

**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 generated 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 CS RNG 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' 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

Copyright 2000-2023 The OpenSSL Project Authors. All Rights Reserved.

Licensed under the Apache License 2.0 (the "License"). You may not use this file except in compliance with the License. You can obtain a copy in the file LICENSE in the source distribution or at <<https://www.openssl.org/source/license.html>>.