NAME

EVP_KDF, EVP_KDF_fetch, EVP_KDF_free, EVP_KDF_up_ref, EVP_KDF_CTX, EVP_KDF_CTX_new, EVP_KDF_CTX_free, EVP_KDF_CTX_dup, EVP_KDF_CTX_reset, EVP_KDF_derive, EVP_KDF_CTX_get_kdf_size, EVP_KDF_get0_provider, EVP_KDF_CTX_kdf, EVP_KDF_is_a, EVP_KDF_get0_name, EVP_KDF_names_do_all, EVP_KDF_get0_description, EVP_KDF_CTX_get_params, EVP_KDF_CTX_set_params, EVP_KDF_do_all_provided, EVP_KDF_get_params, EVP_KDF_gettable_params, EVP_KDF_gettable_ctx_params, EVP_KDF_settable_ctx_params, EVP_KDF_CTX_gettable_params, EVP_KDF_CTX_settable_params - EVP KDF routines

SYNOPSIS

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

typedef struct evp_kdf_st EVP_KDF;
typedef struct evp_kdf_ctx_st EVP_KDF_CTX;

EVP_KDF_CTX *EVP_KDF_CTX_new(const EVP_KDF *kdf);
const EVP_KDF *EVP_KDF_CTX_kdf(EVP_KDF_CTX *ctx);
void EVP_KDF_CTX_free(EVP_KDF_CTX *ctx);
EVP_KDF_CTX *EVP_KDF_CTX_dup(const EVP_KDF_CTX *src);
void EVP_KDF_CTX_reset(EVP_KDF_CTX *ctx);
size_t EVP_KDF_CTX_get_kdf_size(EVP_KDF_CTX *ctx);
int EVP_KDF_derive(EVP_KDF_CTX *ctx, unsigned char *key, size_t keylen,
                   const OSSL_PARAM params[]);
int EVP_KDF_up_ref(EVP_KDF *kdf);
void EVP_KDF_free(EVP_KDF *kdf);
EVP_KDF *EVP_KDF_fetch(OSSL_LIB_CTX *libctx, const char *algorithm,
                       const char *properties);
int EVP_KDF_is_a(const EVP_KDF *kdf, const char *name);
const char *EVP_KDF_get0_name(const EVP_KDF *kdf);
const char *EVP_KDF_get0_description(const EVP_KDF *kdf);
const OSSL_PROVIDER *EVP_KDF_get0_provider(const EVP_KDF *kdf);
void EVP_KDF_do_all_provided(OSSL_LIB_CTX *libctx,
                             void (*fn)(EVP_KDF *kdf, void *arg),
                             void *arg);
int EVP_KDF_names_do_all(const EVP_KDF *kdf,
                         void (*fn)(const char *name, void *data),
                         void *data);
int EVP_KDF_get_params(EVP_KDF *kdf, OSSL_PARAM params[]);
int EVP_KDF_CTX_get_params(EVP_KDF_CTX *ctx, OSSL_PARAM params[]);
```

```
int EVP_KDF_CTX_set_params(EVP_KDF_CTX *ctx, const OSSL_PARAM params[]);
const OSSL_PARAM *EVP_KDF_gettable_params(const EVP_KDF *kdf);
const OSSL_PARAM *EVP_KDF_gettable_ctx_params(const EVP_KDF *kdf);
const OSSL_PARAM *EVP_KDF_settable_ctx_params(const EVP_KDF *kdf);
const OSSL_PARAM *EVP_KDF_CTX_gettable_params(const EVP_KDF *kdf);
const OSSL_PARAM *EVP_KDF_CTX_settable_params(const EVP_KDF *kdf);
const OSSL_PROVIDER *EVP_KDF_get0_provider(const EVP_KDF *kdf);
```

DESCRIPTION

The EVP KDF routines are a high-level interface to Key Derivation Function algorithms and should be used instead of algorithm-specific functions.

After creating a **EVP_KDF_CTX** for the required algorithm using **EVP_KDF_CTX_new()**, inputs to the algorithm are supplied either by passing them as part of the **EVP_KDF_derive()** call or using calls to **EVP_KDF_CTX_set_params()** before calling **EVP_KDF_derive()** to derive the key.

Types

EVP_KDF is a type that holds the implementation of a KDF.

EVP_KDF_CTX is a context type that holds the algorithm inputs.

Algorithm implementation fetching

EVP_KDF_fetch() fetches an implementation of a KDF *algorithm*, given a library context *libctx* and a set of *properties*. See "ALGORITHM FETCHING" in **crypto**(7) for further information.

See "Key Derivation Function (KDF)" in **OSSL_PROVIDER-default**(7) for the lists of algorithms supported by the default provider.

The returned value must eventually be freed with **EVP_KDF_free**(3).

EVP_KDF_up_ref() increments the reference count of an already fetched KDF.

EVP_KDF_free() frees a fetched algorithm. NULL is a valid parameter, for which this function is a no-op.

Context manipulation functions

EVP_KDF_CTX_new() creates a new context for the KDF implementation *kdf*.

EVP_KDF_CTX_free() frees up the context *ctx*. If *ctx* is NULL, nothing is done.

EVP_KDF_CTX_kdf() returns the **EVP_KDF** associated with the context *ctx*.

Computing functions

EVP_KDF_CTX_reset() resets the context to the default state as if the context had just been created.

EVP_KDF_derive() processes any parameters in *Params* and then derives *keylen* bytes of key material and places it in the *key* buffer. If the algorithm produces a fixed amount of output then an error will occur unless the *keylen* parameter is equal to that output size, as returned by **EVP_KDF_CTX_get_kdf_size()**.

EVP_KDF_get_params() retrieves details about the implementation *kdf*. The set of parameters given with *params* determine exactly what parameters should be retrieved. Note that a parameter that is unknown in the underlying context is simply ignored.

EVP_KDF_CTX_get_params() retrieves chosen parameters, given the context *ctx* and its underlying context. The set of parameters given with *params* determine exactly what parameters should be retrieved. Note that a parameter that is unknown in the underlying context is simply ignored.

EVP_KDF_CTX_set_params() passes chosen parameters to the underlying context, given a context *ctx*. The set of parameters given with *params* determine exactly what parameters are passed down. Note that a parameter that is unknown in the underlying context is simply ignored. Also, what happens when a needed parameter isn't passed down is defined by the implementation.

EVP_KDF_gettable_params() returns an **OSSL_PARAM**(3) array that describes the retrievable and settable parameters. **EVP_KDF_gettable_params()** returns parameters that can be used with **EVP_KDF_get_params()**.

EVP_KDF_gettable_ctx_params() and **EVP_KDF_CTX_gettable_params()** return constant **OSSL_PARAM**(3) arrays that describe the retrievable parameters that can be used with **EVP_KDF_CTX_get_params()**. **EVP_KDF_gettable_ctx_params()** returns the parameters that can be retrieved from the algorithm, whereas **EVP_KDF_CTX_gettable_params()** returns the parameters that can be retrieved in the context's current state.

EVP_KDF_settable_ctx_params() and **EVP_KDF_CTX_settable_params()** return constant **OSSL_PARAM**(3) arrays that describe the settable parameters that can be used with **EVP_KDF_CTX_set_params()**. **EVP_KDF_settable_ctx_params()** returns the parameters that can be retrieved from the algorithm, whereas **EVP_KDF_CTX_settable_params()** returns the parameters that can be retrieved in the context's current state.

Information functions

EVP_KDF_CTX_get_kdf_size() returns the output size if the algorithm produces a fixed amount of output and **SIZE_MAX** otherwise. If an error occurs then 0 is returned. For some algorithms an error may result if input parameters necessary to calculate a fixed output size have not yet been supplied.

EVP_KDF_is_a() returns 1 if *kdf* is an implementation of an algorithm that's identifiable with *name*, otherwise 0.

EVP_KDF_get0_provider() returns the provider that holds the implementation of the given *kdf*.

EVP_KDF_do_all_provided() traverses all KDF implemented by all activated providers in the given library context *libctx*, and for each of the implementations, calls the given function *fn* with the implementation method and the given *arg* as argument.

EVP_KDF_get0_name() return the name of the given KDF. For fetched KDFs with multiple names, only one of them is returned; it's recommended to use **EVP_KDF_names_do_all()** instead.

EVP_KDF_names_do_all() traverses all names for *kdf*, and calls *fn* with each name and *data*.

EVP_KDF_get0_description() returns a description of the *kdf*, meant for display and human consumption. The description is at the discretion of the *kdf* implementation.

PARAMETERS

The standard parameter names are:

"pass" (**OSSL_KDF_PARAM_PASSWORD**) <octet string>

Some KDF implementations require a password. For those KDF implementations that support it, this parameter sets the password.

"salt" (**OSSL_KDF_PARAM_SALT**) <octet string>

Some KDF implementations can take a non-secret unique cryptographic salt. For those KDF implementations that support it, this parameter sets the salt.

The default value, if any, is implementation dependent.

"iter" (**OSSL_KDF_PARAM_ITER**) <unsigned integer>

Some KDF implementations require an iteration count. For those KDF implementations that support it, this parameter sets the iteration count.

The default value, if any, is implementation dependent.

"properties" (**OSSL_KDF_PARAM_PROPERTIES**) <UTF8 string>
"mac" (**OSSL_KDF_PARAM_MAC**) <UTF8 string>
"digest" (**OSSL_KDF_PARAM_DIGEST**) <UTF8 string>
"cipher" (**OSSL_KDF_PARAM_CIPHER**) <UTF8 string>
For KDF implementations that use an underlying computation MAC, digest or cipher, these parameters set what the algorithm should be.

The value is always the name of the intended algorithm, or the properties.

Note that not all algorithms may support all possible underlying implementations.

"key" (**OSSL_KDF_PARAM_KEY**) <octet string>
Some KDF implementations require a key. For those KDF implementations that support it, this octet string parameter sets the key.

"info" (**OSSL_KDF_PARAM_INFO**) <octet string>
Some KDF implementations, such as **EVP_KDF-HKDF**(7), take an 'info' parameter for binding the derived key material to application- and context-specific information. This parameter sets the info, fixed info, other info or shared info argument. You can specify this parameter multiple times, and each instance will be concatenated to form the final value.

"maclen" (**OSSL_KDF_PARAM_MAC_SIZE**) <unsigned integer>
Used by implementations that use a MAC with a variable output size (KMAC). For those KDF implementations that support it, this parameter sets the MAC output size.

The default value, if any, is implementation dependent. The length must never exceed what can be given with a **size_t**.

"maxmem_bytes" (**OSSL_KDF_PARAM_SCRYPT_MAXMEM**) <unsigned integer>
Memory-hard password-based KDF algorithms, such as scrypt, use an amount of memory that depends on the load factors provided as input. For those KDF implementations that support it, this **uint64_t** parameter sets an upper limit on the amount of memory that may be consumed while performing a key derivation. If this memory usage limit is exceeded because the load factors are chosen too high, the key derivation will fail.

The default value is implementation dependent. The memory size must never exceed what can be given with a **size_t**.

RETURN VALUES

EVP_KDF_fetch() returns a pointer to a newly fetched **EVP_KDF**, or NULL if allocation failed.

EVP_KDF_get0_provider() returns a pointer to the provider for the KDF, or NULL on error.

EVP_KDF_up_ref() returns 1 on success, 0 on error.

EVP_KDF_CTX_new() returns either the newly allocated **EVP_KDF_CTX** structure or NULL if an error occurred.

EVP_KDF_CTX_free() and **EVP_KDF_CTX_reset()** do not return a value.

EVP_KDF_CTX_get_kdf_size() returns the output size. **SIZE_MAX** is returned to indicate that the algorithm produces a variable amount of output; 0 to indicate failure.

EVP_KDF_get0_name() returns the name of the KDF, or NULL on error.

EVP_KDF_names_do_all() returns 1 if the callback was called for all names. A return value of 0 means that the callback was not called for any names.

The remaining functions return 1 for success and 0 or a negative value for failure. In particular, a return value of -2 indicates the operation is not supported by the KDF algorithm.

NOTES

The KDF life-cycle is described in **life_cycle-kdf**(7). In the future, the transitions described there will be enforced. When this is done, it will not be considered a breaking change to the API.

SEE ALSO

"Key Derivation Function (KDF)" in **OSSL_PROVIDER-default**(7), **life_cycle-kdf**(7).

HISTORY

This functionality was added in OpenSSL 3.0.

COPYRIGHT