

**NAME**

EC\_POINT\_add, EC\_POINT\_dbl, EC\_POINT\_invert, EC\_POINT\_is\_at\_infinity,  
 EC\_POINT\_is\_on\_curve, EC\_POINT\_cmp, EC\_POINT\_make\_affine, EC\_POINTS\_make\_affine,  
 EC\_POINTS\_mul, EC\_POINT\_mul, EC\_GROUP\_precompute\_mult,  
 EC\_GROUP\_have\_precompute\_mult - Functions for performing mathematical operations and tests on EC\_POINT objects.

**SYNOPSIS**

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

```
int EC_POINT_add(const EC_GROUP *group, EC_POINT *r, const EC_POINT *a, const EC_POINT *b, BN_CTX *ctx);
int EC_POINT_dbl(const EC_GROUP *group, EC_POINT *r, const EC_POINT *a, BN_CTX *ctx);
int EC_POINT_invert(const EC_GROUP *group, EC_POINT *a, BN_CTX *ctx);
int EC_POINT_is_at_infinity(const EC_GROUP *group, const EC_POINT *p);
int EC_POINT_is_on_curve(const EC_GROUP *group, const EC_POINT *point, BN_CTX *ctx);
int EC_POINT_cmp(const EC_GROUP *group, const EC_POINT *a, const EC_POINT *b, BN_CTX *ctx);
int EC_POINT_make_affine(const EC_GROUP *group, EC_POINT *point, BN_CTX *ctx);
int EC_POINTS_make_affine(const EC_GROUP *group, size_t num, EC_POINT *points[], BN_CTX *ctx);
int EC_POINTS_mul(const EC_GROUP *group, EC_POINT *r, const BIGNUM *n, size_t num, const EC_POINT *points, BN_CTX *ctx);
int EC_POINT_mul(const EC_GROUP *group, EC_POINT *r, const BIGNUM *n, const EC_POINT *q, const EC_GROUP *group, BN_CTX *ctx);
int EC_GROUP_precompute_mult(EC_GROUP *group, BN_CTX *ctx);
int EC_GROUP_have_precompute_mult(const EC_GROUP *group);
```

**DESCRIPTION**

EC\_POINT\_add adds the two points **a** and **b** and places the result in **r**. Similarly EC\_POINT\_dbl doubles the point **a** and places the result in **r**. In both cases it is valid for **r** to be one of **a** or **b**.

EC\_POINT\_invert calculates the inverse of the supplied point **a**. The result is placed back in **a**.

The function EC\_POINT\_is\_at\_infinity tests whether the supplied point is at infinity or not.

EC\_POINT\_is\_on\_curve tests whether the supplied point is on the curve or not.

EC\_POINT\_cmp compares the two supplied points and tests whether or not they are equal.

The functions EC\_POINT\_make\_affine and EC\_POINTS\_make\_affine force the internal representation of the EC\_POINT(s) into the affine co-ordinate system. In the case of EC\_POINTS\_make\_affine the value **num** provides the number of points in the array **points** to be forced.

EC\_POINT\_mul calculates the value generator \* **n** + **q** \* **m** and stores the result in **r**. The value **n** may be NULL in which case the result is just **q** \* **m**.

EC\_POINTS\_mul calculates the value generator \* **n** + **q**[0] \* **m**[0] + ... + **q**[**num**-1] \* **m**[**num**-1]. As for EC\_POINT\_mul the value **n** may be NULL.

The function EC\_GROUP\_precompute\_mult stores multiples of the generator for faster point multiplication, whilst EC\_GROUP\_have\_precompute\_mult tests whether precomputation has already been done. See [EC\\_GROUP\\_copy\(3\)](#) for information about the generator.

**RETURN VALUES**

The following functions return 1 on success or 0 on error: EC\_POINT\_add, EC\_POINT\_dbl, EC\_POINT\_invert, EC\_POINT\_make\_affine, EC\_POINTS\_make\_affine, EC\_POINTS\_make\_affine, EC\_POINT\_mul, EC\_POINTS\_mul and EC\_GROUP\_precompute\_mult.

EC\_POINT\_is\_at\_infinity returns 1 if the point is at infinity, or 0 otherwise.

EC\_POINT\_is\_on\_curve returns 1 if the point is on the curve, 0 if not, or -1 on error.

EC\_POINT\_cmp returns 1 if the points are not equal, 0 if they are, or -1 on error.

EC\_GROUP\_have\_precompute\_mult return 1 if a precomputation has been done, or 0 if not.

**SEE ALSO**

[crypto\(3\)](#), [ec\(3\)](#), [EC\\_GROUP\\_new\(3\)](#), [EC\\_GROUP\\_copy\(3\)](#), [EC\\_POINT\\_new\(3\)](#),  
[EC\\_KEY\\_new\(3\)](#), [EC\\_GFp\\_simple\\_method\(3\)](#), [d2i\\_ECParameters\(3\)](#)