NAME

OSSL_DECODER_CTX_new_for_pkey, OSSL_DECODER_CTX_set_passphrase, OSSL_DECODER_CTX_set_pem_password_cb, OSSL_DECODER_CTX_set_passphrase_ui, OSSL_DECODER_CTX_set_passphrase_cb - Decoder routines to decode EVP_PKEYs

SYNOPSIS

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

OSSL_DECODER_CTX *
OSSL_DECODER_CTX_new_for_pkey(EVP_PKEY **pkey,
                              const char *input_type,
                              const char *input_struct,
                              const char *keytype, int selection,
                              OSSL_LIB_CTX *libctx, const char *propquery);

int OSSL_DECODER_CTX_set_passphrase(OSSL_DECODER_CTX *ctx,
                                    const unsigned char *kstr,
                                    size_t klen);
int OSSL_DECODER_CTX_set_pem_password_cb(OSSL_DECODER_CTX *ctx,
                                         pem_password_cb *cb,
                                         void *cbarg);
int OSSL_DECODER_CTX_set_passphrase_ui(OSSL_DECODER_CTX *ctx,
                                       const UI_METHOD *ui_method,
                                       void *ui_data);
int OSSL_DECODER_CTX_set_passphrase_cb(OSSL_DECODER_CTX *ctx,
                                       OSSL_PASSPHRASE_CALLBACK *cb,
                                       void *cbarg);
```

DESCRIPTION

OSSL_DECODER_CTX_new_for_pkey() is a utility function that creates a **OSSL_DECODER_CTX**, finds all applicable decoder implementations and sets them up, so all the caller has to do next is call functions like **OSSL_DECODER_from_bio**(3). The caller may use the optional *input_type*, *input_struct*, *keytype* and *selection* to specify what the input is expected to contain. The *pkey* must reference an **EVP_PKEY *** variable that will be set to the newly created **EVP_PKEY** on successful decoding. The referenced variable must be initialized to NULL before calling the function.

Internally **OSSL_DECODER_CTX_new_for_pkey()** searches for all available **EVP_KEYMGMT**(3) implementations, and then builds a list of all potential decoder implementations that may be able to process the encoded input into data suitable for **EVP_PKEY**s. All these implementations are implicitly fetched using *libctx* and *propquery*.

The search of decoder implementations can be limited with *input_type* and *input_struct* which specifies a starting input type and input structure. NULL is valid for both of them and signifies that the decoder implementations will find out the input type on their own. They are set with **OSSL_DECODER_CTX_set_input_type**(3) and **OSSL_DECODER_CTX_set_input_structure**(3). See "Input Types" and "Input Structures" below for further information.

The search of decoder implementations can also be limited with *keytype* and *selection*, which specifies the expected resulting keytype and contents. NULL and zero are valid and signify that the decoder implementations will find out the keytype and key contents on their own from the input they get.

If no suitable decoder implementation is found, **OSSL_DECODER_CTX_new_for_pkey()** still creates a **OSSL_DECODER_CTX**, but with no associated decoder (**OSSL_DECODER_CTX_get_num_decoders**(3) returns zero). This helps the caller to distinguish between an error when creating the **OSSL_ENCODER_CTX** and missing encoder implementation, and allows it to act accordingly.

OSSL_DECODER_CTX_set_passphrase() gives the implementation a pass phrase to use when decrypting the encoded private key. Alternatively, a pass phrase callback may be specified with the following functions.

OSSL_DECODER_CTX_set_pem_password_cb(), **OSSL_DECODER_CTX_set_passphrase_ui()** and **OSSL_DECODER_CTX_set_passphrase_cb()** set up a callback method that the implementation can use to prompt for a pass phrase, giving the caller the choice of preferred pass phrase callback form. These are called indirectly, through an internal **OSSL_PASSPHRASE_CALLBACK**(3) function.

The internal **OSSL_PASSPHRASE_CALLBACK**(3) function caches the pass phrase, to be re-used in all decodings that are performed in the same decoding run (for example, within one **OSSL_DECODER_from_bio**(3) call).

Input Types

Available input types depend on the implementations that available providers offer, and provider documentation should have the details.

Among the known input types that OpenSSL decoder implementations offer for **EVP_PKEY**s are "DER", "PEM", "MSBLOB" and "PVK". See **openssl-glossary**(7) for further information on what these input types mean.

Input Structures

Available input structures depend on the implementations that available providers offer, and provider documentation should have the details.

Among the known input structures that OpenSSL decoder implementations offer for **EVP_PKEY**s are "pkcs8" and "SubjectPublicKeyInfo".

OpenSSL decoder implementations also support the input structure "type-specific". This is the structure used for keys encoded according to key type specific specifications. For example, RSA keys encoded according to PKCS#1.

Selections

selection can be any one of the values described in "Selections" in **EVP_PKEY_fromdata**(3). Additionally *selection* can also be set to **0** to indicate that the code will auto detect the selection.

RETURN VALUES

OSSL_DECODER_CTX_new_for_pkey() returns a pointer to a **OSSL_DECODER_CTX**, or NULL if it couldn't be created.

OSSL_DECODER_CTX_set_passphrase(), **OSSL_DECODER_CTX_set_pem_password_cb()**, **OSSL_DECODER_CTX_set_passphrase_ui()** and **OSSL_DECODER_CTX_set_passphrase_cb()** all return 1 on success, or 0 on failure.

SEE ALSO

provider(7), **OSSL_DECODER**(3), **OSSL_DECODER_CTX**(3)

HISTORY

The functions described here were added in OpenSSL 3.0.

COPYRIGHT