

**NAME**

`BN_generate_prime_ex`, `BN_is_prime_ex`, `BN_is_prime_fasttest_ex`, `BN_GENCB_call`, `BN_GENCB_set_old`, `BN_GENCB_set`, `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_ex(BIGNUM *ret,int bits,int safe, const BIGNUM *add,
const BIGNUM *rem, BN_GENCB *cb);
```

```
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);
```

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

```
#define BN_GENCB_set_old(genccb, callback, cb_arg) ...
```

```
#define BN_GENCB_set(genccb, callback, cb_arg) ...
```

Deprecated:

```
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 *a, int checks, void (*callback)(int, int,
void *), BN_CTX *ctx, void *cb_arg);
```

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

**DESCRIPTION**

`BN_generate_prime_ex()` generates a pseudo-random prime number of bit length `bits`. 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 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).

The PRNG must be seeded prior to calling `BN_generate_prime_ex()`. The prime number generation has a negligible error probability.

`BN_is_prime_ex()` and `BN_is_prime_fasttest_ex()` test if the number `p` is prime. The following tests are performed until one of them shows that `p` is composite; if `p` passes all these tests, it is considered prime.

`BN_is_prime_fasttest_ex()`, when called with `do_trial_division == 1`, first attempts trial division by a number of small primes; if no divisors are found by this test and `cb` is not `NULL`,

**BN\_GENCB\_call**(**cb**, **1**, **-1**) is called. If **do\_trial\_division == 0**, this test is skipped.

Both *BN\_is\_prime\_ex()* and *BN\_is\_prime\_fasttest\_ex()* perform a Miller-Rabin probabilistic primality test with **nchecks** iterations. If **nchecks == BN\_prime\_checks**, a number of iterations is used that yields a false positive rate of at most  $2^{-80}$  for random input.

If **cb** is not **NULL**, **BN\_GENCB\_call**(**cb**, **1**, **j**) is called after the **j**-th iteration (**j** = 0, 1, ...). **ctx** is a pre-allocated **BN\_CTX** (to save the overhead of allocating and freeing the structure in a loop), or **NULL**.

**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.

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.

**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**. Similarly **BN\_is\_prime** and **BN\_is\_prime\_fasttest** are deprecated and can 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()* and *BN\_is\_prime\_fasttest()* return 0 if the number is composite, 1 if it is prime with an error probability of less than  $0.25^{nchecks}$ , and -1 on error.

*BN\_generate\_prime()* returns the prime number on success, **NULL** otherwise.

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

The error codes can be obtained by *ERR\_get\_error(3)*.

## SEE ALSO

*bn(3)*, *ERR\_get\_error(3)*, *rand(3)*

## HISTORY

The **cb\_arg** arguments to *BN\_generate\_prime()* and to *BN\_is\_prime()* were added in SSLeay 0.9.0. The **ret** argument to *BN\_generate\_prime()* was added in SSLeay 0.9.1. *BN\_is\_prime\_fasttest()* was added in OpenSSL 0.9.5.