NAME

provider−rand − The random number generation 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_rand_newctx(void *provctx, void *parent,
                            const OSSL_DISPATCH *parent_calls);
void OSSL_FUNC_rand_freectx(void *ctx);

/* Random number generator functions: NIST */
int OSSL_FUNC_rand_instantiate(void *ctx, unsigned int strength,
                               int prediction_resistance,
                               const unsigned char *pstr, size_t pstr_len,
                               const OSSL_PARAM params[]);
int OSSL_FUNC_rand_uninstantiate(void *ctx);
int OSSL_FUNC_rand_generate(void *ctx, unsigned char *out, size_t outlen,
                            unsigned int strength, int prediction_resistance,
                            const unsigned char *addin, size_t addin_len);
int OSSL_FUNC_rand_reseed(void *ctx, int prediction_resistance,
                          const unsigned char *ent, size_t ent_len,
                          const unsigned char *addin, size_t addin_len);

/* Random number generator functions: additional */
size_t OSSL_FUNC_rand_nonce(void *ctx, unsigned char *out, size_t outlen,
                            int strength, size_t min_noncelen,
                            size_t max_noncelen);
size_t OSSL_FUNC_rand_get_seed(void *ctx, unsigned char **buffer,
                               int entropy, size_t min_len, size_t max_len,
                               int prediction_resistance,
                               const unsigned char *adin, size_t adin_len);
void OSSL_FUNC_rand_clear_seed(void *ctx, unsigned char *buffer, size_t b_len);
int OSSL_FUNC_rand_verify_zeroization(void *ctx);

/* Context Locking */
int OSSL_FUNC_rand_enable_locking(void *ctx);
int OSSL_FUNC_rand_lock(void *ctx);
void OSSL_FUNC_rand_unlock(void *ctx);

/* RAND parameter descriptors */
const OSSL_PARAM *OSSL_FUNC_rand_gettable_params(void *provctx);
const OSSL_PARAM *OSSL_FUNC_rand_gettable_ctx_params(void *ctx, void *provctx);
const OSSL_PARAM *OSSL_FUNC_rand_settable_ctx_params(void *ctx, void *provctx);

/* RAND parameters */
int OSSL_FUNC_rand_get_params(OSSL_PARAM params[]);
int OSSL_FUNC_rand_get_ctx_params(void *ctx, OSSL_PARAM params[]);
```

```
int OSSL_FUNC_rand_set_ctx_params(void *ctx, const OSSL_PARAM params[]);
```

DESCRIPTION

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

The RAND operation enables providers to implement random number generation algorithms and random number sources and make them available to applications via the API function **EVP_RAND** (3).

Context Management Functions

OSSL_FUNC_rand_newctx() should create and return a pointer to a provider side structure for holding context information during a rand operation. A pointer to this context will be passed back in a number of the other rand operation function calls. The parameter *provctx* is the provider context generated during provider initialisation (see **provider** (7)). The parameter*par ent* specifies another rand instance to be used for seeding purposes. If NULL and the specific instance supports it, the operating system will be used for seeding. The parameter*par ent_calls* points to the dispatch table for *parent*. Thus, the parent need not be from the same provider as the new instance.

OSSL_FUNC_rand_freectx() is passed a pointer to the provider side rand context in the *mctx* parameter. If it receives NULL as *ctx* value, it should not do anything other than return. This function should free any resources associated with that context.

Random Number Generator Functions: NIST

These functions correspond to those defined in NIST SP 800−90A and SP 800−90C.

OSSL_FUNC_rand_instantiate() is used to instantiate the DRBG *ctx* at a requested security *strength*. In addition, *prediction_resistance* can be requested. Additional input *addin* of length *addin_len* bytes can optionally be provided. The parameters specified in*par ams* configure the DRBG and these should be processed before instantiation.

OSSL_FUNC_rand_uninstantiate() is used to uninstantiate the DRBG *ctx*. After being uninstantiated, a DRBG is unable to produce output until it is instantiated anew.

OSSL_FUNC_rand_generate() is used to generate random bytes from the DRBG *ctx*. It will generate *outlen* bytes placing them into the buffer pointed to by *out*. The generated bytes will meet the specified security *strength* and, if *prediction_resistance* is true, the bytes will be produced after reseeding from a live entropy source. Additional input *addin* of length *addin_len* bytes can optionally be provided.

Random Number Generator Functions: Additional

OSSL_FUNC_rand_nonce() is used to generate a nonce of the given *strength* with a length from *min_noncelen* to *max_noncelen*. If the output buffer *out* is NULL, the length of the nonce should be returned.

OSSL_FUNC_rand_get_seed() is used by deterministic generators to obtain their seeding material from their parent. The seed bytes will meet the specified security level of *entropy* bits and there will be between *min_len* and *max_len* inclusive bytes in total. If *prediction_resistance* is true, the bytes will be produced from a live entropy source. Additional input *addin* of length *addin_len* bytes can optionally be provided. A pointer to the seed material is returned in **buffer* and this must be freed by a later call to **OSSL_FUNC_rand_clear_seed()**.

OSSL_FUNC_rand_clear_seed() frees a seed *buffer* of length *b_len* bytes which was previously allocated by **OSSL_FUNC_rand_get_seed()**.

OSSL_FUNC_rand_verify_zeroization() is used to determine if the internal state of the DRBG is zero. This capability is mandated by NIST as part of the self tests, it is unlikely to be useful in other circumstances.

Context Locking

When DRBGs are used by multiple threads, there must be locking employed to ensure their proper operation. Because locking introduces an overhead, it is disabled by default.

OSSL_FUNC_rand_enable_locking() allows locking to be turned on for a DRBG and all of its parent DRBGs. From this call onwards, the DRBG can be used in a thread safe manner.

OSSL_FUNC_rand_lock() is used to lock a DRBG. Once locked, exclusive access is guaranteed.

OSSL_FUNC_rand_unlock() is used to unlock a DRBG.

Rand Parameters

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

OSSL_FUNC_rand_get_params() gets details of parameter values associated with the provider algorithm and stores them in *params*.

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

OSSL_FUNC_rand_get_ctx_params() gets details of currently set parameter values associated with the given provider side rand context *ctx* and stores them in *params*. Passing NULL for *params* should return true.

OSSL_FUNC_rand_gettable_params(), **OSSL_FUNC_rand_gettable_ctx_params()**, and **OSSL_FUNC_rand_settable_ctx_params()** all return constant **OSSL_PARAM** arrays as descriptors of the parameters that **OSSL_FUNC_rand_get_params()**, **OSSL_FUNC_rand_get_ctx_params()**, and **OSSL_FUNC_rand_set_ctx_params()** can handle, respectively. **OSSL_FUNC_rand_gettable_ctx_params()** and **OSSL_FUNC_rand_settable_ctx_params()** will return the parameters associated with the provider side context *ctx* in its current state if it is not NULL. Otherwise, they return the parameters associated with the provider side algorithm *provctx*.

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

‘‘state’’ (**OSSL_RAND_PARAM_STATE**) <integer>
: Returns the state of the random number generator.

‘‘strength’’ (**OSSL_RAND_PARAM_STRENGTH**) <unsigned integer>
: Returns the bit strength of the random number generator.

For rands that are also deterministic random bit generators (DRBGs), these additional parameters are recognised. Not all parameters are relevant to, or are understood by all DRBG rands:

‘‘reseed_requests’’ (**OSSL_DRBG_PARAM_RESEED_REQUESTS**) <unsigned integer>
: Reads or set the number of generate requests before reseeding the associated RAND ctx.

‘‘reseed_time_interval’’ (**OSSL_DRBG_PARAM_RESEED_TIME_INTERVAL**) <integer>
: Reads or set the number of elapsed seconds before reseeding the associated RAND ctx.

‘‘max_request’’ (**OSSL_DRBG_PARAM_RESEED_REQUESTS**) <unsigned integer>
: Specifies the maximum number of bytes that can be generated in a single call to OSSL_FUNC_rand_generate.

‘‘min_entropylen’’ (**OSSL_DRBG_PARAM_MIN_ENTROPYLEN**) <unsigned integer>
‘‘max_entropylen’’ (**OSSL_DRBG_PARAM_MAX_ENTROPYLEN**) <unsigned integer>
: Specify the minimum and maximum number of bytes of random material that can be used to seed the DRBG.

‘‘min_noncelen’’ (**OSSL_DRBG_PARAM_MIN_NONCELEN**) <unsigned integer>
‘‘max_noncelen’’ (**OSSL_DRBG_PARAM_MAX_NONCELEN**) <unsigned integer>
: Specify the minimum and maximum number of bytes of nonce that can be used to instantiate the DRBG.

‘‘max_perslen’’ (**OSSL_DRBG_PARAM_MAX_PERSLEN**) <unsigned integer>
‘‘max_adinlen’’ (**OSSL_DRBG_PARAM_MAX_ADINLEN**) <unsigned integer>
: Specify the minimum and maximum number of bytes of personalisation string that can be used with the DRBG.

‘‘reseed_counter’’ (**OSSL_DRBG_PARAM_RESEED_COUNTER**) <unsigned integer>
Specifies the number of times the DRBG has been seeded or reseeded.

‘‘digest’’ (**OSSL_DRBG_PARAM_DIGEST**) <UTF8 string>
‘‘cipher’’ (**OSSL_DRBG_PARAM_CIPHER**) <UTF8 string>
‘‘mac’’ (**OSSL_DRBG_PARAM_MAC**) <UTF8 string>
Sets the name of the underlying cipher, digest or MAC to be used. It must name a suitable algorithm for the DRBG that’s being used.

‘‘properties’’ (**OSSL_DRBG_PARAM_PROPERTIES**) <UTF8 string>
Sets the properties to be queried when trying to fetch an underlying algorithm. This must be given together with the algorithm naming parameter to be considered valid.

RETURN VALUES

OSSL_FUNC_rand_newctx() should return the newly created provider side rand context, or NULL on failure.

OSSL_FUNC_rand_gettable_params(), **OSSL_FUNC_rand_gettable_ctx_params()** and **OSSL_FUNC_rand_settable_ctx_params()** should return a constant **OSSL_PARAM** array, or NULL if none is offered.

OSSL_FUNC_rand_nonce() returns the size of the generated nonce, or 0 on error.

OSSL_FUNC_rand_get_seed() returns the size of the generated seed, or 0 on error.

All of the remaining functions should return 1 for success or 0 on error.

NOTES

The RAND life-cycle is described in **life_cycle−rand** (7). Providers should ensure that the various transitions listed there are supported. At some point the EVP layer will begin enforcing the listed transitions.

SEE ALSO

provider (7), **RAND** (7), **EVP_RAND** (7), **life_cycle−rand** (7), **EVP_RAND** (3)

HISTORY

The provider RAND interface was introduced in OpenSSL 3.0.

COPYRIGHT