NAME

provider−signature − The signature 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_signature_newctx(void *provctx, const char *propq);
void OSSL_FUNC_signature_freectx(void *ctx);
void *OSSL_FUNC_signature_dupctx(void *ctx);

/* Signing */
int OSSL_FUNC_signature_sign_init(void *ctx, void *provkey,
                                  const OSSL_PARAM params[]);
int OSSL_FUNC_signature_sign(void *ctx, unsigned char *sig, size_t *siglen,
                             size_t sigsize, const unsigned char *tbs, size_t tb

/* Verifying */
int OSSL_FUNC_signature_verify_init(void *ctx, void *provkey,
                                    const OSSL_PARAM params[]);
int OSSL_FUNC_signature_verify(void *ctx, const unsigned char *sig, size_t sigle
                               const unsigned char *tbs, size_t tbslen);

/* Verify Recover */
int OSSL_FUNC_signature_verify_recover_init(void *ctx, void *provkey,
                                            const OSSL_PARAM params[]);
int OSSL_FUNC_signature_verify_recover(void *ctx, unsigned char *rout,
                                       size_t *routlen, size_t routsize,
                                       const unsigned char *sig, size_t siglen);

/* Digest Sign */
int OSSL_FUNC_signature_digest_sign_init(void *ctx, const char *mdname,
                                         const char *props, void *provkey,
                                         const OSSL_PARAM params[]);
int OSSL_FUNC_signature_digest_sign_update(void *ctx, const unsigned char *data,
                                           size_t datalen);
int OSSL_FUNC_signature_digest_sign_final(void *ctx, unsigned char *sig,
                                          size_t *siglen, size_t sigsize);
int OSSL_FUNC_signature_digest_sign(void *ctx,
                                    unsigned char *sigret, size_t *siglen,
                                    size_t sigsize, const unsigned char *tbs,
                                    size_t tbslen);

/* Digest Verify */
int OSSL_FUNC_signature_digest_verify_init(void *ctx, const char *mdname,
                                           const char *props, void *provkey,
                                           const OSSL_PARAM params[]);
int OSSL_FUNC_signature_digest_verify_update(void *ctx,
```

```
                                                const unsigned char *data,
                                                size_t datalen);
int OSSL_FUNC_signature_digest_verify_final(void *ctx, const unsigned char *sig,
                                            size_t siglen);
int OSSL_FUNC_signature_digest_verify(void *ctx, const unsigned char *sig,
                                      size_t siglen, const unsigned char *tbs,
                                      size_t tbslen);

/* Signature parameters */
int OSSL_FUNC_signature_get_ctx_params(void *ctx, OSSL_PARAM params[]);
const OSSL_PARAM *OSSL_FUNC_signature_gettable_ctx_params(void *ctx,
                                                          void *provctx);
int OSSL_FUNC_signature_set_ctx_params(void *ctx, const OSSL_PARAM params[]);
const OSSL_PARAM *OSSL_FUNC_signature_settable_ctx_params(void *ctx,
                                                          void *provctx);
/* MD parameters */
int OSSL_FUNC_signature_get_ctx_md_params(void *ctx, OSSL_PARAM params[]);
const OSSL_PARAM * OSSL_FUNC_signature_gettable_ctx_md_params(void *ctx);
int OSSL_FUNC_signature_set_ctx_md_params(void *ctx, const OSSL_PARAM params[]);
const OSSL_PARAM * OSSL_FUNC_signature_settable_ctx_md_params(void *ctx);
```

DESCRIPTION

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

The signature (OSSL_OP_SIGNATURE) operation enables providers to implement signature algorithms and make them available to applications via the API functions **EVP_PKEY_sign** (3), **EVP_PKEY_verify** (3), and **EVP_PKEY_verify_recover** (3) (as well as 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_signature_newctx()** has these:

```
typedef void *(OSSL_FUNC_signature_newctx_fn)(void *provctx, const char *propq);
static ossl_inline OSSL_FUNC_signature_newctx_fn
    OSSL_FUNC_signature_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_signature_newctx                OSSL_FUNC_SIGNATURE_NEWCTX
OSSL_FUNC_signature_freectx               OSSL_FUNC_SIGNATURE_FREECTX
OSSL_FUNC_signature_dupctx                OSSL_FUNC_SIGNATURE_DUPCTX

OSSL_FUNC_signature_sign_init             OSSL_FUNC_SIGNATURE_SIGN_INIT
OSSL_FUNC_signature_sign                  OSSL_FUNC_SIGNATURE_SIGN

OSSL_FUNC_signature_verify_init           OSSL_FUNC_SIGNATURE_VERIFY_INIT
OSSL_FUNC_signature_verify                OSSL_FUNC_SIGNATURE_VERIFY

OSSL_FUNC_signature_verify_recover_init   OSSL_FUNC_SIGNATURE_VERIFY_RECOVER_IN
OSSL_FUNC_signature_verify_recover        OSSL_FUNC_SIGNATURE_VERIFY_RECOVER

OSSL_FUNC_signature_digest_sign_init      OSSL_FUNC_SIGNATURE_DIGEST_SIGN_INIT
OSSL_FUNC_signature_digest_sign_update    OSSL_FUNC_SIGNATURE_DIGEST_SIGN_UPDAT
OSSL_FUNC_signature_digest_sign_final     OSSL_FUNC_SIGNATURE_DIGEST_SIGN_FINAL
```

```
OSSL_FUNC_signature_digest_sign             OSSL_FUNC_SIGNATURE_DIGEST_SIGN

OSSL_FUNC_signature_digest_verify_init      OSSL_FUNC_SIGNATURE_DIGEST_VERIFY_INI
OSSL_FUNC_signature_digest_verify_update    OSSL_FUNC_SIGNATURE_DIGEST_VERIFY_UPD
OSSL_FUNC_signature_digest_verify_final     OSSL_FUNC_SIGNATURE_DIGEST_VERIFY_FIN
OSSL_FUNC_signature_digest_verify           OSSL_FUNC_SIGNATURE_DIGEST_VERIFY

OSSL_FUNC_signature_get_ctx_params          OSSL_FUNC_SIGNATURE_GET_CTX_PARAMS
OSSL_FUNC_signature_gettable_ctx_params     OSSL_FUNC_SIGNATURE_GETTABLE_CTX_PARA
OSSL_FUNC_signature_set_ctx_params          OSSL_FUNC_SIGNATURE_SET_CTX_PARAMS
OSSL_FUNC_signature_settable_ctx_params     OSSL_FUNC_SIGNATURE_SETTABLE_CTX_PARA

OSSL_FUNC_signature_get_ctx_md_params       OSSL_FUNC_SIGNATURE_GET_CTX_MD_PARAMS
OSSL_FUNC_signature_gettable_ctx_md_params OSSL_FUNC_SIGNATURE_GETTABLE_CTX_MD_P
OSSL_FUNC_signature_set_ctx_md_params       OSSL_FUNC_SIGNATURE_SET_CTX_MD_PARAMS
OSSL_FUNC_signature_settable_ctx_md_params OSSL_FUNC_SIGNATURE_SETTABLE_CTX_MD_P
```

A signature algorithm implementation may not implement all of these functions. In order to be a consistent set of functions we must have at least a set of context functions (OSSL_FUNC_signature_newctx and OSSL_FUNC_signature_freectx) as well as a set of ‘‘signature’’ functions, i.e. at least one of:

OSSL_FUNC_signature_sign_init and OSSL_FUNC_signature_sign
OSSL_FUNC_signature_verify_init and OSSL_FUNC_signature_verify
OSSL_FUNC_signature_verify_recover_init and OSSL_FUNC_signature_verify_init
OSSL_FUNC_signature_digest_sign_init, OSSL_FUNC_signature_digest_sign_update and OSSL_FUNC_signature_digest_sign_final
OSSL_FUNC_signature_digest_verify_init, OSSL_FUNC_signature_digest_verify_update and OSSL_FUNC_signature_digest_verify_final
OSSL_FUNC_signature_digest_sign_init and OSSL_FUNC_signature_digest_sign
OSSL_FUNC_signature_digest_verify_init and OSSL_FUNC_signature_digest_verify

OSSL_FUNC_signature_set_ctx_params and OSSL_FUNC_signature_settable_ctx_params are optional, but if one of them is present then the other one must also be present. The same applies to OSSL_FUNC_signature_get_ctx_params and OSSL_FUNC_signature_gettable_ctx_params, as well as the ‘‘md_params’’ functions. The OSSL_FUNC_signature_dupctx function is optional.

A signature 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_signature_newctx() should create and return a pointer to a provider side structure for holding context information during a signature operation. A pointer to this context will be passed back in a number of the other signature operation function calls. The parameter *provctx* is the provider context generated during provider initialisation (see **provider** (7)). The *propq* parameter is a property query string that may be (optionally) used by the provider during any ‘‘fetches’’ that it may perform (if it performs any).

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

OSSL_FUNC_signature_dupctx() should duplicate the provider side signature context in the *ctx* parameter and return the duplicate copy.

Signing Functions

OSSL_FUNC_signature_sign_init() initialises a context for signing given a provider side signature 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_signature_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_signature_sign() performs the actual signing itself. A previously initialised signature context is passed in the *ctx* parameter. The data to be signed is pointed to be the*tbs* parameter which is *tbslen* bytes long. Unless *sig* is NULL, the signature should be written to the location pointed to by the *sig* parameter and it should not exceed *sigsize* bytes in length. The length of the signature should be written to **siglen*. If*sig* is NULL then the maximum length of the signature should be written to **siglen*.

Verify Functions

OSSL_FUNC_signature_verify_init() initialises a context for verifying a signature given a provider side signature context in the *ctx* parameter, and a pointer to a provider key object in the *provkey* parameter. The *params*, if not NULL, should be set on the context in a manner similar to using **OSSL_FUNC_signature_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_signature_verify() performs the actual verification itself. A previously initialised signature context is passed in the *ctx* parameter. The data that the signature covers is pointed to be the *tbs* parameter which is *tbslen* bytes long. The signature is pointed to by the *sig* parameter which is *siglen* bytes long.

Verify Recover Functions

OSSL_FUNC_signature_verify_recover_init() initialises a context for recovering the signed data given a provider side signature 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_signature_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_signature_verify_recover() performs the actual verify recover itself. A previously initialised signature context is passed in the *ctx* parameter. The signature is pointed to by the*sig* parameter which is *siglen* bytes long. Unless *rout* is NULL, the recovered data should be written to the location pointed to by *rout* which should not exceed *routsize* bytes in length. The length of the recovered data should be written to **routlen*. If*r out* is NULL then the maximum size of the output buffer is written to the *routlen* parameter.

Digest Sign Functions

OSSL_FUNC_signature_digeset_sign_init() initialises a context for signing given a provider side signature context in the *ctx* parameter, and a pointer to a provider key object in the *provkey* parameter. The *params*, if not NULL, should be set on the context in a manner similar to using **OSSL_FUNC_signature_set_ctx_params()** and **OSSL_FUNC_signature_set_ctx_md_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)>. The name of the digest to be used will be in the *mdname* parameter. There may also be properties to be used in fetching the digest in the *props* parameter, although this may be ignored by providers.

OSSL_FUNC_signature_digest_sign_update() provides data to be signed in the *data* parameter which should be of length *datalen*. A previously initialised signature context is passed in the *ctx* parameter. This function may be called multiple times to cumulatively add data to be signed.

OSSL_FUNC_signature_digest_sign_final() finalises a signature operation previously started through **OSSL_FUNC_signature_digest_sign_init()** and **OSSL_FUNC_signature_digest_sign_update()** calls. Once finalised no more data will be added through **OSSL_FUNC_signature_digest_sign_update()**. A previously initialised signature context is passed in the *ctx* parameter. Unless *sig* is NULL, the signature should be written to the location pointed to by the *sig* parameter and it should not exceed *sigsize* bytes in length. The length of the signature should be written to **siglen*. If *sig* is NULL then the maximum length of the signature should be written to **siglen*.

OSSL_FUNC_signature_digest_sign() implements a ‘‘one shot’’ digest sign operation previously started through **OSSL_FUNC_signature_digeset_sign_init()**. A previously initialised signature context is passed in the *ctx* parameter. The data to be signed is in *tbs* which should be *tbslen* bytes long. Unless *sig* is NULL, the signature should be written to the location pointed to by the *sig* parameter and it should not exceed

sigsize bytes in length. The length of the signature should be written to **siglen*. If *sig* is NULL then the maximum length of the signature should be written to **siglen*.

Digest Verify Functions

OSSL_FUNC_signature_digeset_verify_init() initialises a context for verifying given a provider side verification context in the *ctx* parameter, and a pointer to a provider key object in the *provkey* parameter. The *params*, if not NULL, should be set on the context in a manner similar to **OSSL_FUNC_signature_set_ctx_params()** and **OSSL_FUNC_signature_set_ctx_md_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)>. The name of the digest to be used will be in the *mdname* parameter. There may also be properties to be used in fetching the digest in the *props* parameter, although this may be ignored by providers.

OSSL_FUNC_signature_digest_verify_update() provides data to be verified in the *data* parameter which should be of length *datalen*. A previously initialised verification context is passed in the *ctx* parameter. This function may be called multiple times to cumulatively add data to be verified.

OSSL_FUNC_signature_digest_verify_final() finalises a verification operation previously started through **OSSL_FUNC_signature_digest_verify_init()** and **OSSL_FUNC_signature_digest_verify_update()** calls. Once finalised no more data will be added through **OSSL_FUNC_signature_digest_verify_update()**. A previously initialised verification context is passed in the *ctx* parameter. The signature to be verified is in *sig* which is *siglen* bytes long.

OSSL_FUNC_signature_digest_verify() implements a ‘‘one shot’’ digest verify operation previously started through **OSSL_FUNC_signature_digeset_verify_init()**. A previously initialised verification context is passed in the *ctx* parameter. The data to be verified is in *tbs* which should be *tbslen* bytes long. The signature to be verified is in *sig* which is *siglen* bytes long.

Signature parameters

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

OSSL_FUNC_signature_get_ctx_params() gets signature parameters associated with the given provider side signature context *ctx* and stored them in *params*. Passing NULL for *params* should return true.

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

Common parameters currently recognised by built-in signature algorithms are as follows.

‘‘digest’’ (**OSSL_SIGNATURE_PARAM_DIGEST**) <UTF8 string>

Get or sets the name of the digest algorithm used for the input to the signature functions. It is required in order to calculate the ‘‘algorithm-id’’.

‘‘properties’’ (**OSSL_SIGNATURE_PARAM_PROPERTIES**) <UTF8 string>

Sets the name of the property query associated with the ‘‘digest’’ algorithm. NULL is used if this optional value is not set.

‘‘digest-size’’ (**OSSL_SIGNATURE_PARAM_DIGEST_SIZE**) <unsigned integer>

Gets or sets the output size of the digest algorithm used for the input to the signature functions. The length of the ‘‘digest-size’’ parameter should not exceed that of a **size_t**.

‘‘algorithm-id’’ (**OSSL_SIGNATURE_PARAM_ALGORITHM_ID**) <octet string>

Gets the DER encoded AlgorithmIdentifier that corresponds to the combination of signature algorithm and digest algorithm for the signature operation.

‘‘kat’’ (**OSSL_SIGNATURE_PARAM_KAT**) <unsigned integer>

Sets a flag to modify the sign operation to return an error if the initial calculated signature is invalid. In the normal mode of operation − new random values are chosen until the signature operation succeeds. By default it retries until a signature is calculated. Setting the value to 0 causes the sign operation to retry, otherwise the sign operation is only tried once and returns whether or not it was

successful. Known answer tests can be performed if the random generator is overridden to supply known values that either pass or fail.

OSSL_FUNC_signature_gettable_ctx_params() and **OSSL_FUNC_signature_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_signature_get_ctx_params()** and **OSSL_FUNC_signature_set_ctx_params()** respectively. See **OSSL_PARAM** (3) for the use of **OSSL_PARAM** as parameter descriptor.

MD parameters

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

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

OSSL_FUNC_signature_set_ms_ctx_params() sets the digest parameters associated with the given provider side digest signature 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 signature algorithms are the same as those for built-in digest algorithms. See ''Digest Parameters'' in **provider−digest** (7) for further information.

OSSL_FUNC_signature_gettable_md_ctx_params() and **OSSL_FUNC_signature_settable_md_ctx_params()** get a constant **OSSL_PARAM** array that describes the gettable and settable digest parameters, i.e. parameters that can be used with **OSSL_FUNC_signature_get_md_ctx_params()** and **OSSL_FUNC_signature_set_md_ctx_params()** respectively. See **OSSL_PARAM** (3) for the use of **OSSL_PARAM** as parameter descriptor.

RETURN VALUES

OSSL_FUNC_signature_newctx() and **OSSL_FUNC_signature_dupctx()** should return the newly created provider side signature, or NULL on failure.

OSSL_FUNC_signature_gettable_ctx_params(), **OSSL_FUNC_signature_settable_ctx_params()**, **OSSL_FUNC_signature_gettable_md_ctx_params()** and **OSSL_FUNC_signature_settable_md_ctx_params()**, return the gettable or settable parameters in a constant **OSSL_PARAM** array.

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

SEE ALSO

provider (7)

HISTORY

The provider SIGNATURE interface was introduced in OpenSSL 3.0.

COPYRIGHT