

Probable prime – OpenSSL

```
int BN_generate_prime_ex(BIGNUM *ret, int bits, int safe,
    const BIGNUM *add, const BIGNUM *rem, BN_GENCB *cb)
{
    BIGNUM *t;
    int found=0;
    int i,j,c1=0;
    BN_CTX *ctx;
    int checks = BN_prime_checks_for_size(bits);

    ctx=BN_CTX_new();
    if (ctx == NULL) goto err;
    BN_CTX_start(ctx);
    t = BN_CTX_get(ctx);
    if(!t) goto err;

loop:
    /* make a random number and set the top and bottom bits */
    if (add == NULL)
    {
        if (!probable_prime(ret,bits)) goto err;
    }
    else
    {
        if (safe)
        {
            if (!probable_prime_dh_safe(ret,bits,add,rem,ctx))
                goto err;
        }
        else
        {
            if (!probable_prime_dh(ret,bits,add,rem,ctx))
                goto err;
        }
    }

    /* if (BN_mod_word(ret, (BN_ULONG)3) == 1) goto loop; */
    if(!BN_GENCB_call(cb, 0, c1++))
        /* aborted */
        goto err;

    if (!safe)
    {
        i=BN_is_prime_fasttest_ex(ret,checks,ctx,0,cb);
        if (i == -1) goto err;
        if (i == 0) goto loop;
    }
    else
    {
        /* for "safe prime" generation,
         * check that (p-1)/2 is prime.
         * Since a prime is odd, We just
         * need to divide by 2 */
        if (!BN_rshift1(t,ret)) goto err;

        for (i=0; i<checks; i++)
        {
            j=BN_is_prime_fasttest_ex(ret,1,ctx,0,cb);
            if (j == -1) goto err;
            if (j == 0) goto loop;

            j=BN_is_prime_fasttest_ex(t,1,ctx,0,cb);
            if (j == -1) goto err;
            if (j == 0) goto loop;

            if(!BN_GENCB_call(cb, 2, c1-1))
                goto err;
            /* We have a safe prime test pass */
        }

        /* we have a prime :-) */
        found = 1;
err:
        if (ctx != NULL)
        {
            BN_CTX_end(ctx);
            BN_CTX_free(ctx);
        }
        bn_check_top(ret);
        return found;
    }

static int probable_prime(BIGNUM *rnd, int bits)
{
    int i;
    BN_ULONG mods[NUMPRIMES];
    BN_ULONG delta,maxdelta;

again:
    if (!BN_rand(rnd,bits,1,1)) return(0);
    /* we now have a random number 'rand' to test. */
    for (i=1; i<NUMPRIMES; i++)
        mods[i]=BN_mod_word(rnd, (BN_ULONG)primes[i]);
    maxdelta=BN_MASK2 - primes[NUMPRIMES-1];
    delta=0;
loop: for (i=1; i<NUMPRIMES; i++)
    {
        /* check that rnd is not a prime and also
         * that gcd(rnd-1,primes) == 1 (except for 2) */
        if ((mods[i]+delta)%primes[i] <= 1)
        {
            delta+=2;
            if (delta > maxdelta) goto again;
            goto loop;
        }
    }
    if (!BN_add_word(rnd,delta)) return(0);
    bn_check_top(rnd);
    return(1);
}

int BN_is_prime_fasttest_ex(const BIGNUM *a, int checks, BN_CTX *ctx_passed,
    int do_trial_division, BN_GENCB *cb)
{
    int i, j, ret = -1;
    int k;
    BN_CTX *ctx = NULL;
    BIGNUM *A1, *A1_odd, *check; /* taken from ctx */
    BN_MONT_CTX *mont = NULL;
    const BIGNUM *A = NULL;

    if (BN_cmp(a, BN_value_one()) <= 0)
        return 0;

    if (checks == BN_prime_checks)
        checks = BN_prime_checks_for_size(BN_num_bits(a));

    /* first look for small factors */
    if (!BN_is_odd(a))
        /* a is even => a is prime if and only if a == 2 */
        return BN_is_word(a, 2);
    if (do_trial_division)
    {
        for (i = 1; i < NUMPRIMES; i++)
            if (BN_mod_word(a, primes[i]) == 0)
                return 0;
        if(!BN_GENCB_call(cb, 1, -1))
            goto err;
    }

    if (ctx_passed != NULL)
        ctx = ctx_passed;
    else
        if ((ctx=BN_CTX_new()) == NULL)
            goto err;
    BN_CTX_start(ctx);

    /* A := abs(a) */
    if (a->neg)
    {
        BIGNUM *t;
        if ((t = BN_CTX_get(ctx)) == NULL) goto err;
        BN_copy(t, a);
        t->neg = 0;
        A = t;
    }
    else
        A = a;
    A1 = BN_CTX_get(ctx);
    A1_odd = BN_CTX_get(ctx);
    check = BN_CTX_get(ctx);
    if (check == NULL) goto err;

    /* compute A1 := A - 1 */
    if (!BN_copy(A1, A))
        goto err;
    if (!BN_sub_word(A1, 1))
        goto err;
    if (BN_is_zero(A1))
    {
        ret = 0;
        goto err;
    }

    /* write A1 as A1_odd * 2^k */
    k = 1;
    while (!BN_is_bit_set(A1, k))
        k++;
    if (!BN_rshift(A1_odd, A1, k))
        goto err;

    /* Montgomery setup for computations mod A */
    mont = BN_MONT_CTX_new();
    if (mont == NULL)
        goto err;
    if (!BN_MONT_CTX_set(mont, A, ctx))
        goto err;

    for (i = 0; i < checks; i++)
    {
        if (!BN_pseudo_rand_range(check, A1))
            goto err;
        if (!BN_add_word(check, 1))
            goto err;
        /* now 1 <= check < A */

        j = witness(check, A, A1, A1_odd, k, ctx, mont);
        if (j == -1) goto err;
        if (j)
        {
            ret=0;
            goto err;
        }
        if(!BN_GENCB_call(cb, 1, i))
            goto err;
    }
    ret=1;

err:
    if (ctx != NULL)
    {
        BN_CTX_end(ctx);
        if (ctx_passed == NULL)
            BN_CTX_free(ctx);
    }
    if (mont != NULL)
        BN_MONT_CTX_free(mont);

    return(ret);
}
```