkrsa->export_key_pem('private', 'secret'));
```

Use keys by OpenSSL:

```
openssl rsa -in rsakey.priv.der -text -inform der
openssl rsa -in rsakey.priv.pem -text
openssl rsa -in rsakey-passwd.priv.pem -text -inform pem -passin pass:secret
openssl rsa -in rsakey.pub.der -pubin -text -inform der
openssl rsa -in rsakey.pub.pem -pubin -text
```

Keys generated by OpenSSL

Generate keys:

```
openssl genrsa -out rsakey.priv.pem 1024
openssl rsa -in rsakey.priv.pem -out rsakey.priv.der -outform der
openssl rsa -in rsakey.priv.pem -out rsakey.pub.pem -pubout
openssl rsa -in rsakey.priv.pem -out rsakey.pub.der -outform der -pubout
openssl rsa -in rsakey.priv.pem -passout pass:secret -des3 -out rsakey-passwd.priv.pem
```

Load keys (Perl code):

```
use Crypt::PK::RSA;
my $pkrsa = Crypt::PK::RSA->new;
$pkrsa->import_key("rsakey.pub.der");
$pkrsa->import_key("rsakey.priv.der");
$pkrsa->import_key("rsakey.pub.pem");
$pkrsa->import_key("rsakey.priv.pem");
$pkrsa->import_key("rsakey-passwd.priv.pem", "secret");
```

SEE ALSO

? <https://en.wikipedia.org/wiki/RSA_%28algorithm%29>