NAME

provider−encoder − The OSSL_ENCODER library <−> provider functions

SYNOPSIS

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

/*
 * None of these are actual functions, but are displayed like this for
 * the function signatures for functions that are offered as function
 * pointers in OSSL_DISPATCH arrays.
 */

/* Encoder parameter accessor and descriptor */
const OSSL_PARAM *OSSL_FUNC_encoder_gettable_params(void *provctx);
int OSSL_FUNC_encoder_get_params(OSSL_PARAM params[]);

/* Functions to construct / destruct / manipulate the encoder context */
void *OSSL_FUNC_encoder_newctx(void *provctx);
void OSSL_FUNC_encoder_freectx(void *ctx);
int OSSL_FUNC_encoder_set_ctx_params(void *ctx, const OSSL_PARAM params[]);
const OSSL_PARAM *OSSL_FUNC_encoder_settable_ctx_params(void *provctx);

/* Functions to check selection support */
int OSSL_FUNC_encoder_does_selection(void *provctx, int selection);

/* Functions to encode object data */
int OSSL_FUNC_encoder_encode(void *ctx, OSSL_CORE_BIO *out,
                             const void *obj_raw,
                             const OSSL_PARAM obj_abstract[],
                             int selection,
                             OSSL_PASSPHRASE_CALLBACK *cb,
                             void *cbarg);

/* Functions to import and free a temporary object to be encoded */
void *OSSL_FUNC_encoder_import_object(void *ctx, int selection,
                                      const OSSL_PARAM params[]);
void OSSL_FUNC_encoder_free_object(void *obj);
```

DESCRIPTION

We use the wide term "encode" in this manual. This includes but is not limited to serialization.

The ENCODER operation is a generic method to encode a provider-native object (*obj_raw*) or an object abstraction (*object_abstract*, see **provider−object** (7)) into an encoded form, and write the result to the given OSSL_CORE_BIO. If the caller wants to get the encoded stream to memory, it should provide a **BIO_s_mem** (3) **BIO**.

The encoder doesn't need to know more about the **OSSL_CORE_BIO** pointer than being able to pass it to the appropriate BIO upcalls (see ''Core functions'' in **provider−base** (7)).

The ENCODER implementation may be part of a chain, where data is passed from one to the next. For example, there may be an implementation to encode an object to DER (that object is assumed to be provider-native and thereby passed via *obj_raw*), and another one that encodes DER to PEM (that one would receive the DER encoding via *obj_abstract*).

The encoding using the **OSSL_PARAM** (3) array form allows a encoder to be used for data that's been exported from another provider, and thereby allow them to exist independently of each other.

The encoding using a provider side object can only be safely used with provider data coming from the same provider, for example keys with the KEYMGMT provider.

All ‘‘functions’’ mentioned here are passed as function pointers between *libcrypto* and the provider in **OSSL_DISPATCH** arrays via **OSSL_ALGORITHM** arrays that are returned by the provider’s **provider_query_operation()** function (see ‘‘Provider Functions’’ in **provider-base** (7)).

All these ‘‘functions’’ have a corresponding function type definition named **OSSL_FUNC_{name}_fn**, and a helper function to retrieve the function pointer from an **OSSL_DISPATCH** element named **OSSL_FUNC_{name}**. For example, the ‘‘function’’ **OSSL_FUNC_encoder_encode()** has these:

```
typedef int
    (OSSL_FUNC_encoder_encode_fn)(void *ctx, OSSL_CORE_BIO *out,
                                  const void *obj_raw,
                                  const OSSL_PARAM obj_abstract[],
                                  int selection,
                                  OSSL_PASSPHRASE_CALLBACK *cb, void *cbarg);
static ossl_inline OSSL_FUNC_encoder_encode_fn
    OSSL_FUNC_encoder_encode(const OSSL_DISPATCH *opf);
```

OSSL_DISPATCH arrays are indexed by numbers that are provided as macros in **openssl-core_dispatch.h** (7), as follows:

```
OSSL_FUNC_encoder_get_params          OSSL_FUNC_ENCODER_GET_PARAMS
OSSL_FUNC_encoder_gettable_params     OSSL_FUNC_ENCODER_GETTABLE_PARAMS

OSSL_FUNC_encoder_newctx              OSSL_FUNC_ENCODER_NEWCTX
OSSL_FUNC_encoder_freectx             OSSL_FUNC_ENCODER_FREECTX
OSSL_FUNC_encoder_set_ctx_params      OSSL_FUNC_ENCODER_SET_CTX_PARAMS
OSSL_FUNC_encoder_settable_ctx_params OSSL_FUNC_ENCODER_SETTABLE_CTX_PARAMS

OSSL_FUNC_encoder_does_selection      OSSL_FUNC_ENCODER_DOES_SELECTION

OSSL_FUNC_encoder_encode              OSSL_FUNC_ENCODER_ENCODE

OSSL_FUNC_encoder_import_object       OSSL_FUNC_ENCODER_IMPORT_OBJECT
OSSL_FUNC_encoder_free_object         OSSL_FUNC_ENCODER_FREE_OBJECT
```

Names and properties

The name of an implementation should match the type of object it handles. For example, an implementation that encodes an RSA key should be named ‘‘RSA’’. Likewise, an implementation that further encodes DER should be named ‘‘DER’’.

Properties can be used to further specify details about an implementation:

output
: This property is used to specify what type of output the implementation produces.

  This property is *mandatory*.

  OpenSSL providers recognize the following output types:

  text An implementation with that output type outputs human readable text, making that implementation suitable for `-text` output in diverse **openssl** (1) commands.

  pem
  : An implementation with that output type outputs PEM formatted data.

  der An implementation with that output type outputs DER formatted data.

  msblob
  : An implementation with that output type outputs MSBLOB formatted data.

  pvk An implementation with that output type outputs PVK formatted data.

structure

This property is used to specify the structure that is used for the encoded object. An example could be `pkcs8`, to specify explicitly that an object (presumably an asymmetric key pair, in this case) will be wrapped in a PKCS#8 structure as part of the encoding.

This property is *optional*.

The possible values of both these properties is open ended. A provider may very well specify output types and structures that libcrypto doesn't know anything about.

Subset selections

Sometimes, an object has more than one subset of data that is interesting to treat separately or together. It's possible to specify what subsets are to be encoded, with a set of bits *selection* that are passed in an **int**.

This set of bits depend entirely on what kind of provider-side object is passed. For example, those bits are assumed to be the same as those used with **provider-keymgmt**(7) (see ''Key Objects'' in **provider-keymgmt**(7)) when the object is an asymmetric keypair.

ENCODER implementations are free to regard the *selection* as a set of hints, but must do so with care. In the end, the output must make sense, and if there's a corresponding decoder, the resulting decoded object must match the original object that was encoded.

OSSL_FUNC_encoder_does_selection() should tell if a particular implementation supports any of the combinations given by *selection*.

Context functions

OSSL_FUNC_encoder_newctx() returns a context to be used with the rest of the functions.

OSSL_FUNC_encoder_freectx() frees the given *ctx*, if it was created by **OSSL_FUNC_encoder_newctx()**.

OSSL_FUNC_encoder_set_ctx_params() sets context data according to parameters from *params* that it recognises. Unrecognised parameters should be ignored. Passing NULL for *params* should return true.

OSSL_FUNC_encoder_settable_ctx_params() returns a constant **OSSL_PARAM** array describing the parameters that **OSSL_FUNC_encoder_set_ctx_params()** can handle.

See **OSSL_PARAM**(3) for further details on the parameters structure used by **OSSL_FUNC_encoder_set_ctx_params()** and **OSSL_FUNC_encoder_settable_ctx_params()**.

Import functions

A provider-native object may be associated with a foreign provider, and may therefore be unsuitable for direct use with a given ENCODER implementation. Provided that the foreign provider's implementation to handle the object has a function to export that object in **OSSL_PARAM**(3) array form, the ENCODER implementation should be able to import that array and create a suitable object to be passed to **OSSL_FUNC_encoder_encode()**'s *obj_raw*.

OSSL_FUNC_encoder_import_object() should import the subset of *params* given with *selection* to create a provider-native object that can be passed as *obj_raw* to **OSSL_FUNC_encoder_encode()**.

OSSL_FUNC_encoder_free_object() should free the object that was created with **OSSL_FUNC_encoder_import_object()**.

Encoding functions

OSSL_FUNC_encoder_encode() should take a provider-native object (in *obj_raw*) or an object abstraction (in *obj_abstract*), and should output the object in encoded form to the **OSSL_CORE_BIO**. The *selection* bits, if relevant, should determine in greater detail what will be output. The encoding functions also take an **OSSL_PASSPHRASE_CALLBACK** function pointer along with a pointer to application data *cbarg*, which should be used when a pass phrase prompt is needed.

Encoder operation parameters

Operation parameters currently recognised by built-in encoders are as follows:

‘‘cipher’’ (**OSSL_ENCODER_PARAM_CIPHER**) <UTF8 string>
The name of the encryption cipher to be used when generating encrypted encoding. This is used when encoding private keys, as well as other objects that need protection.

If this name is invalid for the encoding implementation, the implementation should refuse to perform the encoding, i.e. **OSSL_FUNC_encoder_encode_data()** and **OSSL_FUNC_encoder_encode_object()** should return an error.

‘‘properties’’ (**OSSL_ENCODER_PARAM_PROPERTIES**) <UTF8 string>
The properties to be queried when trying to fetch the algorithm given with the ‘‘cipher’’ parameter. This must be given together with the ‘‘cipher’’ parameter to be considered valid.

The encoding implementation isn’t obligated to use this value. However, it is recommended that implementations that do not handle property strings return an error on receiving this parameter unless its value NULL or the empty string.

‘‘save-parameters’’ (**OSSL_ENCODER_PARAM_SAVE_PARAMETERS**) <integer>
If set to 0 disables saving of key domain parameters. Default is 1. It currently has an effect only on DSA keys.

Parameters currently recognised by the built-in pass phrase callback:

‘‘info’’ (**OSSL_PASSPHRASE_PARAM_INFO**) <UTF8 string>
A string of information that will become part of the pass phrase prompt. This could be used to give the user information on what kind of object it’s being prompted for.

RETURN VALUES

OSSL_FUNC_encoder_newctx() returns a pointer to a context, or NULL on failure.

OSSL_FUNC_encoder_set_ctx_params() returns 1, unless a recognised parameter was invalid or caused an error, for which 0 is returned.

OSSL_FUNC_encoder_settable_ctx_params() returns a pointer to an array of constant **OSSL_PARAM** elements.

OSSL_FUNC_encoder_does_selection() returns 1 if the encoder implementation supports any of the *selection* bits, otherwise 0.

OSSL_FUNC_encoder_encode() returns 1 on success, or 0 on failure.

SEE ALSO

provider (7)

HISTORY

The ENCODER interface was introduced in OpenSSL 3.0.

COPYRIGHT