Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'PEM_read_DSA_PUBKEY.3ossl' command

$ man PEM_read_DSA_PUBKEY.3ossl

PEM_READ_BIO_PRIVATEKEY(3ossl) OpenSSL PEM_READ_BIO_PRIVATEKEY(3ossl)

NAME

pem_password_cb, PEM_read_bio_PrivateKey_ex, PEM_read_bio_PrivateKey,

PEM_read_PrivateKey_ex, PEM_read_PrivateKey,

PEM_write_bio_PrivateKey_ex, PEM_write_bio_PrivateKey,

PEM_write_bio_PrivateKey_traditional, PEM_write_PrivateKey_ex,

PEM_write_PrivateKey, PEM_write_bio_PKCS8PrivateKey,

PEM_write_PKCS8PrivateKey, PEM_write_bio_PKCS8PrivateKey_nid,

PEM_write_PKCS8PrivateKey_nid, PEM_read_bio_PUBKEY_ex,

PEM_read_bio_PUBKEY, PEM_read_PUBKEY_ex, PEM_read_PUBKEY,

PEM_write_bio_PUBKEY_ex, PEM_write_bio_PUBKEY, PEM_write_PUBKEY_ex,

PEM_write_PUBKEY, PEM_read_bio_RSAPrivateKey, PEM_read_RSAPrivateKey,

PEM_write_bio_RSAPrivateKey, PEM_write_RSAPrivateKey,

PEM_read_bio_RSAPublicKey, PEM_read_RSAPublicKey,

PEM_write_bio_RSAPublicKey, PEM_write_RSAPublicKey,

PEM_read_bio_RSA_PUBKEY, PEM_read_RSA_PUBKEY, PEM_write_bio_RSA_PUBKEY,

PEM_write_RSA_PUBKEY, PEM_read_bio_DSAPrivateKey,

PEM_read_DSAPrivateKey, PEM_write_bio_DSAPrivateKey,

PEM_write_DSAPrivateKey, PEM_read_bio_DSA_PUBKEY, PEM_read_DSA_PUBKEY,

PEM_write_bio_DSA_PUBKEY, PEM_write_DSA_PUBKEY,

PEM_read_bio_Parameters_ex, PEM_read_bio_Parameters,

PEM_write_bio_Parameters, PEM_read_bio_DSAparams, PEM_read_DSAparams,

PEM_write_bio_DSAparams, PEM_write_DSAparams, PEM_read_bio_DHparams,

PEM_read_DHparams, PEM_write_bio_DHparams, PEM_write_DHparams,

PEM_read_bio_X509, PEM_read_X509, PEM_write_bio_X509, PEM_write_X509,

PEM_read_bio_X509_AUX, PEM_read_X509_AUX, PEM_write_bio_X509_AUX,

PEM_write_X509_AUX, PEM_read_bio_X509_REQ, PEM_read_X509_REQ,

PEM_write_bio_X509_REQ, PEM_write_X509_REQ, PEM_write_bio_X509_REQ_NEW,

PEM_write_X509_REQ_NEW, PEM_read_bio_X509_CRL, PEM_read_X509_CRL,

PEM_write_bio_X509_CRL, PEM_write_X509_CRL, PEM_read_bio_PKCS7,

PEM_read_PKCS7, PEM_write_bio_PKCS7, PEM_write_PKCS7 - PEM routines

SYNOPSIS

```
#include <openssl/pem.h>

typedef int pem_password_cb(char *buf, int size, int rwflag, void *u);

EVP_PKEY *PEM_read_bio_PrivateKey_ex(BIO *bp, EVP_PKEY **x,
                                     pem_password_cb *cb, void *u,
                                     OSSL_LIB_CTX *libctx, const char *propq);
EVP_PKEY *PEM_read_bio_PrivateKey(BIO *bp, EVP_PKEY **x,
                                  pem_password_cb *cb, void *u);
EVP_PKEY *PEM_read_PrivateKey_ex(FILE *fp, EVP_PKEY **x, pem_password_cb *cb,
                                 void *u, OSSL_LIB_CTX *libctx,
                                 const char *propq);
EVP_PKEY *PEM_read_PrivateKey(FILE *fp, EVP_PKEY **x,
                              pem_password_cb *cb, void *u);
int PEM_write_bio_PrivateKey_ex(BIO *bp, const EVP_PKEY *x,
                                const EVP_CIPHER *enc,
                                unsigned char *kstr, int klen,
                                pem_password_cb *cb, void *u,
                                OSSL_LIB_CTX *libctx, const char *propq);
int PEM_write_bio_PrivateKey(BIO *bp, const EVP_PKEY *x, const EVP_CIPHER *enc,
                             unsigned char *kstr, int klen,
                             pem_password_cb *cb, void *u);
int PEM_write_bio_PrivateKey_traditional(BIO *bp, EVP_PKEY *x,
                                         const EVP_CIPHER *enc,
                                         unsigned char *kstr, int klen,
                                         pem_password_cb *cb, void *u);
```

```
int PEM_write_PrivateKey_ex(FILE *fp, EVP_PKEY *x, const EVP_CIPHER *enc,
                            unsigned char *kstr, int klen,
                            pem_password_cb *cb, void *u,
                            OSSL_LIB_CTX *libctx, const char *propq);
int PEM_write_PrivateKey(FILE *fp, EVP_PKEY *x, const EVP_CIPHER *enc,
                         unsigned char *kstr, int klen,
                         pem_password_cb *cb, void *u);
int PEM_write_bio_PKCS8PrivateKey(BIO *bp, EVP_PKEY *x, const EVP_CIPHER *enc,
                                  char *kstr, int klen,
                                  pem_password_cb *cb, void *u);
int PEM_write_PKCS8PrivateKey(FILE *fp, EVP_PKEY *x, const EVP_CIPHER *enc,
                              char *kstr, int klen,
                              pem_password_cb *cb, void *u);
int PEM_write_bio_PKCS8PrivateKey_nid(BIO *bp, const EVP_PKEY *x, int nid,
                                      char *kstr, int klen,
                                      pem_password_cb *cb, void *u);
int PEM_write_PKCS8PrivateKey_nid(FILE *fp, const EVP_PKEY *x, int nid,
                                  char *kstr, int klen,
                                  pem_password_cb *cb, void *u);
EVP_PKEY *PEM_read_bio_PUBKEY_ex(BIO *bp, EVP_PKEY **x,
                                 pem_password_cb *cb, void *u,
                                 OSSL_LIB_CTX *libctx, const char *propq);
EVP_PKEY *PEM_read_bio_PUBKEY(BIO *bp, EVP_PKEY **x,
                              pem_password_cb *cb, void *u);
EVP_PKEY *PEM_read_PUBKEY_ex(FILE *fp, EVP_PKEY **x,
                             pem_password_cb *cb, void *u,
                             OSSL_LIB_CTX *libctx, const char *propq);
EVP_PKEY *PEM_read_PUBKEY(FILE *fp, EVP_PKEY **x,
                          pem_password_cb *cb, void *u);
int PEM_write_bio_PUBKEY_ex(BIO *bp, EVP_PKEY *x,
                            OSSL_LIB_CTX *libctx, const char *propq);
int PEM_write_bio_PUBKEY(BIO *bp, EVP_PKEY *x);
int PEM_write_PUBKEY_ex(FILE *fp, EVP_PKEY *x,
```

```
                    OSSL_LIB_CTX *libctx, const char *propq);
int PEM_write_PUBKEY(FILE *fp, EVP_PKEY *x);
EVP_PKEY *PEM_read_bio_Parameters_ex(BIO *bp, EVP_PKEY **x,
                                     OSSL_LIB_CTX *libctx, const char *propq);
EVP_PKEY *PEM_read_bio_Parameters(BIO *bp, EVP_PKEY **x);
int PEM_write_bio_Parameters(BIO *bp, const EVP_PKEY *x);
X509 *PEM_read_bio_X509(BIO *bp, X509 **x, pem_password_cb *cb, void *u);
X509 *PEM_read_X509(FILE *fp, X509 **x, pem_password_cb *cb, void *u);
int PEM_write_bio_X509(BIO *bp, X509 *x);
int PEM_write_X509(FILE *fp, X509 *x);
X509 *PEM_read_bio_X509_AUX(BIO *bp, X509 **x, pem_password_cb *cb, void *u);
X509 *PEM_read_X509_AUX(FILE *fp, X509 **x, pem_password_cb *cb, void *u);
int PEM_write_bio_X509_AUX(BIO *bp, X509 *x);
int PEM_write_X509_AUX(FILE *fp, X509 *x);
X509_REQ *PEM_read_bio_X509_REQ(BIO *bp, X509_REQ **x,
                                pem_password_cb *cb, void *u);
X509_REQ *PEM_read_X509_REQ(FILE *fp, X509_REQ **x,
                            pem_password_cb *cb, void *u);
int PEM_write_bio_X509_REQ(BIO *bp, X509_REQ *x);
int PEM_write_X509_REQ(FILE *fp, X509_REQ *x);
int PEM_write_bio_X509_REQ_NEW(BIO *bp, X509_REQ *x);
int PEM_write_X509_REQ_NEW(FILE *fp, X509_REQ *x);
X509_CRL *PEM_read_bio_X509_CRL(BIO *bp, X509_CRL **x,
                                pem_password_cb *cb, void *u);
X509_CRL *PEM_read_X509_CRL(FILE *fp, X509_CRL **x,
                            pem_password_cb *cb, void *u);
int PEM_write_bio_X509_CRL(BIO *bp, X509_CRL *x);
int PEM_write_X509_CRL(FILE *fp, X509_CRL *x);
PKCS7 *PEM_read_bio_PKCS7(BIO *bp, PKCS7 **x, pem_password_cb *cb, void *u);
PKCS7 *PEM_read_PKCS7(FILE *fp, PKCS7 **x, pem_password_cb *cb, void *u);
int PEM_write_bio_PKCS7(BIO *bp, PKCS7 *x);
int PEM_write_PKCS7(FILE *fp, PKCS7 *x);
```

The following functions have been deprecated since OpenSSL 3.0, and can

be hidden entirely by defining OPENSSL_API_COMPAT with a suitable version value, see openssl_user_macros(7):

```
 RSA *PEM_read_bio_RSAPrivateKey(BIO *bp, RSA **x,
                                 pem_password_cb *cb, void *u);
 RSA *PEM_read_RSAPrivateKey(FILE *fp, RSA **x,
                             pem_password_cb *cb, void *u);
 int PEM_write_bio_RSAPrivateKey(BIO *bp, RSA *x, const EVP_CIPHER *enc,
                                 unsigned char *kstr, int klen,
                                 pem_password_cb *cb, void *u);
 int PEM_write_RSAPrivateKey(FILE *fp, RSA *x, const EVP_CIPHER *enc,
                             unsigned char *kstr, int klen,
                             pem_password_cb *cb, void *u);
 RSA *PEM_read_bio_RSAPublicKey(BIO *bp, RSA **x,
                                pem_password_cb *cb, void *u);
 RSA *PEM_read_RSAPublicKey(FILE *fp, RSA **x,
                            pem_password_cb *cb, void *u);
 int PEM_write_bio_RSAPublicKey(BIO *bp, RSA *x);
 int PEM_write_RSAPublicKey(FILE *fp, RSA *x);
 RSA *PEM_read_bio_RSA_PUBKEY(BIO *bp, RSA **x,
                              pem_password_cb *cb, void *u);
 RSA *PEM_read_RSA_PUBKEY(FILE *fp, RSA **x,
                          pem_password_cb *cb, void *u);
 int PEM_write_bio_RSA_PUBKEY(BIO *bp, RSA *x);
 int PEM_write_RSA_PUBKEY(FILE *fp, RSA *x);
 DSA *PEM_read_bio_DSAPrivateKey(BIO *bp, DSA **x,
                                 pem_password_cb *cb, void *u);
 DSA *PEM_read_DSAPrivateKey(FILE *fp, DSA **x,
                             pem_password_cb *cb, void *u);
 int PEM_write_bio_DSAPrivateKey(BIO *bp, DSA *x, const EVP_CIPHER *enc,
                                 unsigned char *kstr, int klen,
                                 pem_password_cb *cb, void *u);
 int PEM_write_DSAPrivateKey(FILE *fp, DSA *x, const EVP_CIPHER *enc,
                             unsigned char *kstr, int klen,
```

```
                                pem_password_cb *cb, void *u);
 DSA *PEM_read_bio_DSA_PUBKEY(BIO *bp, DSA **x,
                                pem_password_cb *cb, void *u);
 DSA *PEM_read_DSA_PUBKEY(FILE *fp, DSA **x,
                                pem_password_cb *cb, void *u);
 int PEM_write_bio_DSA_PUBKEY(BIO *bp, DSA *x);
 int PEM_write_DSA_PUBKEY(FILE *fp, DSA *x);
 DSA *PEM_read_bio_DSAparams(BIO *bp, DSA **x, pem_password_cb *cb, void *u);
 DSA *PEM_read_DSAparams(FILE *fp, DSA **x, pem_password_cb *cb, void *u);
 int PEM_write_bio_DSAparams(BIO *bp, DSA *x);
 int PEM_write_DSAparams(FILE *fp, DSA *x);
 DH *PEM_read_bio_DHparams(BIO *bp, DH **x, pem_password_cb *cb, void *u);
 DH *PEM_read_DHparams(FILE *fp, DH **x, pem_password_cb *cb, void *u);
 int PEM_write_bio_DHparams(BIO *bp, DH *x);
 int PEM_write_DHparams(FILE *fp, DH *x);
```

DESCRIPTION

All of the functions described on this page that have a TYPE of DH, DSA and RSA are deprecated. Applications should use OSSL_ENCODER_to_bio(3) and OSSL_DECODER_from_bio(3) instead.

The PEM functions read or write structures in PEM format. In this sense PEM format is simply base64 encoded data surrounded by header lines.

For more details about the meaning of arguments see the PEM FUNCTION ARGUMENTS section.

Each operation has four functions associated with it. For brevity the term "TYPE functions" will be used below to collectively refer to the PEM_read_bio_TYPE(), PEM_read_TYPE(), PEM_write_bio_TYPE(), and PEM_write_TYPE() functions.

Some operations have additional variants that take a library context libctx and a property query string propq. The X509, X509_REQ and X509_CRL objects may have an associated library context or property query string but there are no variants of these functions that take a library context or property query string parameter. In this case it is possible to set the appropriate library context or property query

string by creating an empty X509, X509_REQ or X509_CRL object using X509_new_ex(3), X509_REQ_new_ex(3) or X509_CRL_new_ex(3) respectively. Then pass the empty object as a parameter to the relevant PEM function. See the "EXAMPLES" section below.

The PrivateKey functions read or write a private key in PEM format using an EVP_PKEY structure. The write routines use PKCS#8 private key format and are equivalent to PEM_write_bio_PKCS8PrivateKey(). The read functions transparently handle traditional and PKCS#8 format encrypted and unencrypted keys.

PEM_write_bio_PrivateKey_traditional() writes out a private key in the "traditional" format with a simple private key marker and should only be used for compatibility with legacy programs.

PEM_write_bio_PKCS8PrivateKey() and PEM_write_PKCS8PrivateKey() write a private key in an EVP_PKEY structure in PKCS#8 EncryptedPrivateKeyInfo format using PKCS#5 v2.0 password based encryption algorithms. The cipher argument specifies the encryption algorithm to use: unlike some other PEM routines the encryption is applied at the PKCS#8 level and not in the PEM headers. If cipher is NULL then no encryption is used and a PKCS#8 PrivateKeyInfo structure is used instead.

PEM_write_bio_PKCS8PrivateKey_nid() and PEM_write_PKCS8PrivateKey_nid() also write out a private key as a PKCS#8 EncryptedPrivateKeyInfo however it uses PKCS#5 v1.5 or PKCS#12 encryption algorithms instead. The algorithm to use is specified in the nid parameter and should be the NID of the corresponding OBJECT IDENTIFIER (see NOTES section).

The PUBKEY functions process a public key using an EVP_PKEY structure. The public key is encoded as a SubjectPublicKeyInfo structure.

The RSAPrivateKey functions process an RSA private key using an RSA structure. The write routines uses traditional format. The read routines handles the same formats as the PrivateKey functions but an error occurs if the private key is not RSA.

The RSAPublicKey functions process an RSA public key using an RSA structure. The public key is encoded using a PKCS#1 RSAPublicKey structure.

The RSA_PUBKEY functions also process an RSA public key using an RSA structure. However, the public key is encoded using a SubjectPublicKeyInfo structure and an error occurs if the public key is not RSA.

The DSAPrivateKey functions process a DSA private key using a DSA structure. The write routines uses traditional format. The read routines handles the same formats as the PrivateKey functions but an error occurs if the private key is not DSA.

The DSA_PUBKEY functions process a DSA public key using a DSA structure. The public key is encoded using a SubjectPublicKeyInfo structure and an error occurs if the public key is not DSA.

The Parameters functions read or write key parameters in PEM format using an EVP_PKEY structure. The encoding depends on the type of key; for DSA key parameters, it will be a Dss-Parms structure as defined in RFC2459, and for DH key parameters, it will be a PKCS#3 DHparameter structure. These functions only exist for the BIO type.

The DSAparams functions process DSA parameters using a DSA structure. The parameters are encoded using a Dss-Parms structure as defined in RFC2459.

The DHparams functions process DH parameters using a DH structure. The parameters are encoded using a PKCS#3 DHparameter structure.

The X509 functions process an X509 certificate using an X509 structure. They will also process a trusted X509 certificate but any trust settings are discarded.

The X509_AUX functions process a trusted X509 certificate using an X509 structure.

The X509_REQ and X509_REQ_NEW functions process a PKCS#10 certificate request using an X509_REQ structure. The X509_REQ write functions use CERTIFICATE REQUEST in the header whereas the X509_REQ_NEW functions use NEW CERTIFICATE REQUEST (as required by some CAs). The X509_REQ read functions will handle either form so there are no X509_REQ_NEW read functions.

The X509_CRL functions process an X509 CRL using an X509_CRL structure.

The PKCS7 functions process a PKCS#7 ContentInfo using a PKCS7 structure.

PEM FUNCTION ARGUMENTS

The PEM functions have many common arguments.

The bp BIO parameter (if present) specifies the BIO to read from or write to.

The fp FILE parameter (if present) specifies the FILE pointer to read from or write to.

The PEM read functions all take an argument TYPE **x and return a TYPE * pointer. Where TYPE is whatever structure the function uses. If x is NULL then the parameter is ignored. If x is not NULL but *x is NULL then the structure returned will be written to *x. If neither x nor *x is NULL then an attempt is made to reuse the structure at *x (but see BUGS and EXAMPLES sections). Irrespective of the value of x a pointer to the structure is always returned (or NULL if an error occurred).

The PEM functions which write private keys take an enc parameter which specifies the encryption algorithm to use, encryption is done at the PEM level. If this parameter is set to NULL then the private key is written in unencrypted form.

The cb argument is the callback to use when querying for the pass phrase used for encrypted PEM structures (normally only private keys).

For the PEM write routines if the kstr parameter is not NULL then klen bytes at kstr are used as the passphrase and cb is ignored.

If the cb parameters is set to NULL and the u parameter is not NULL then the u parameter is interpreted as a NUL terminated string to use as the passphrase. If both cb and u are NULL then the default callback routine is used which will typically prompt for the passphrase on the current terminal with echoing turned off.

The default passphrase callback is sometimes inappropriate (for example in a GUI application) so an alternative can be supplied. The callback routine has the following form:

```
int cb(char *buf, int size, int rwflag, void *u);
```

buf is the buffer to write the passphrase to. size is the maximum

length of the passphrase (i.e. the size of buf). rwflag is a flag which is set to 0 when reading and 1 when writing. A typical routine will ask the user to verify the passphrase (for example by prompting for it twice) if rwflag is 1. The u parameter has the same value as the u parameter passed to the PEM routine. It allows arbitrary data to be passed to the callback by the application (for example a window handle in a GUI application). The callback must return the number of characters in the passphrase or -1 if an error occurred. The passphrase can be arbitrary data; in the case where it is a string, it is not NUL terminated. See the "EXAMPLES" section below.

Some implementations may need to use cryptographic algorithms during their operation. If this is the case and libctx and propq parameters have been passed then any algorithm fetches will use that library context and property query string. Otherwise the default library context and property query string will be used.

NOTES

The PEM reading functions will skip any extraneous content or PEM data of a different type than they expect. This allows for example having a certificate (or multiple certificates) and a key in the PEM format in a single file.

The old PrivateKey write routines are retained for compatibility. New applications should write private keys using the PEM_write_bio_PKCS8PrivateKey() or PEM_write_PKCS8PrivateKey() routines because they are more secure (they use an iteration count of 2048 whereas the traditional routines use a count of 1) unless compatibility with older versions of OpenSSL is important.

The PrivateKey read routines can be used in all applications because they handle all formats transparently.

A frequent cause of problems is attempting to use the PEM routines like this:

```
 X509 *x;
 PEM_read_bio_X509(bp, &x, 0, NULL);
```

this is a bug because an attempt will be made to reuse the data at x

which is an uninitialised pointer.

These functions make no assumption regarding the pass phrase received from the password callback. It will simply be treated as a byte sequence.

PEM ENCRYPTION FORMAT

These old PrivateKey routines use a non standard technique for encryption.

The private key (or other data) takes the following form:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,3F17F5316E2BAC89
...base64 encoded data...
-----END RSA PRIVATE KEY-----
```

The line beginning with Proc-Type contains the version and the protection on the encapsulated data. The line beginning DEK-Info contains two comma separated values: the encryption algorithm name as used by EVP_get_cipherbyname() and an initialization vector used by the cipher encoded as a set of hexadecimal digits. After those two lines is the base64-encoded encrypted data.

The encryption key is derived using EVP_BytesToKey(). The cipher's initialization vector is passed to EVP_BytesToKey() as the salt parameter. Internally, PKCS5_SALT_LEN bytes of the salt are used (regardless of the size of the initialization vector). The user's password is passed to EVP_BytesToKey() using the data and datal parameters. Finally, the library uses an iteration count of 1 for EVP_BytesToKey().

The key derived by EVP_BytesToKey() along with the original initialization vector is then used to decrypt the encrypted data. The iv produced by EVP_BytesToKey() is not utilized or needed, and NULL should be passed to the function.

The pseudo code to derive the key would look similar to:

```
EVP_CIPHER* cipher = EVP_des_ede3_cbc();
EVP_MD* md = EVP_md5();
```

```
unsigned int nkey = EVP_CIPHER_get_key_length(cipher);
unsigned int niv = EVP_CIPHER_get_iv_length(cipher);
unsigned char key[nkey];
unsigned char iv[niv];
memcpy(iv, HexToBin("3F17F5316E2BAC89"), niv);
rc = EVP_BytesToKey(cipher, md, iv /*salt*/, pword, plen, 1, key, NULL /*iv*/);
if (rc != nkey)
    /* Error */
/* On success, use key and iv to initialize the cipher */
```

BUGS

The PEM read routines in some versions of OpenSSL will not correctly reuse an existing structure. Therefore, the following:

```
PEM_read_bio_X509(bp, &x, 0, NULL);
```

where x already contains a valid certificate, may not work, whereas:

```
X509_free(x);
x = PEM_read_bio_X509(bp, NULL, 0, NULL);
```

is guaranteed to work. It is always acceptable for x to contain a newly allocated, empty X509 object (for example allocated via X509_new_ex(3)).

RETURN VALUES

The read routines return either a pointer to the structure read or NULL if an error occurred.

The write routines return 1 for success or 0 for failure.

EXAMPLES

Although the PEM routines take several arguments in almost all applications most of them are set to 0 or NULL.

To read a certificate with a library context in PEM format from a BIO:

```
X509 *x = X509_new_ex(libctx, NULL);
if (x == NULL)
    /* Error */
if (PEM_read_bio_X509(bp, &x, 0, NULL) == NULL)
    /* Error */
```

Read a certificate in PEM format from a BIO:

```
 X509 *x;
 x = PEM_read_bio_X509(bp, NULL, 0, NULL);
 if (x == NULL)
    /* Error */
```

Alternative method:

```
 X509 *x = NULL;
 if (!PEM_read_bio_X509(bp, &x, 0, NULL))
    /* Error */
```

Write a certificate to a BIO:

```
 if (!PEM_write_bio_X509(bp, x))
    /* Error */
```

Write a private key (using traditional format) to a BIO using triple

DES encryption, the pass phrase is prompted for:

```
 if (!PEM_write_bio_PrivateKey(bp, key, EVP_des_ede3_cbc(), NULL, 0, 0, NULL))
    /* Error */
```

Write a private key (using PKCS#8 format) to a BIO using triple DES

encryption, using the pass phrase "hello":

```
 if (!PEM_write_bio_PKCS8PrivateKey(bp, key, EVP_des_ede3_cbc(),
                                    NULL, 0, 0, "hello"))
    /* Error */
```

Read a private key from a BIO using a pass phrase callback:

```
 key = PEM_read_bio_PrivateKey(bp, NULL, pass_cb, "My Private Key");
 if (key == NULL)
    /* Error */
```

Skeleton pass phrase callback:

```
 int pass_cb(char *buf, int size, int rwflag, void *u)
 {
    /* We'd probably do something else if 'rwflag' is 1 */
    printf("Enter pass phrase for \"%s\"\n", (char *)u);
    /* get pass phrase, length 'len' into 'tmp' */
    char *tmp = "hello";
    if (tmp == NULL) /* An error occurred */
       return -1;
```

```
    size_t len = strlen(tmp);
    if (len > size)
        len = size;
    memcpy(buf, tmp, len);
    return len;
}
```

SEE ALSO

EVP_EncryptInit(3), EVP_BytesToKey(3), passphrase-encoding(7)

HISTORY

The old Netscape certificate sequences were no longer documented in OpenSSL 1.1.0; applications should use the PKCS7 standard instead as they will be formally deprecated in a future releases.

PEM_read_bio_PrivateKey_ex(), PEM_read_PrivateKey_ex(), PEM_read_bio_PUBKEY_ex(), PEM_read_PUBKEY_ex() and PEM_read_bio_Parameters_ex() were introduced in OpenSSL 3.0.

The functions PEM_read_bio_RSAPrivateKey(), PEM_read_RSAPrivateKey(), PEM_write_bio_RSAPrivateKey(), PEM_write_RSAPrivateKey(), PEM_read_bio_RSAPublicKey(), PEM_read_RSAPublicKey(), PEM_write_bio_RSAPublicKey(), PEM_write_RSAPublicKey(), PEM_read_bio_RSA_PUBKEY(), PEM_read_RSA_PUBKEY(), PEM_write_bio_RSA_PUBKEY(), PEM_write_RSA_PUBKEY(), PEM_read_bio_DSAPrivateKey(), PEM_read_DSAPrivateKey(), PEM_write_bio_DSAPrivateKey(), PEM_write_DSAPrivateKey(), PEM_read_bio_DSA_PUBKEY(), PEM_read_DSA_PUBKEY(), PEM_write_bio_DSA_PUBKEY(), PEM_write_DSA_PUBKEY();

PEM_read_bio_DSAparams(), PEM_read_DSAparams(), PEM_write_bio_DSAparams(), PEM_write_DSAparams(), PEM_read_bio_DHparams(), PEM_read_DHparams(), PEM_write_bio_DHparams() and PEM_write_DHparams() were deprecated in 3.0.

COPYRIGHT

in the file LICENSE in the source distribution or at

<https://www.openssl.org/source/license.html>.