NAME

provider-keymgmt - The KEYMGMT 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.
 */

/* Key object (keydata) creation and destruction */
void *OSSL_FUNC_keymgmt_new(void *provctx);
void OSSL_FUNC_keymgmt_free(void *keydata);

/* Generation, a more complex constructor */
void *OSSL_FUNC_keymgmt_gen_init(void *provctx, int selection,
                                 const OSSL_PARAM params[]);
int OSSL_FUNC_keymgmt_gen_set_template(void *genctx, void *template);
int OSSL_FUNC_keymgmt_gen_set_params(void *genctx, const OSSL_PARAM params[]);
const OSSL_PARAM *OSSL_FUNC_keymgmt_gen_settable_params(void *genctx,
                                                        void *provctx);
void *OSSL_FUNC_keymgmt_gen(void *genctx, OSSL_CALLBACK *cb, void *cbarg);
void OSSL_FUNC_keymgmt_gen_cleanup(void *genctx);

/* Key loading by object reference, also a constructor */
void *OSSL_FUNC_keymgmt_load(const void *reference, size_t *reference_sz);

/* Key object information */
int OSSL_FUNC_keymgmt_get_params(void *keydata, OSSL_PARAM params[]);
const OSSL_PARAM *OSSL_FUNC_keymgmt_gettable_params(void *provctx);
int OSSL_FUNC_keymgmt_set_params(void *keydata, const OSSL_PARAM params[]);
const OSSL_PARAM *OSSL_FUNC_keymgmt_settable_params(void *provctx);

/* Key object content checks */
int OSSL_FUNC_keymgmt_has(const void *keydata, int selection);
int OSSL_FUNC_keymgmt_match(const void *keydata1, const void *keydata2,
                            int selection);
```

```
/* Discovery of supported operations */
const char *OSSL_FUNC_keymgmt_query_operation_name(int operation_id);

/* Key object import and export functions */
int OSSL_FUNC_keymgmt_import(void *keydata, int selection, const OSSL_PARAM params[]);
const OSSL_PARAM *OSSL_FUNC_keymgmt_import_types(int selection);
int OSSL_FUNC_keymgmt_export(void *keydata, int selection,
                             OSSL_CALLBACK *param_cb, void *cbarg);
const OSSL_PARAM *OSSL_FUNC_keymgmt_export_types(int selection);

/* Key object duplication, a constructor */
void *OSSL_FUNC_keymgmt_dup(const void *keydata_from, int selection);

/* Key object validation */
int OSSL_FUNC_keymgmt_validate(const void *keydata, int selection, int checktype);
```

DESCRIPTION

The KEYMGMT operation doesn't have much public visibility in OpenSSL libraries, it's rather an internal operation that's designed to work in tandem with operations that use private/public key pairs.

Because the KEYMGMT operation shares knowledge with the operations it works with in tandem, they must belong to the same provider. The OpenSSL libraries will ensure that they do.

The primary responsibility of the KEYMGMT operation is to hold the provider side key data for the OpenSSL library EVP_PKEY structure.

All "functions" mentioned here are passed as function pointers between *libcrypto* and the provider in **OSSL_DISPATCH**(3) arrays via **OSSL_ALGORITHM**(3) 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 a **OSSL_DISPATCH**(3) element named **OSSL_FUNC_{name}**. For example, the "function" **OSSL_FUNC_keymgmt_new()** has these:

```
typedef void *(OSSL_FUNC_keymgmt_new_fn)(void *provctx);
static ossl_inline OSSL_FUNC_keymgmt_new_fn
    OSSL_FUNC_keymgmt_new(const OSSL_DISPATCH *opf);
```

OSSL_DISPATCH(3) arrays are indexed by numbers that are provided as macros in **openssl-core_dispatch.h**(7), as follows:

OSSL_FUNC_keymgmt_new	OSSL_FUNC_KEYMGMT_NEW
OSSL_FUNC_keymgmt_free	OSSL_FUNC_KEYMGMT_FREE
OSSL_FUNC_keymgmt_gen_init	OSSL_FUNC_KEYMGMT_GEN_INIT
OSSL_FUNC_keymgmt_gen_set_template	OSSL_FUNC_KEYMGMT_GEN_SET_TEMPLATE
OSSL_FUNC_keymgmt_gen_set_params	OSSL_FUNC_KEYMGMT_GEN_SET_PARAMS
OSSL_FUNC_keymgmt_gen_settable_params	OSSL_FUNC_KEYMGMT_GEN_SETTABLE_PARAMS
OSSL_FUNC_keymgmt_gen	OSSL_FUNC_KEYMGMT_GEN
OSSL_FUNC_keymgmt_gen_cleanup	OSSL_FUNC_KEYMGMT_GEN_CLEANUP
OSSL_FUNC_keymgmt_load	OSSL_FUNC_KEYMGMT_LOAD
OSSL_FUNC_keymgmt_get_params	OSSL_FUNC_KEYMGMT_GET_PARAMS
OSSL_FUNC_keymgmt_gettable_params	OSSL_FUNC_KEYMGMT_GETTABLE_PARAMS
OSSL_FUNC_keymgmt_set_params	OSSL_FUNC_KEYMGMT_SET_PARAMS
OSSL_FUNC_keymgmt_settable_params	OSSL_FUNC_KEYMGMT_SETTABLE_PARAMS
OSSL_FUNC_keymgmt_query_operation_name	OSSL_FUNC_KEYMGMT_QUERY_OPERATION_NAME
OSSL_FUNC_keymgmt_has	OSSL_FUNC_KEYMGMT_HAS
OSSL_FUNC_keymgmt_validate	OSSL_FUNC_KEYMGMT_VALIDATE
OSSL_FUNC_keymgmt_match	OSSL_FUNC_KEYMGMT_MATCH
OSSL_FUNC_keymgmt_import	OSSL_FUNC_KEYMGMT_IMPORT
OSSL_FUNC_keymgmt_import_types	OSSL_FUNC_KEYMGMT_IMPORT_TYPES
OSSL_FUNC_keymgmt_export	OSSL_FUNC_KEYMGMT_EXPORT
OSSL_FUNC_keymgmt_export_types	OSSL_FUNC_KEYMGMT_EXPORT_TYPES
OSSL_FUNC_keymgmt_dup	OSSL_FUNC_KEYMGMT_DUP

Key Objects

A key object is a collection of data for an asymmetric key, and is represented as *keydata* in this manual.

The exact contents of a key object are defined by the provider, and it is assumed that different operations in one and the same provider use the exact same structure to represent this collection of data, so that for example, a key object that has been created using the KEYMGMT interface that we document here can be passed as is to other provider operations, such as **OP_signature_sign_init()** (see **provider-signature**(7)).

With some of the KEYMGMT functions, it’s possible to select a specific subset of data to handle,

governed by the bits in a *selection* indicator. The bits are:

OSSL_KEYMGMT_SELECT_PRIVATE_KEY
Indicating that the private key data in a key object should be considered.

OSSL_KEYMGMT_SELECT_PUBLIC_KEY
Indicating that the public key data in a key object should be considered.

OSSL_KEYMGMT_SELECT_DOMAIN_PARAMETERS
Indicating that the domain parameters in a key object should be considered.

OSSL_KEYMGMT_SELECT_OTHER_PARAMETERS
Indicating that other parameters in a key object should be considered.

Other parameters are key parameters that don't fit any other classification. In other words, this particular selector bit works as a last resort bit bucket selector.

Some selector bits have also been combined for easier use:

OSSL_KEYMGMT_SELECT_ALL_PARAMETERS
Indicating that all key object parameters should be considered, regardless of their more granular classification.

This is a combination of **OSSL_KEYMGMT_SELECT_DOMAIN_PARAMETERS** and **OSSL_KEYMGMT_SELECT_OTHER_PARAMETERS**.

OSSL_KEYMGMT_SELECT_KEYPAIR
Indicating that both the whole key pair in a key object should be considered, i.e. the combination of public and private key.

This is a combination of **OSSL_KEYMGMT_SELECT_PRIVATE_KEY** and **OSSL_KEYMGMT_SELECT_PUBLIC_KEY**.

OSSL_KEYMGMT_SELECT_ALL
Indicating that everything in a key object should be considered.

The exact interpretation of those bits or how they combine is left to each function where you can specify a selector.

It's left to the provider implementation to decide what is reasonable to do with regards to received

selector bits and how to do it. Among others, an implementation of **OSSL_FUNC_keymgmt_match()** might opt to not compare the private half if it has compared the public half, since a match of one half implies a match of the other half.

Constructing and Destructing Functions

OSSL_FUNC_keymgmt_new() should create a provider side key object. The provider context *provctx* is passed and may be incorporated in the key object, but that is not mandatory.

OSSL_FUNC_keymgmt_free() should free the passed *keydata*.

OSSL_FUNC_keymgmt_gen_init(), **OSSL_FUNC_keymgmt_gen_set_template()**, **OSSL_FUNC_keymgmt_gen_set_params()**, **OSSL_FUNC_keymgmt_gen_settable_params()**, **OSSL_FUNC_keymgmt_gen()** and **OSSL_FUNC_keymgmt_gen_cleanup()** work together as a more elaborate context based key object constructor.

OSSL_FUNC_keymgmt_gen_init() should create the key object generation context and initialize it with *selections*, which will determine what kind of contents the key object to be generated should get. The *params*, if not NULL, should be set on the context in a manner similar to using **OSSL_FUNC_keymgmt_set_params()**.

OSSL_FUNC_keymgmt_gen_set_template() should add *template* to the context *genctx*. The *template* is assumed to be a key object constructed with the same KEYMGMT, and from which content that the implementation chooses can be used as a template for the key object to be generated. Typically, the generation of a DSA or DH key would get the domain parameters from this *template*.

OSSL_FUNC_keymgmt_gen_set_params() should set additional parameters from *params* in the key object generation context *genctx*.

OSSL_FUNC_keymgmt_gen_settable_params() should return a constant array of descriptor **OSSL_PARAM**(3), for parameters that **OSSL_FUNC_keymgmt_gen_set_params()** can handle.

OSSL_FUNC_keymgmt_gen() should perform the key object generation itself, and return the result. The callback *cb* should be called at regular intervals with indications on how the key object generation progresses.

OSSL_FUNC_keymgmt_gen_cleanup() should clean up and free the key object generation context *genctx*

OSSL_FUNC_keymgmt_load() creates a provider side key object based on a *reference* object with a size of *reference_sz* bytes, that only the provider knows how to interpret, but that may come from other

operations. Outside the provider, this reference is simply an array of bytes.

At least one of **OSSL_FUNC_keymgmt_new()**, **OSSL_FUNC_keymgmt_gen()** and **OSSL_FUNC_keymgmt_load()** are mandatory, as well as **OSSL_FUNC_keymgmt_free()** and **OSSL_FUNC_keymgmt_has()**. Additionally, if **OSSL_FUNC_keymgmt_gen()** is present, **OSSL_FUNC_keymgmt_gen_init()** and **OSSL_FUNC_keymgmt_gen_cleanup()** must be present as well.

Key Object Information Functions

OSSL_FUNC_keymgmt_get_params() should extract information data associated with the given *keydata*, see "Common Information Parameters".

OSSL_FUNC_keymgmt_gettable_params() should return a constant array of descriptor **OSSL_PARAM**(3), for parameters that **OSSL_FUNC_keymgmt_get_params()** can handle.

If **OSSL_FUNC_keymgmt_gettable_params()** is present, **OSSL_FUNC_keymgmt_get_params()** must also be present, and vice versa.

OSSL_FUNC_keymgmt_set_params() should update information data associated with the given *keydata*, see "Common Information Parameters".

OSSL_FUNC_keymgmt_settable_params() should return a constant array of descriptor **OSSL_PARAM**(3), for parameters that **OSSL_FUNC_keymgmt_set_params()** can handle.

If **OSSL_FUNC_keymgmt_settable_params()** is present, **OSSL_FUNC_keymgmt_set_params()** must also be present, and vice versa.

Key Object Checking Functions

OSSL_FUNC_keymgmt_query_operation_name() should return the name of the supported algorithm for the operation *operation_id*. This is similar to **provider_query_operation()** (see **provider-base**(7)), but only works as an advisory. If this function is not present, or returns NULL, the caller is free to assume that there's an algorithm from the same provider, of the same name as the one used to fetch the keymgmt and try to use that.

OSSL_FUNC_keymgmt_has() should check whether the given *keydata* contains the subsets of data indicated by the *selector*. A combination of several selector bits must consider all those subsets, not just one. An implementation is, however, free to consider an empty subset of data to still be a valid subset. For algorithms where some selection is not meaningful such as **OSSL_KEYMGMT_SELECT_DOMAIN_PARAMETERS** for RSA keys the function should just return 1 as the selected subset is not really missing in the key.

OSSL_FUNC_keymgmt_validate() should check if the *keydata* contains valid data subsets indicated by *selection*. Some combined selections of data subsets may cause validation of the combined data. For example, the combination of **OSSL_KEYMGMT_SELECT_PRIVATE_KEY** and **OSSL_KEYMGMT_SELECT_PUBLIC_KEY** (or **OSSL_KEYMGMT_SELECT_KEYPAIR** for short) is expected to check that the pairwise consistency of *keydata* is valid. The *checktype* parameter controls what type of check is performed on the subset of data. Two types of check are defined: **OSSL_KEYMGMT_VALIDATE_FULL_CHECK** and **OSSL_KEYMGMT_VALIDATE_QUICK_CHECK**. The interpretation of how much checking is performed in a full check versus a quick check is key type specific. Some providers may have no distinction between a full check and a quick check. For algorithms where some selection is not meaningful such as **OSSL_KEYMGMT_SELECT_DOMAIN_PARAMETERS** for RSA keys the function should just return 1 as there is nothing to validate for that selection.

OSSL_FUNC_keymgmt_match() should check if the data subset indicated by *selection* in *keydata1* and *keydata2* match. It is assumed that the caller has ensured that *keydata1* and *keydata2* are both owned by the implementation of this function.

Key Object Import, Export and Duplication Functions

OSSL_FUNC_keymgmt_import() should import data indicated by *selection* into *keydata* with values taken from the **OSSL_PARAM**(3) array *params*.

OSSL_FUNC_keymgmt_export() should extract values indicated by *selection* from *keydata*, create an **OSSL_PARAM**(3) array with them and call *param_cb* with that array as well as the given *cbarg*.

OSSL_FUNC_keymgmt_import_types() should return a constant array of descriptor **OSSL_PARAM**(3) for data indicated by *selection*, for parameters that **OSSL_FUNC_keymgmt_import()** can handle.

OSSL_FUNC_keymgmt_export_types() should return a constant array of descriptor **OSSL_PARAM**(3) for data indicated by *selection*, that the **OSSL_FUNC_keymgmt_export()** callback can expect to receive.

OSSL_FUNC_keymgmt_dup() should duplicate data subsets indicated by *selection* or the whole key data *keydata_from* and create a new provider side key object with the data.

Common Information Parameters

See **OSSL_PARAM**(3) for further details on the parameters structure.

Common information parameters currently recognised by all built-in keymgmt algorithms are as follows:

"bits" (**OSSL_PKEY_PARAM_BITS**) <integer>
The value should be the cryptographic length of the cryptosystem to which the key belongs, in bits. The definition of cryptographic length is specific to the key cryptosystem.

"max-size" (**OSSL_PKEY_PARAM_MAX_SIZE**) <integer>
The value should be the maximum size that a caller should allocate to safely store a signature (called *sig* in **provider-signature**(7)), the result of asymmmetric encryption / decryption (*out* in **provider-asym_cipher**(7), a derived secret (*secret* in **provider-keyexch**(7), and similar data).

Because an EVP_KEYMGMT method is always tightly bound to another method (signature, asymmetric cipher, key exchange, ...) and must be of the same provider, this number only needs to be synchronised with the dimensions handled in the rest of the same provider.

"security-bits" (**OSSL_PKEY_PARAM_SECURITY_BITS**) <integer>
The value should be the number of security bits of the given key. Bits of security is defined in SP800-57.

"mandatory-digest" (**OSSL_PKEY_PARAM_MANDATORY_DIGEST**) <UTF8 string>
If there is a mandatory digest for performing a signature operation with keys from this keymgmt, this parameter should get its name as value.

When **EVP_PKEY_get_default_digest_name()** queries this parameter and it’s filled in by the implementation, its return value will be 2.

If the keymgmt implementation fills in the value "" or "UNDEF", **EVP_PKEY_get_default_digest_name**(3) will place the string "UNDEF" into its argument *mdname*. This signifies that no digest should be specified with the corresponding signature operation.

"default-digest" (**OSSL_PKEY_PARAM_DEFAULT_DIGEST**) <UTF8 string>
If there is a default digest for performing a signature operation with keys from this keymgmt, this parameter should get its name as value.

When **EVP_PKEY_get_default_digest_name**(3) queries this parameter and it’s filled in by the implementation, its return value will be 1. Note that if **OSSL_PKEY_PARAM_MANDATORY_DIGEST** is responded to as well, **EVP_PKEY_get_default_digest_name**(3) ignores the response to this parameter.

If the keymgmt implementation fills in the value "" or "UNDEF", **EVP_PKEY_get_default_digest_name**(3) will place the string "UNDEF" into its argument

mdname. This signifies that no digest has to be specified with the corresponding signature operation, but may be specified as an option.

RETURN VALUES

OSSL_FUNC_keymgmt_new() and **OSSL_FUNC_keymgmt_dup()** should return a valid reference to the newly created provider side key object, or NULL on failure.

OSSL_FUNC_keymgmt_import(), **OSSL_FUNC_keymgmt_export()**, **OSSL_FUNC_keymgmt_get_params()** and **OSSL_FUNC_keymgmt_set_params()** should return 1 for success or 0 on error.

OSSL_FUNC_keymgmt_validate() should return 1 on successful validation, or 0 on failure.

OSSL_FUNC_keymgmt_has() should return 1 if all the selected data subsets are contained in the given *keydata* or 0 otherwise.

OSSL_FUNC_keymgmt_query_operation_name() should return a pointer to a string matching the requested operation, or NULL if the same name used to fetch the keymgmt applies.

OSSL_FUNC_keymgmt_gettable_params() and **OSSL_FUNC_keymgmt_settable_params()** **OSSL_FUNC_keymgmt_import_types()**, **OSSL_FUNC_keymgmt_export_types()** should always return a constant **OSSL_PARAM**(3) array.

SEE ALSO

provider(7), **EVP_PKEY-X25519**(7), **EVP_PKEY-X448**(7), **EVP_PKEY-ED25519**(7), **EVP_PKEY-ED448**(7), **EVP_PKEY-EC**(7), **EVP_PKEY-RSA**(7), **EVP_PKEY-DSA**(7), **EVP_PKEY-DH**(7)

HISTORY

The KEYMGMT interface was introduced in OpenSSL 3.0.

COPYRIGHT