



Red Hat Enterprise Linux Release 9.2 Manual Pages on 'BN_is_prime_fasttest.3oss1' command

\$ man BN_is_prime_fasttest.3oss1

BN_GENERATE_PRIME(3oss1) OpenSSL BN_GENERATE_PRIME(3oss1)

NAME

BN_generate_prime_ex2, BN_generate_prime_ex, BN_is_prime_ex,
BN_check_prime, BN_is_prime_fasttest_ex, BN_GENCB_call, BN_GENCB_new,
BN_GENCB_free, BN_GENCB_set_old, BN_GENCB_set, BN_GENCB_get_arg,
BN_generate_prime, BN_is_prime, BN_is_prime_fasttest - generate primes
and test for primality

SYNOPSIS

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

int BN_generate_prime_ex2(BIGNUM *ret, int bits, int safe,
                          const BIGNUM *add, const BIGNUM *rem, BN_GENCB *cb,
                          BN_CTX *ctx);

int BN_generate_prime_ex(BIGNUM *ret, int bits, int safe, const BIGNUM *add,
                          const BIGNUM *rem, BN_GENCB *cb);

int BN_check_prime(const BIGNUM *p, BN_CTX *ctx, BN_GENCB *cb);

int BN_GENCB_call(BN_GENCB *cb, int a, int b);

BN_GENCB *BN_GENCB_new(void);

void BN_GENCB_free(BN_GENCB *cb);

void BN_GENCB_set_old(BN_GENCB *gencb,
                      void (*callback)(int, int, void *), void *cb_arg);

void BN_GENCB_set(BN_GENCB *gencb,
                  int (*callback)(int, int, BN_GENCB *), void *cb_arg);

void *BN_GENCB_get_arg(BN_GENCB *cb);
```

The following functions have been deprecated since OpenSSL 0.9.8, and can be hidden entirely by defining OPENSSL_API_COMPAT with a suitable version value, see openssl_user_macros(7):

```
BIGNUM *BN_generate_prime(BIGNUM *ret, int num, int safe, BIGNUM *add,
                          BIGNUM *rem, void (*callback)(int, int, void *),
                          void *cb_arg);

int BN_is_prime(const BIGNUM *p, int nchecks,
               void (*callback)(int, int, void *), BN_CTX *ctx, void *cb_arg);

int BN_is_prime_fasttest(const BIGNUM *p, int nchecks,
                        void (*callback)(int, int, void *), BN_CTX *ctx,
                        void *cb_arg, int do_trial_division);
```

The following functions have been deprecated since OpenSSL 3.0, and can be hidden entirely by defining OPENSSL_API_COMPAT with a suitable version value, see openssl_user_macros(7):

```
int BN_is_prime_ex(const BIGNUM *p, int nchecks, BN_CTX *ctx, BN_GENCB *cb);
int BN_is_prime_fasttest_ex(const BIGNUM *p, int nchecks, BN_CTX *ctx,
                            int do_trial_division, BN_GENCB *cb);
```

DESCRIPTION

BN_generate_prime_ex2() generates a pseudo-random prime number of at least bit length bits using the BN_CTX provided in ctx. The value of ctx must not be NULL.

The returned number is probably prime with a negligible error. The maximum error rate is 2^{-128} . It's 2^{-287} for a 512 bit prime, 2^{-435} for a 1024 bit prime, 2^{-648} for a 2048 bit prime, and lower than 2^{-882} for primes larger than 2048 bit.

If add is NULL the returned prime number will have exact bit length bits with the top most two bits set.

If ret is not NULL, it will be used to store the number.

If cb is not NULL, it is used as follows:

? BN_GENCB_call(cb, 0, i) is called after generating the i-th potential prime number.

? While the number is being tested for primality, BN_GENCB_call(cb, 1, j) is called as described below.

? When a prime has been found, BN_GENCB_call(cb, 2, i) is called.

? The callers of BN_generate_prime_ex() may call BN_GENCB_call(cb, i, j) with other values as described in their respective man pages; see "SEE ALSO".

The prime may have to fulfill additional requirements for use in

Diffie-Hellman key exchange:

If add is not NULL, the prime will fulfill the condition $p \% add == rem$ ($p \% add == 1$ if $rem == NULL$) in order to suit a given generator.

If safe is true, it will be a safe prime (i.e. a prime p so that $(p-1)/2$ is also prime). If safe is true, and $rem == NULL$ the condition will be $p \% add == 3$. It is recommended that add is a multiple of 4.

The random generator must be seeded prior to calling

BN_generate_prime_ex(). If the automatic seeding or reseeding of the OpenSSL CSPRNG fails due to external circumstances (see RAND(7)), the operation will fail. The random number generator configured for the OSSL_LIB_CTX associated with ctx will be used.

BN_generate_prime_ex() is the same as BN_generate_prime_ex2() except that no ctx parameter is passed. In this case the random number generator associated with the default OSSL_LIB_CTX will be used.

BN_check_prime(), BN_is_prime_ex(), BN_is_prime_fasttest_ex(), BN_is_prime() and BN_is_prime_fasttest() test if the number p is prime.

The functions tests until one of the tests shows that p is composite, or all the tests passed. If p passes all these tests, it is considered a probable prime.

The test performed on p are trial division by a number of small primes and rounds of the of the Miller-Rabin probabilistic primality test.

The functions do at least 64 rounds of the Miller-Rabin test giving a maximum false positive rate of 2^{-128} . If the size of p is more than 2048 bits, they do at least 128 rounds giving a maximum false positive rate of 2^{-256} .

If nchecks is larger than the minimum above (64 or 128), nchecks rounds of the Miller-Rabin test will be done.

If do_trial_division set to 0, the trial division will be skipped.

`BN_is_prime_ex()` and `BN_is_prime()` always skip the trial division.

`BN_is_prime_ex()`, `BN_is_prime_fasttest_ex()`, `BN_is_prime()` and `BN_is_prime_fasttest()` are deprecated.

`BN_is_prime_fasttest()` and `BN_is_prime()` behave just like `BN_is_prime_fasttest_ex()` and `BN_is_prime_ex()` respectively, but with the old style call back.

`ctx` is a preallocated `BN_CTX` (to save the overhead of allocating and freeing the structure in a loop), or `NULL`.

If the trial division is done, and no divisors are found and `cb` is not `NULL`, `BN_GENCB_call(cb, 1, -1)` is called.

After each round of the Miller-Rabin probabilistic primality test, if `cb` is not `NULL`, `BN_GENCB_call(cb, 1, j)` is called with `j` the iteration (`j = 0, 1, ...`).

`BN_GENCB_call()` calls the callback function held in the `BN_GENCB` structure and passes the ints `a` and `b` as arguments. There are two types of `BN_GENCB` structure that are supported: "new" style and "old" style. New programs should prefer the "new" style, whilst the "old" style is provided for backwards compatibility purposes.

A `BN_GENCB` structure should be created through a call to `BN_GENCB_new()`, and freed through a call to `BN_GENCB_free()`.

For "new" style callbacks a `BN_GENCB` structure should be initialised with a call to `BN_GENCB_set()`, where `gencb` is a `BN_GENCB *`, `callback` is of type `int (*callback)(int, int, BN_GENCB *)` and `cb_arg` is a `void *`.

"Old" style callbacks are the same except they are initialised with a call to `BN_GENCB_set_old()` and `callback` is of type `void (*callback)(int, int, void *)`.

A callback is invoked through a call to `BN_GENCB_call`. This will check the type of the callback and will invoke `callback(a, b, gencb)` for new style callbacks or `callback(a, b, cb_arg)` for old style.

It is possible to obtain the argument associated with a `BN_GENCB` structure (set via a call to `BN_GENCB_set` or `BN_GENCB_set_old`) using `BN_GENCB_get_arg`.

`BN_generate_prime()` (deprecated) works in the same way as

BN_generate_prime_ex() but expects an old-style callback function directly in the callback parameter, and an argument to pass to it in the cb_arg. BN_is_prime() and BN_is_prime_fasttest() can similarly be compared to BN_is_prime_ex() and BN_is_prime_fasttest_ex(), respectively.

RETURN VALUES

BN_generate_prime_ex() return 1 on success or 0 on error.

BN_is_prime_ex(), BN_is_prime_fasttest_ex(), BN_is_prime(),

BN_is_prime_fasttest() and BN_check_prime return 0 if the number is composite, 1 if it is prime with an error probability of less than 0.25^n checks, and -1 on error.

BN_generate_prime() returns the prime number on success, NULL otherwise.

BN_GENCB_new returns a pointer to a BN_GENCB structure on success, or NULL otherwise.

BN_GENCB_get_arg returns the argument previously associated with a BN_GENCB structure.

Callback functions should return 1 on success or 0 on error.

The error codes can be obtained by ERR_get_error(3).

REMOVED FUNCTIONALITY

As of OpenSSL 1.1.0 it is no longer possible to create a BN_GENCB structure directly, as in:

```
BN_GENCB callback;
```

Instead applications should create a BN_GENCB structure using

```
BN_GENCB_new:
```

```
BN_GENCB *callback;
```

```
callback = BN_GENCB_new();
```

```
if (!callback)
```

```
    /* error */
```

```
...
```

```
BN_GENCB_free(callback);
```

SEE ALSO

DH_generate_parameters(3), DSA_generate_parameters(3),

RSA_generate_key(3), ERR_get_error(3), RAND_bytes(3), RAND(7)

HISTORY

The BN_is_prime_ex() and BN_is_prime_fasttest_ex() functions were deprecated in OpenSSL 3.0.

The BN_GENCB_new(), BN_GENCB_free(), and BN_GENCB_get_arg() functions were added in OpenSSL 1.1.0.

BN_check_prime() was added in OpenSSL 3.0.

COPYRIGHT

Copyright 2000-2021 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>>.

3.0.7 2023-07-13 BN_GENERATE_PRIME(3openssl)