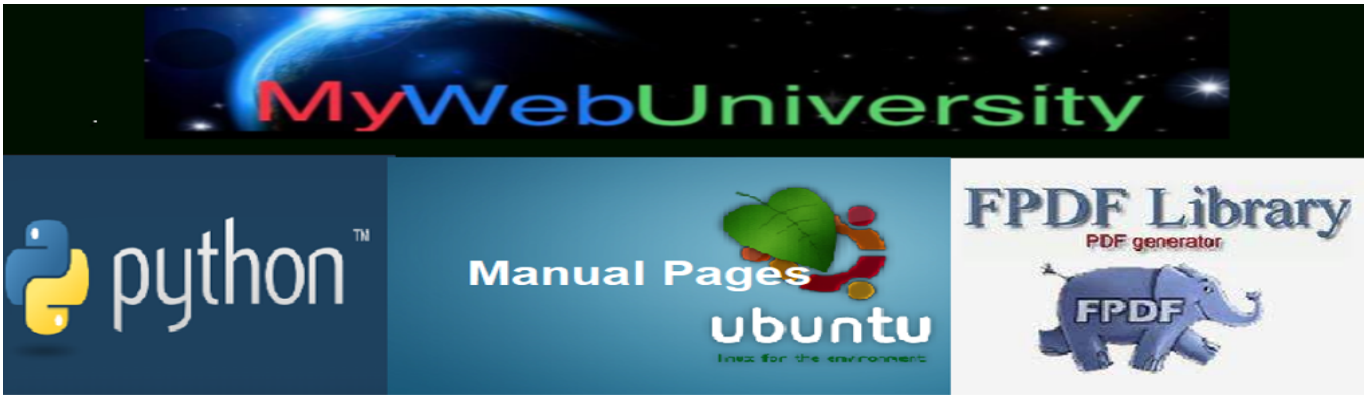

Rocky Enterprise Linux 9.2 Manual Pages on command 'Crypt::PK::DSA.3pm'

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

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

NAME

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

SYNOPSIS

```
### OO interface
#Encryption: Alice
my $pub = Crypt::PK::DSA->new('Bob_pub_dsa1.der');
my $ct = $pub->encrypt("secret message");
#
#Encryption: Bob (received ciphertext $ct)
my $priv = Crypt::PK::DSA->new('Bob_priv_dsa1.der');
my $pt = $priv->decrypt($ct);
#Signature: Alice
my $priv = Crypt::PK::DSA->new('Alice_priv_dsa1.der');
my $sig = $priv->sign_message($message);
#
#Signature: Bob (received $message + $sig)
my $pub = Crypt::PK::DSA->new('Alice_pub_dsa1.der');
$pub->verify_message($sig, $message) or die "ERROR";
#Key generation
my $pk = Crypt::PK::DSA->new();
$pk->generate_key(30, 256);
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 = dsa_encrypt('Bob_pub_dsa1.der', "secret message");
#Encryption: Bob (received ciphertext $ct)
my $pt = dsa_decrypt('Bob_priv_dsa1.der', $ct);
#Signature: Alice
my $sig = dsa_sign_message('Alice_priv_dsa1.der', $message);
#Signature: Bob (received $message + $sig)
dsa_verify_message('Alice_pub_dsa1.der', $sig, $message) or die "ERROR";
```

METHODS

new

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

Support for password protected PEM keys

```
my $pk = Crypt::PK::DSA->new($priv_pem_key_filename, $password);
#or
my $pk = Crypt::PK::DSA->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).

```
$pk->generate_key($group_size, $modulus_size);
# $group_size  ... in bytes .. 15 < $group_size < 1024
# $modulus_size .. in bytes .. ($modulus_size - $group_size) < 512
### Bits of Security according to libtomcrypt documentation
# 80 bits   => generate_key(20, 128)
# 120 bits  => generate_key(30, 256)
# 140 bits  => generate_key(35, 384)
```

```
    # 160 bits  => generate_key(40, 512)
    ### Sizes according section 4.2 of FIPS 186-4
    # (L and N are the bit lengths of p and q respectively)
    # L = 1024, N = 160 => generate_key(20, 128)
    # L = 2048, N = 224 => generate_key(28, 256)
    # L = 2048, N = 256 => generate_key(32, 256)
    # L = 3072, N = 256 => generate_key(32, 384)
    $pk->generate_key($param_hash)
    # $param_hash is { d => $d, p => $p, q => $q }
    # where $d, $p, $q are hex strings
    $pk->generate_key(\$dsa_param)
    # $dsa_param is the content of DER or PEM file with DSA params
    # e.g. openssl dsaparam 2048
```

import_key

Loads private or public key in DER or PEM format.

```
  $pk->import_key($filename);
  #or
  $pk->import_key(\$buffer_containing_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);
  # where $hashref is a key exported via key2hash
  $pk->import_key({
    p => "AAF839A764E04D80824B79FA1F0496C093...", #prime modulus
    q => "D05C4CB45F29D353442F1FEC43A6BE2BE8...", #prime divisor
    g => "847E8896D12C9BF18FE283AE7AD58ED7F3...", #generator of a subgroup of order q in GF(p)
    x => "6C801901AC74E2DC714D75A9F6969483CF...", #private key, random  0 < x < q
    y => "8F7604D77FA62C7539562458A63C7611B7...", #public key, where y = g^x mod p
  });
```

Supported key formats:

? DSA public keys

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

MIIBtjCCASsGByqGSM44BAEwggEeAoGBAJKyu+puNMGLpGIhbD1Iatnwll79ePr4

YHe2KBhRkheKxWUZRpN1Vd/+usS2IHSJ9op5cSWETiP05d7PMtJaitklw7jhudq3

GxNvV/GRdCQm3H6d76FHP88dms4vcDYc6ry6wKERGfNEtZ+4BAKrMZK+gDYsF4Aw

U6WVR969kYZhAhUA6w25FgSRmJ8W4XkvC60n8Wv3DpMCgYA4ZFE+3tLOM24PZj9Z

rxuqUzZZdR+kIzrsIYpWN9ustbmdKLKwsqIaUIxc5zxHEhbAjAIf8toPD+VEQIpY

7vgJgDhXuPq45BgN19iLTzOJwIhAFXPZvnAdIo9D/AnMw688gT6g6U8QCZwX2XYg

ICiVcriYVNcjVKHSFY/X0Oi7CgOBhAACgYB4ZTn4OYT/pjUd6tNhGPtOS3CE1oaj

5ScbetXg4ZDpceEyQi8VG+/ZTbs8var8X77JdEdeQA686cAxpOaVgW8V4odvcmfA

BfueiGnPXjqGfppiHAyL1Ngyd+EsXKmKVXZYAVFVI0WuJKiZBSVURU7+ByxOfpGa

fZhibr0SggWixQ==

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

? DSA private keys

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

MIIBuwIBAAKBgQCSsrvqbjTBi6RiIWw9SGrZ8JSO/Xj6+GB3tigYUZIXisVlGUaT

dVXf/rrEtiB0ifaKeXElhE4j9OXezzLSWorZJcO44bnatxsTb1fxkXQkJtx+ne+h

Rz/PHZrOL3A2HOq8usChERnzRLWfuAQCqzGSvoA2LBeAMFOllUfevZGGYQIVAOsN

uRYEkZifFuF5LwutJ/Fr9w6TAoGAOGRRPt7SzjNuD2Y/Wa8bqlM2WXUfpCM67CGK

VjfbrLW5nSiysLKiGlCMXOc8RxIWwIwCH/LaDw/lRECKWO74CYA4V7j6uOQYDdfY

i08zicCIQBVz2b5wHSKPQ/wJzMOvPIE+oOlPEAmcF9l2ICAolXK4mFTXI1Sh0hWP

19DouwoCgYB4ZTn4OYT/pjUd6tNhGPtOS3CE1oaj5ScbetXg4ZDpceEyQi8VG+/Z

Tbs8var8X77JdEdeQA686cAxpOaVgW8V4odvcmfABfueiGnPXjqGfppiHAyL1Ngy

d+EsXKmKVXZYAVFVI0WuJKiZBSVURU7+ByxOfpGafZhibr0SggWixQIVAL7Sia03

8bvANjjL9Sitk8slrM6P

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

? DSA private keys in password protected PEM format:

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

Proc-Type: 4,ENCRYPTED

DEK-Info: DES-CBC,227ADC3AA0299491

UISxBYAxPQMl2eK9LMAeHsssF6IxO+4G2ta2Jn8VE+boJrrH3iSTKeMXGjGaXl0z

DwcLGV+KMR70y+cxtTb34rFy+uSpBy10dOQJhxALDbe1XfCDQIUfaXRfMNA3um2I

JdZixUD/zcxBOUzao+MCr0V9XlJDgqBhJ5EEr53XHH07Eo5fhiBfbbR9NzdUPFrQ

```
        p2ASyZtFh7RXoIBUCQgg21oeLddcNWV7gd/Y46kghO9s0JbJ8C+IsuWEPRSq502h
        tSoDN6B0sxbVvOUICLLbQaxt7yduTAhRxVIJZ1PWATTVD7CZBVz9uIDZ7LOv+er2
        1q3vkwb8E9spPsA240+BnfD571XEop4jrawxC0VKQZ+3cPVLc6jhIsxvzzFQUt67
        g66v8GUgt7KF3KhVV7qEtntybQWDWb+K/uTIH9Ra8nP820d3Rnl61pPXDPlluteT
        WSLOvEMN2zRmkaxQNv/tLdT0SYpQtdjw74G3A6T7+KnvinKrjtp1a/AXkCF9hNEx
        DGbxOYo1UOmk8qdxWCrab34nO+Q8oQc9wjXHG+ZtRYIMoGMKREK8DeL4H1RPNkMf
        rwXWk8scd8QFmJAb8De1VQ==
        -----END DSA PRIVATE KEY-----
    ?   SSH public DSA keys
        ssh-dss AAAAB3NzaC1kc3MAAACBAKU8/avmk...4XOwuEssAVhmwA==
    ?   SSH public DSA keys (RFC-4716 format)
        ---- BEGIN SSH2 PUBLIC KEY ----
        Comment: "1024-bit DSA, converted from OpenSSH"
        AAAAB3NzaC1kc3MAAACBAKU8/avmkFeGnSqwYG7dZnQlG+01QNaxu3F5v0NcL/SRUW7Idp
        Uq8t14siK0mA6yjphLhOf5t8gugTEVBllP86ANSbFigH7WN3v6ydJWqm60pNhNHN//50cn
        NtIsXbxeq3VtsI64pkH1OJqeZDHLmu73k4T0EKOzsylSfF/wtVBJAAAAFQChpubLHViwPB
        +jSvUb8e4THS7PBQAAAIAJD1PMCiTCQa1xyD/NCWOajCufTOIzKAhm6l+nlBVPiKI+262X
        pYt127Ke4mPL8XJBizoTjSQN08uHMg/8L6W/cdO2aZ+mhkBnS1xAm83DAwqLrDraR1w/4Q
        RFxr5Vbyy8qnejrPjTJobBN1BGsv84wHkjmoCn6pFIfkGYeATlJgAAAIAHYPU1zMVBTDWr
        u7SNC4G2UyWGWYYLjLytBVHfQmBa51CmqrSs2kCfGLGA1ynfYENsxcJq9nsXrb4i17H5BH
        JFkH0g7BUDpeBeLr8gsK3WgfqWwtZsDkltObw9chUD/siK6q/dk/fSIB2Ho0inev7k68Z5
        ZkNI4XOwuEssAVhmwA==
        ---- END SSH2 PUBLIC KEY ----
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 DSA PUBLIC KEY------

-----END DSA 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)
```

encrypt

```
my $pk = Crypt::PK::DSA->new($pub_key_filename);
my $ct = $pk->encrypt($message);
#or
my $ct = $pk->encrypt($message, $hash_name);
#NOTE: $hash_name can be 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
```

decrypt

```
my $pk = Crypt::PK::DSA->new($priv_key_filename);
my $pt = $pk->decrypt($ciphertext);
```

sign_message

```
my $pk = Crypt::PK::DSA->new($priv_key_filename);
my $signature = $priv->sign_message($message);
#or
my $signature = $priv->sign_message($message, $hash_name);
```

```
    #NOTE: $hash_name can be 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
verify_message
    my $pk = Crypt::PK::DSA->new($pub_key_filename);
    my $valid = $pub->verify_message($signature, $message)
    #or
    my $valid = $pub->verify_message($signature, $message, $hash_name);
    #NOTE: $hash_name can be 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
sign_hash
    my $pk = Crypt::PK::DSA->new($priv_key_filename);
    my $signature = $priv->sign_hash($message_hash);
verify_hash
    my $pk = Crypt::PK::DSA->new($pub_key_filename);
    my $valid = $pub->verify_hash($signature, $message_hash);
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 (length of the prime p) in bytes or undef if key not loaded
size_q
    my $size = $pk->size_q;
    # returns length of the prime q in bytes or undef if key not 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
      p => "AAF839A764E04D80824B79FA1F0496C093...", #prime modulus
      q => "D05C4CB45F29D353442F1FEC43A6BE2BE8...", #prime divisor
```

```
    g => "847E8896D12C9BF18FE283AE7AD58ED7F3...", #generator of a subgroup of order q in GF(p)
    x => "6C801901AC74E2DC714D75A9F6969483CF...", #private key, random  0 < x < q
    y => "8F7604D77FA62C7539562458A63C7611B7...", #public key, where y = g^x mod p
  }
```

FUNCTIONS

dsa_encrypt

DSA based encryption as implemented by libtomcrypt. See method "encrypt" below.

```
 my $ct = dsa_encrypt($pub_key_filename, $message);
 #or
 my $ct = dsa_encrypt(\$buffer_containing_pub_key, $message);
 #or
 my $ct = dsa_encrypt($pub_key_filename, $message, $hash_name);
 #NOTE: $hash_name can be 'SHA1' (DEFAULT), 'SHA256' or any other hash supported by Crypt::Digest
```

Encryption works similar to the Crypt::PK::ECC encryption whereas shared DSA key is computed, and the hash of the shared key XOR'ed against the plaintext forms the ciphertext.

dsa_decrypt

DSA based decryption as implemented by libtomcrypt. See method "decrypt" below.

```
 my $pt = dsa_decrypt($priv_key_filename, $ciphertext);
 #or
 my $pt = dsa_decrypt(\$buffer_containing_priv_key, $ciphertext);
```

dsa_sign_message

Generate DSA signature. See method "sign_message" below.

```
 my $sig = dsa_sign_message($priv_key_filename, $message);
 #or
 my $sig = dsa_sign_message(\$buffer_containing_priv_key, $message);
 #or
 my $sig = dsa_sign_message($priv_key, $message, $hash_name);
```

dsa_verify_message

Verify DSA signature. See method "verify_message" below.

```
 dsa_verify_message($pub_key_filename, $signature, $message) or die "ERROR";
 #or
 dsa_verify_message(\$buffer_containing_pub_key, $signature, $message) or die "ERROR";
```

```
#or
dsa_verify_message($pub_key, $signature, $message, $hash_name) or die "ERROR";
```

dsa_sign_hash

Generate DSA signature. See method "sign_hash" below.

```
my $sig = dsa_sign_hash($priv_key_filename, $message_hash);
#or
my $sig = dsa_sign_hash(\$buffer_containing_priv_key, $message_hash);
```

dsa_verify_hash

Verify DSA signature. See method "verify_hash" below.

```
dsa_verify_hash($pub_key_filename, $signature, $message_hash) or die "ERROR";
#or
dsa_verify_hash(\$buffer_containing_pub_key, $signature, $message_hash) or die "ERROR";
```

OpenSSL interoperability

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

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

Create signature (from commandline):

```
openssl dgst -sha1 -sign dsakey.priv.pem -out input.sha1-dsa.sig input.data
```

Verify signature (Perl code):

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

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

Create signature (Perl code):

```
use Crypt::PK::DSA;
use Crypt::Digest 'digest_file';
use Crypt::Misc 'write_rawfile';
```

```
my $pkdsa = Crypt::PK::DSA->new("dsakey.priv.pem");
my $signature = $pkdsa->sign_hash(digest_file("SHA1", "input.data"), "SHA1", "v1.5");
write_rawfile("input.sha1-dsa.sig", $signature);
```

Verify signature (from commandline):

```
openssl dgst -sha1 -verify dsakey.pub.pem -signature input.sha1-dsa.sig input.data
```

Keys generated by Crypt::PK::DSA

Generate keys (Perl code):

```
use Crypt::PK::DSA;
use Crypt::Misc 'write_rawfile';
my $pkdsa = Crypt::PK::DSA->new;
$pkdsa->generate_key(20, 128);
write_rawfile("dsakey.pub.der",  $pkdsa->export_key_der('public'));
write_rawfile("dsakey.priv.der", $pkdsa->export_key_der('private'));
write_rawfile("dsakey.pub.pem",  $pkdsa->export_key_pem('public_x509'));
write_rawfile("dsakey.priv.pem", $pkdsa->export_key_pem('private'));
write_rawfile("dsakey-passwd.priv.pem", $pkdsa->export_key_pem('private', 'secret'));
```

Use keys by OpenSSL:

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

Keys generated by OpenSSL

Generate keys:

```
openssl dsaparam -genkey -out dsakey.priv.pem 1024
openssl dsa -in dsakey.priv.pem -out dsakey.priv.der -outform der
openssl dsa -in dsakey.priv.pem -out dsakey.pub.pem -pubout
openssl dsa -in dsakey.priv.pem -out dsakey.pub.der -outform der -pubout
openssl dsa -in dsakey.priv.pem -passout pass:secret -des3 -out dsakey-passwd.priv.pem
```

Load keys (Perl code):

```
use Crypt::PK::DSA;
my $pkdsa = Crypt::PK::DSA->new;
$pkdsa->import_key("dsakey.pub.der");
```

```
$pkdsa->import_key("dsakey.priv.der");

$pkdsa->import_key("dsakey.pub.pem");

$pkdsa->import_key("dsakey.priv.pem");

$pkdsa->import_key("dsakey-passwd.priv.pem", "secret");
```

SEE ALSO

- ? <https://en.wikipedia.org/wiki/Digital_Signature_Algorithm>