NAME

provider−asym_cipher − The asym_cipher library <−> provider functions

SYNOPSIS

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

/* Context management */
void *OSSL_FUNC_asym_cipher_newctx(void *provctx);
void OSSL_FUNC_asym_cipher_freectx(void *ctx);
void *OSSL_FUNC_asym_cipher_dupctx(void *ctx);

/* Encryption */
int OSSL_FUNC_asym_cipher_encrypt_init(void *ctx, void *provkey,
                                       const OSSL_PARAM params[]);
int OSSL_FUNC_asym_cipher_encrypt(void *ctx, unsigned char *out, size_t *outlen,
                                  size_t outsize, const unsigned char *in,
                                  size_t inlen);

/* Decryption */
int OSSL_FUNC_asym_cipher_decrypt_init(void *ctx, void *provkey,
                                       const OSSL_PARAM params[]);
int OSSL_FUNC_asym_cipher_decrypt(void *ctx, unsigned char *out, size_t *outlen,
                                  size_t outsize, const unsigned char *in,
                                  size_t inlen);

/* Asymmetric Cipher parameters */
int OSSL_FUNC_asym_cipher_get_ctx_params(void *ctx, OSSL_PARAM params[]);
const OSSL_PARAM *OSSL_FUNC_asym_cipher_gettable_ctx_params(void *provctx);
int OSSL_FUNC_asym_cipher_set_ctx_params(void *ctx, const OSSL_PARAM params[]);
const OSSL_PARAM *OSSL_FUNC_asym_cipher_settable_ctx_params(void *provctx);
```

DESCRIPTION

This documentation is primarily aimed at provider authors. See **provider** (7) for further information.

The asymmetric cipher (OSSL_OP_ASYM_CIPHER) operation enables providers to implement asymmetric cipher algorithms and make them available to applications via the API functions **EVP_PKEY_encrypt** (3), **EVP_PKEY_decrypt** (3) and other related functions).

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_asym_cipher_newctx()** has these:

```
typedef void *(OSSL_FUNC_asym_cipher_newctx_fn)(void *provctx);
static ossl_inline OSSL_FUNC_asym_cipher_newctx_fn
    OSSL_FUNC_asym_cipher_newctx(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_asym_cipher_newctx	OSSL_FUNC_ASYM_CIPHER_NEWCTX
OSSL_FUNC_asym_cipher_freectx	OSSL_FUNC_ASYM_CIPHER_FREECTX
OSSL_FUNC_asym_cipher_dupctx	OSSL_FUNC_ASYM_CIPHER_DUPCTX
OSSL_FUNC_asym_cipher_encrypt_init	OSSL_FUNC_ASYM_CIPHER_ENCRYPT_INIT
OSSL_FUNC_asym_cipher_encrypt	OSSL_FUNC_ASYM_CIPHER_ENCRYPT
OSSL_FUNC_asym_cipher_decrypt_init	OSSL_FUNC_ASYM_CIPHER_DECRYPT_INIT
OSSL_FUNC_asym_cipher_decrypt	OSSL_FUNC_ASYM_CIPHER_DECRYPT
OSSL_FUNC_asym_cipher_get_ctx_params	OSSL_FUNC_ASYM_CIPHER_GET_CTX_PARAMS
OSSL_FUNC_asym_cipher_gettable_ctx_params	OSSL_FUNC_ASYM_CIPHER_GETTABLE_CTX_PA
OSSL_FUNC_asym_cipher_set_ctx_params	OSSL_FUNC_ASYM_CIPHER_SET_CTX_PARAMS
OSSL_FUNC_asym_cipher_settable_ctx_params	OSSL_FUNC_ASYM_CIPHER_SETTABLE_CTX_PA

An asymmetric cipher algorithm implementation may not implement all of these functions. In order to be a consistent set of functions a provider must implement OSSL_FUNC_asym_cipher_newctx and OSSL_FUNC_asym_cipher_freectx. It must also implement both of OSSL_FUNC_asym_cipher_encrypt_init and OSSL_FUNC_asym_cipher_encrypt, or both of OSSL_FUNC_asym_cipher_decrypt_init and OSSL_FUNC_asym_cipher_decrypt. OSSL_FUNC_asym_cipher_get_ctx_params is optional but if it is present then so must OSSL_FUNC_asym_cipher_gettable_ctx_params. Similarly, OSSL_FUNC_asym_cipher_set_ctx_params is optional but if it is present then so must OSSL_FUNC_asym_cipher_settable_ctx_params.

An asymmetric cipher algorithm must also implement some mechanism for generating, loading or importing keys via the key management (OSSL_OP_KEYMGMT) operation. See **provider−keymgmt** (7) for further details.

Context Management Functions

OSSL_FUNC_asym_cipher_newctx() should create and return a pointer to a provider side structure for holding context information during an asymmetric cipher operation. A pointer to this context will be passed back in a number of the other asymmetric cipher operation function calls. The parameter *provctx* is the provider context generated during provider initialisation (see **provider** (7)).

OSSL_FUNC_asym_cipher_freectx() is passed a pointer to the provider side asymmetric cipher context in the *ctx* parameter. This function should free any resources associated with that context.

OSSL_FUNC_asym_cipher_dupctx() should duplicate the provider side asymmetric cipher context in the *ctx* parameter and return the duplicate copy.

Encryption Functions

OSSL_FUNC_asym_cipher_encrypt_init() initialises a context for an asymmetric encryption given a provider side asymmetric cipher context in the *ctx* parameter, and a pointer to a provider key object in the *provkey* parameter. The*par ams*, if not NULL, should be set on the context in a manner similar to using **OSSL_FUNC_asym_cipher_set_ctx_params()**. The key object should have been previously generated, loaded or imported into the provider using the key management (OSSL_OP_KEYMGMT) operation (see **provider−keymgmt** (7)>. **OSSL_FUNC_asym_cipher_encrypt()** performs the actual encryption itself. A previously initialised asymmetric cipher context is passed in the *ctx* parameter. The data to be encrypted is pointed to by the *in* parameter which is *inlen* bytes long. Unless *out* is NULL, the encrypted data should be written to the location pointed to by the *out* parameter and it should not exceed *outsize* bytes in length. The length of the encrypted data should be written to **outlen*. If*out* is NULL then the maximum length of the encrypted data should be written to **outlen*.

Decryption Functions

OSSL_FUNC_asym_cipher_decrypt_init() initialises a context for an asymmetric decryption given a provider side asymmetric cipher context in the *ctx* parameter, and a pointer to a provider key object in the *provkey* parameter. The*par ams*, if not NULL, should be set on the context in a manner similar to using **OSSL_FUNC_asym_cipher_set_ctx_params()**. The key object should have been previously generated, loaded or imported into the provider using the key management (OSSL_OP_KEYMGMT) operation (see

provider−keymgmt (7)>.

OSSL_FUNC_asym_cipher_decrypt() performs the actual decryption itself. A previously initialised asymmetric cipher context is passed in the *ctx* parameter. The data to be decrypted is pointed to by the*in* parameter which is *inlen* bytes long. Unless *out* is NULL, the decrypted data should be written to the location pointed to by the *out* parameter and it should not exceed *outsize* bytes in length. The length of the decrypted data should be written to **outlen*. If*out* is NULL then the maximum length of the decrypted data should be written to **outlen*.

Asymmetric Cipher Parameters

See **OSSL_PARAM** (3) for further details on the parameters structure used by the **OSSL_FUNC_asym_cipher_get_ctx_params()** and **OSSL_FUNC_asym_cipher_set_ctx_params()** functions.

OSSL_FUNC_asym_cipher_get_ctx_params() gets asymmetric cipher parameters associated with the given provider side asymmetric cipher context *ctx* and stores them in *params*. Passing NULL for *params* should return true.

OSSL_FUNC_asym_cipher_set_ctx_params() sets the asymmetric cipher parameters associated with the given provider side asymmetric cipher context *ctx* to *params*. Any parameter settings are additional to any that were previously set. Passing NULL for *params* should return true.

Parameters currently recognised by built-in asymmetric cipher algorithms are as follows. Not all parameters are relevant to, or are understood by all asymmetric cipher algorithms:

‘‘pad-mode’’ (**OSSL_ASYM_CIPHER_PARAM_PAD_MODE**) <integer>
The type of padding to be used. The interpretation of this value will depend on the algorithm in use. The default provider understands these RSA padding modes: 1 (RSA_PKCS1_PADDING), 3 (RSA_NO_PADDING), 4 (RSA_PKCS1_OAEP_PADDING), 5 (RSA_X931_PADDING), 6 (RSA_PKCS1_PSS_PADDING) and 7 (RSA_PKCS1_WITH_TLS_PADDING). See **EVP_PKEY_CTX_set_rsa_padding** (3) for further details.

‘‘digest’’ (**OSSL_ASYM_CIPHER_PARAM_OAEP_DIGEST**) <UTF8 string>
Gets or sets the name of the OAEP digest algorithm used when OAEP padding is in use.

‘‘digest’’ (**OSSL_ASYM_CIPHER_PARAM_DIGEST**) <UTF8 string>
Gets or sets the name of the digest algorithm used by the algorithm (where applicable).

‘‘digest-props’’ (**OSSL_ASYM_CIPHER_PARAM_OAEP_DIGEST_PROPS**) <UTF8 string>
Gets or sets the properties to use when fetching the OAEP digest algorithm.

‘‘digest-props’’ (**OSSL_ASYM_CIPHER_PARAM_DIGEST_PROPS**) <UTF8 string>
Gets or sets the properties to use when fetching the cipher digest algorithm.

‘‘mgf1−digest’’ (**OSSL_ASYM_CIPHER_PARAM_MGF1_DIGEST**) <UTF8 string>
Gets or sets the name of the MGF1 digest algorithm used when OAEP or PSS padding is in use.

‘‘mgf1−digest−props’’ (**OSSL_ASYM_CIPHER_PARAM_MGF1_DIGEST_PROPS**) <UTF8 string>
Gets or sets the properties to use when fetching the MGF1 digest algorithm.

‘‘oaep-label’’ (**OSSL_ASYM_CIPHER_PARAM_OAEP_LABEL**) <octet string>
Gets or sets the OAEP label used when OAEP padding is in use.

‘‘tls-client-version’’ (**OSSL_ASYM_CIPHER_PARAM_TLS_CLIENT_VERSION**) <unsigned integer>
The TLS protocol version first requested by the client. See **RSA_PKCS1_WITH_TLS_PADDING** on the page **EVP_PKEY_CTX_set_rsa_padding** (3).

‘‘tls-negotiated-version’’ (**OSSL_ASYM_CIPHER_PARAM_TLS_CLIENT_VERSION**) <unsigned integer>
The negotiated TLS protocol version. See **RSA_PKCS1_WITH_TLS_PADDING** on the page **EVP_PKEY_CTX_set_rsa_padding** (3).

OSSL_FUNC_asym_cipher_gettable_ctx_params() and **OSSL_FUNC_asym_cipher_settable_ctx_params()** get a constant **OSSL_PARAM** array that describes the gettable and settable parameters, i.e. parameters that can be used with

OSSL_FUNC_asym_cipherget_ctx_params() and **OSSL_FUNC_asym_cipher_set_ctx_params()** respectively. See **OSSL_PARAM** (3) for the use of **OSSL_PARAM** as parameter descriptor.

RETURN VALUES

OSSL_FUNC_asym_cipher_newctx() and **OSSL_FUNC_asym_cipher_dupctx()** should return the newly created provider side asymmetric cipher context, or NULL on failure.

All other functions should return 1 for success or 0 on error.

SEE ALSO

provider (7)

HISTORY

The provider ASYM_CIPHER interface was introduced in OpenSSL 3.0.

COPYRIGHT