Rocky Enterprise Linux 9.2 Manual Pages on command 'RAND_priv_bytes.3ossl'

$ man RAND_priv_bytes.3ossl

RAND_BYTES(3ossl) OpenSSL RAND_BYTES(3ossl)

NAME

RAND_bytes, RAND_priv_bytes, RAND_bytes_ex, RAND_priv_bytes_ex,

RAND_pseudo_bytes - generate random data

SYNOPSIS

```
#include <openssl/rand.h>
int RAND_bytes(unsigned char *buf, int num);
int RAND_priv_bytes(unsigned char *buf, int num);
int RAND_bytes_ex(OSSL_LIB_CTX *ctx, unsigned char *buf, size_t num,
                  unsigned int strength);
int RAND_priv_bytes_ex(OSSL_LIB_CTX *ctx, unsigned char *buf, size_t num,
                       unsigned int strength);
```

The following function has been deprecated since OpenSSL 1.1.0, and can be hidden entirely by defining OPENSSL_API_COMPAT with a suitable version value, see openssl_user_macros(7):

```
int RAND_pseudo_bytes(unsigned char *buf, int num);
```

DESCRIPTION

RAND_bytes() generates num random bytes using a cryptographically

secure pseudo random generator (CSPRNG) and stores them in buf.

RAND_priv_bytes() has the same semantics as RAND_bytes(). It is intended to be used for generating values that should remain private. If using the default RAND_METHOD, this function uses a separate "private" PRNG instance so that a compromise of the "public" PRNG instance will not affect the secrecy of these private values, as described in RAND(7) and EVP_RAND(7).

RAND_bytes_ex() and RAND_priv_bytes_ex() are the same as RAND_bytes() and RAND_priv_bytes() except that they both take additional strength and ctx parameters. The bytes genreated will have a security strength of at least strength bits. The DRBG used for the operation is the public or private DRBG associated with the specified ctx. The parameter can be NULL, in which case the default library context is used (see OSSL_LIB_CTX(3). If the default RAND_METHOD has been changed then for compatibility reasons the RAND_METHOD will be used in preference and the DRBG of the library context ignored.

NOTES

By default, the OpenSSL CSPRNG supports a security level of 256 bits, provided it was able to seed itself from a trusted entropy source. On all major platforms supported by OpenSSL (including the Unix-like platforms and Windows), OpenSSL is configured to automatically seed the CSPRNG on first use using the operating systems's random generator. If the entropy source fails or is not available, the CSPRNG will enter an error state and refuse to generate random bytes. For that reason, it is important to always check the error return value of RAND_bytes() and RAND_priv_bytes() and not take randomness for granted.

On other platforms, there might not be a trusted entropy source available or OpenSSL might have been explicitly configured to use different entropy sources. If you are in doubt about the quality of the entropy source, don't hesitate to ask your operating system vendor or post a question on GitHub or the openssl-users mailing list.

RETURN VALUES

RAND_bytes() and RAND_priv_bytes() return 1 on success, -1 if not

supported by the current RAND method, or 0 on other failure. The error code can be obtained by ERR_get_error(3).

SEE ALSO

RAND_add(3), RAND_bytes(3), RAND_priv_bytes(3), ERR_get_error(3), RAND(7), EVP_RAND(7)

HISTORY

? RAND_pseudo_bytes() was deprecated in OpenSSL 1.1.0; use RAND_bytes() instead.

? The RAND_priv_bytes() function was added in OpenSSL 1.1.1.

? The RAND_bytes_ex() and RAND_priv_bytes_ex() functions were added in OpenSSL 3.0

COPYRIGHT