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_get_EVP_CIPHER_INFO.3ossl' command

$ man PEM_get_EVP_CIPHER_INFO.3ossl

PEM_READ(3ossl) OpenSSL PEM_READ(3ossl)

NAME

PEM_write, PEM_write_bio, PEM_read, PEM_read_bio, PEM_do_header,

PEM_get_EVP_CIPHER_INFO - PEM encoding routines

SYNOPSIS

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

int PEM_write(FILE *fp, const char *name, const char *header,
              const unsigned char *data, long len);
int PEM_write_bio(BIO *bp, const char *name, const char *header,
                  const unsigned char *data, long len);

int PEM_read(FILE *fp, char **name, char **header,
             unsigned char **data, long *len);
int PEM_read_bio(BIO *bp, char **name, char **header,
                 unsigned char **data, long *len);

int PEM_get_EVP_CIPHER_INFO(char *header, EVP_CIPHER_INFO *cinfo);
int PEM_do_header(EVP_CIPHER_INFO *cinfo, unsigned char *data, long *len,
                  pem_password_cb *cb, void *u);
```

DESCRIPTION

These functions read and write PEM-encoded objects, using the PEM type name, any additional header information, and the raw data of length len.

PEM is the term used for binary content encoding first defined in IETF RFC 1421. The content is a series of base64-encoded lines, surrounded by begin/end markers each on their own line. For example:

```
-----BEGIN PRIVATE KEY-----
MIICdg....
... bhTQ==
-----END PRIVATE KEY-----
```

Optional header line(s) may appear after the begin line, and their existence depends on the type of object being written or read.

PEM_write() writes to the file fp, while PEM_write_bio() writes to the BIO bp. The name is the name to use in the marker, the header is the header value or NULL, and data and len specify the data and its length.

The final data buffer is typically an ASN.1 object which can be decoded with the d2i function appropriate to the type name; see d2i_X509(3) for examples.

PEM_read() reads from the file fp, while PEM_read_bio() reads from the BIO bp. Both skip any non-PEM data that precedes the start of the next PEM object. When an object is successfully retrieved, the type name from the "----BEGIN <type>-----" is returned via the name argument, any encapsulation headers are returned in header and the base64-decoded content and its length are returned via data and len respectively. The name, header and data pointers are allocated via OPENSSL_malloc() and should be freed by the caller via OPENSSL_free() when no longer needed.

PEM_get_EVP_CIPHER_INFO() can be used to determine the data returned by PEM_read() or PEM_read_bio() is encrypted and to retrieve the associated cipher and IV. The caller passes a pointer to structure of type EVP_CIPHER_INFO via the cinfo argument and the header returned via PEM_read() or PEM_read_bio(). If the call is successful 1 is returned and the cipher and IV are stored at the address pointed to by cinfo. When the header is malformed, or not supported or when the cipher is unknown or some internal error happens 0 is returned. This function is deprecated, see NOTES below.

PEM_do_header() can then be used to decrypt the data if the header indicates encryption. The cinfo argument is a pointer to the structure initialized by the previous call to PEM_get_EVP_CIPHER_INFO(). The data and len arguments are those returned by the previous call to PEM_read() or PEM_read_bio(). The cb and u arguments make it possible to override the default password prompt function as described in PEM_read_PrivateKey(3). On successful completion the data is decrypted in place, and len is updated to indicate the plaintext length. This function is deprecated, see NOTES below.

If the data is a priori known to not be encrypted, then neither PEM_do_header() nor PEM_get_EVP_CIPHER_INFO() need be called.

RETURN VALUES

PEM_read() and PEM_read_bio() return 1 on success and 0 on failure, the latter includes the case when no more PEM objects remain in the input file. To distinguish end of file from more serious errors the caller must peek at the error stack and check for PEM_R_NO_START_LINE, which indicates that no more PEM objects were found. See ERR_peek_last_error(3), ERR_GET_REASON(3).

PEM_get_EVP_CIPHER_INFO() and PEM_do_header() return 1 on success, and

0 on failure. The data is likely meaningless if these functions fail.

NOTES

The PEM_get_EVP_CIPHER_INFO() and PEM_do_header() functions are deprecated. This is because the underlying PEM encryption format is obsolete, and should be avoided. It uses an encryption format with an OpenSSL-specific key-derivation function, which employs MD5 with an iteration count of 1! Instead, private keys should be stored in PKCS#8 form, with a strong PKCS#5 v2.0 PBE. See PEM_write_PrivateKey(3) and d2i_PKCS8PrivateKey_bio(3).

PEM_do_header() makes no assumption regarding the pass phrase received from the password callback. It will simply be treated as a byte sequence.

SEE ALSO

ERR_peek_last_error(3), ERR_GET_LIB(3), d2i_PKCS8PrivateKey_bio(3), passphrase-encoding(7)

COPYRIGHT