NAME

ec - Elliptic Curve functions

SYNOPSIS

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

const EC_METHOD *EC_GFp_simple_method(void);
const EC_METHOD *EC_GFp_mont_method(void);
const EC_METHOD *EC_GFp_nist_method(void);
const EC_METHOD *EC_GFp_nistp224_method(void);
const EC_METHOD *EC_GFp_nistp256_method(void);
const EC_METHOD *EC_GFp_nistp521_method(void);

const EC_METHOD *EC_GF2m_simple_method(void);

EC_GROUP *EC_GROUP_new(const EC_METHOD *meth);
void EC_GROUP_free(EC_GROUP *group);
void EC_GROUP_clear_free(EC_GROUP *group);
int EC_GROUP_copy(EC_GROUP *dst, const EC_GROUP *src);
EC_GROUP *EC_GROUP_dup(const EC_GROUP *src);
const EC_METHOD *EC_GROUP_method_of(const EC_GROUP *group);
int EC_METHOD_get_field_type(const EC_METHOD *meth);
int EC_GROUP_set_generator(EC_GROUP *group, const EC_POINT *generator, const BIGNUM *order,
const EC_POINT *EC_GROUP_get0_generator(const EC_GROUP *group);
int EC_GROUP_get_order(const EC_GROUP *group, BIGNUM *order, BN_CTX *ctx);
int EC_GROUP_get_cofactor(const EC_GROUP *group, BIGNUM *cofactor, BN_CTX *ctx);
void EC_GROUP_set_curve_name(EC_GROUP *group, int nid);
int EC_GROUP_get_curve_name(const EC_GROUP *group);
void EC_GROUP_set_asn1_flag(EC_GROUP *group, int flag);
int EC_GROUP_get_asn1_flag(const EC_GROUP *group);
void EC_GROUP_set_point_conversion_form(EC_GROUP *group, point_conversion_form_t form);
point_conversion_form_t EC_GROUP_get_point_conversion_form(const EC_GROUP *);
unsigned char *EC_GROUP_get0_seed(const EC_GROUP *x);
size_t EC_GROUP_get_seed_len(const EC_GROUP *);
size_t EC_GROUP_set_seed(EC_GROUP *, const unsigned char *, size_t len);
int EC_GROUP_set_curve_GFp(EC_GROUP *group, const BIGNUM *p, const BIGNUM *a, const BIGNUM *
int EC_GROUP_get_curve_GFp(const EC_GROUP *group, BIGNUM *p, BIGNUM *a, BIGNUM *b, BN_CTX *c
int EC_GROUP_set_curve_GF2m(EC_GROUP *group, const BIGNUM *p, const BIGNUM *a, const BIGNUM
int EC_GROUP_get_curve_GF2m(const EC_GROUP *group, BIGNUM *p, BIGNUM *a, BIGNUM *b, BN_CTX *
int EC_GROUP_get_degree(const EC_GROUP *group);
int EC_GROUP_check(const EC_GROUP *group, BN_CTX *ctx);
int EC_GROUP_check_discriminant(const EC_GROUP *group, BN_CTX *ctx);
int EC_GROUP_cmp(const EC_GROUP *a, const EC_GROUP *b, BN_CTX *ctx);
EC_GROUP *EC_GROUP_new_curve_GFp(const BIGNUM *p, const BIGNUM *a, const BIGNUM *b, BN_CTX *
EC_GROUP *EC_GROUP_new_curve_GF2m(const BIGNUM *p, const BIGNUM *a, const BIGNUM *b, BN_CTX
EC_GROUP *EC_GROUP_new_by_curve_name(int nid);

size_t EC_get_builtin_curves(EC_builtin_curve *r, size_t nitems);

EC_POINT *EC_POINT_new(const EC_GROUP *group);
void EC_POINT_free(EC_POINT *point);
void EC_POINT_clear_free(EC_POINT *point);
int EC_POINT_copy(EC_POINT *dst, const EC_POINT *src);
EC_POINT *EC_POINT_dup(const EC_POINT *src, const EC_GROUP *group);
```

```
const EC_METHOD *EC_POINT_method_of(const EC_POINT *point);
int EC_POINT_set_to_infinity(const EC_GROUP *group, EC_POINT *point);
int EC_POINT_set_Jprojective_coordinates_GFp(const EC_GROUP *group, EC_POINT *p,
const BIGNUM *x, const BIGNUM *y, const BIGNUM *z, BN_CTX *ctx);
int EC_POINT_get_Jprojective_coordinates_GFp(const EC_GROUP *group,
const EC_POINT *p, BIGNUM *x, BIGNUM *y, BIGNUM *z, BN_CTX *ctx);
int EC_POINT_set_affine_coordinates_GFp(const EC_GROUP *group, EC_POINT *p,
const BIGNUM *x, const BIGNUM *y, BN_CTX *ctx);
int EC_POINT_get_affine_coordinates_GFp(const EC_GROUP *group,
const EC_POINT *p, BIGNUM *x, BIGNUM *y, BN_CTX *ctx);
int EC_POINT_set_compressed_coordinates_GFp(const EC_GROUP *group, EC_POINT *p,
const BIGNUM *x, int y_bit, BN_CTX *ctx);
int EC_POINT_set_affine_coordinates_GF2m(const EC_GROUP *group, EC_POINT *p,
const BIGNUM *x, const BIGNUM *y, BN_CTX *ctx);
int EC_POINT_get_affine_coordinates_GF2m(const EC_GROUP *group,
const EC_POINT *p, BIGNUM *x, BIGNUM *y, BN_CTX *ctx);
int EC_POINT_set_compressed_coordinates_GF2m(const EC_GROUP *group, EC_POINT *p,
const BIGNUM *x, int y_bit, BN_CTX *ctx);
size_t EC_POINT_point2oct(const EC_GROUP *group, const EC_POINT *p,
point_conversion_form_t form,
unsigned char *buf, size_t len, BN_CTX *ctx);
int EC_POINT_oct2point(const EC_GROUP *group, EC_POINT *p,
const unsigned char *buf, size_t len, BN_CTX *ctx);
BIGNUM *EC_POINT_point2bn(const EC_GROUP *, const EC_POINT *,
point_conversion_form_t form, BIGNUM *, BN_CTX *);
EC_POINT *EC_POINT_bn2point(const EC_GROUP *, const BIGNUM *,
EC_POINT *, BN_CTX *);
char *EC_POINT_point2hex(const EC_GROUP *, const EC_POINT *,
point_conversion_form_t form, BN_CTX *);
EC_POINT *EC_POINT_hex2point(const EC_GROUP *, const char *,
EC_POINT *, BN_CTX *);

int EC_POINT_add(const EC_GROUP *group, EC_POINT *r, const EC_POINT *a, const EC_POINT *b, B
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_
int EC_POINT_mul(const EC_GROUP *group, EC_POINT *r, const BIGNUM *n, const EC_POINT *q, con
int EC_GROUP_precompute_mult(EC_GROUP *group, BN_CTX *ctx);
int EC_GROUP_have_precompute_mult(const EC_GROUP *group);

int EC_GROUP_get_basis_type(const EC_GROUP *);
int EC_GROUP_get_trinomial_basis(const EC_GROUP *, unsigned int *k);
int EC_GROUP_get_pentanomial_basis(const EC_GROUP *, unsigned int *k1,
unsigned int *k2, unsigned int *k3);
EC_GROUP *d2i_ECPKParameters(EC_GROUP **, const unsigned char **in, long len);
int i2d_ECPKParameters(const EC_GROUP *, unsigned char **out);
#define d2i_ECPKParameters_bio(bp,x) ASN1_d2i_bio_of(EC_GROUP,NULL,d2i_ECPKParameters,bp,x)
#define i2d_ECPKParameters_bio(bp,x) ASN1_i2d_bio_of_const(EC_GROUP,i2d_ECPKParameters,bp,x)
#define d2i_ECPKParameters_fp(fp,x) (EC_GROUP *)ASN1_d2i_fp(NULL, \
```

```
(char *(*)())d2i_ECPKParameters,(fp),(unsigned char **)(x))
#define i2d_ECPKParameters_fp(fp,x) ASN1_i2d_fp(i2d_ECPKParameters,(fp), \
(unsigned char *)(x))
int ECPKParameters_print(BIO *bp, const EC_GROUP *x, int off);
int ECPKParameters_print_fp(FILE *fp, const EC_GROUP *x, int off);

EC_KEY *EC_KEY_new(void);
int EC_KEY_get_flags(const EC_KEY *key);
void EC_KEY_set_flags(EC_KEY *key, int flags);
void EC_KEY_clear_flags(EC_KEY *key, int flags);
EC_KEY *EC_KEY_new_by_curve_name(int nid);
void EC_KEY_free(EC_KEY *key);
EC_KEY *EC_KEY_copy(EC_KEY *dst, const EC_KEY *src);
EC_KEY *EC_KEY_dup(const EC_KEY *src);
int EC_KEY_up_ref(EC_KEY *key);
const EC_GROUP *EC_KEY_get0_group(const EC_KEY *key);
int EC_KEY_set_group(EC_KEY *key, const EC_GROUP *group);
const BIGNUM *EC_KEY_get0_private_key(const EC_KEY *key);
int EC_KEY_set_private_key(EC_KEY *key, const BIGNUM *prv);
const EC_POINT *EC_KEY_get0_public_key(const EC_KEY *key);
int EC_KEY_set_public_key(EC_KEY *key, const EC_POINT *pub);
unsigned EC_KEY_get_enc_flags(const EC_KEY *key);
void EC_KEY_set_enc_flags(EC_KEY *eckey, unsigned int flags);
point_conversion_form_t EC_KEY_get_conv_form(const EC_KEY *key);
void EC_KEY_set_conv_form(EC_KEY *eckey, point_conversion_form_t cform);
void *EC_KEY_get_key_method_data(EC_KEY *key,
void *(*dup_func)(void *), void (*free_func)(void *), void (*clear_free_func)(void *));
void EC_KEY_insert_key_method_data(EC_KEY *key, void *data,
void *(*dup_func)(void *), void (*free_func)(void *), void (*clear_free_func)(void *));
void EC_KEY_set_asn1_flag(EC_KEY *eckey, int asn1_flag);
int EC_KEY_precompute_mult(EC_KEY *key, BN_CTX *ctx);
int EC_KEY_generate_key(EC_KEY *key);
int EC_KEY_check_key(const EC_KEY *key);
int EC_KEY_set_public_key_affine_coordinates(EC_KEY *key, BIGNUM *x, BIGNUM *y);

EC_KEY *d2i_ECPrivateKey(EC_KEY **key, const unsigned char **in, long len);
int i2d_ECPrivateKey(EC_KEY *key, unsigned char **out);

EC_KEY *d2i_ECParameters(EC_KEY **key, const unsigned char **in, long len);
int i2d_ECParameters(EC_KEY *key, unsigned char **out);

EC_KEY *o2i_ECPublicKey(EC_KEY **key, const unsigned char **in, long len);
int i2o_ECPublicKey(EC_KEY *key, unsigned char **out);
int ECParameters_print(BIO *bp, const EC_KEY *key);
int EC_KEY_print(BIO *bp, const EC_KEY *key, int off);
int ECParameters_print_fp(FILE *fp, const EC_KEY *key);
int EC_KEY_print_fp(FILE *fp, const EC_KEY *key, int off);
#define ECParameters_dup(x) ASN1_dup_of(EC_KEY,i2d_ECParameters,d2i_ECParameters,x)
#define EVP_PKEY_CTX_set_ec_paramgen_curve_nid(ctx, nid) \
EVP_PKEY_CTX_ctrl(ctx, EVP_PKEY_EC, EVP_PKEY_OP_PARAMGEN, \
EVP_PKEY_CTRL_EC_PARAMGEN_CURVE_NID, nid, NULL)
```

DESCRIPTION

This library provides an extensive set of functions for performing operations on elliptic curves over finite fields. In general an elliptic curve is one with an equation of the form:

y^2 = x^3 + ax + b

An **EC_GROUP** structure is used to represent the definition of an elliptic curve. Points on a curve are stored using an **EC_POINT** structure. An **EC_KEY** is used to hold a private/public key pair, where a private key is simply a BIGNUM and a public key is a point on a curve (represented by an **EC_POINT**).

The library contains a number of alternative implementations of the different functions. Each implementation is optimised for different scenarios. No matter which implementation is being used, the interface remains the same. The library handles calling the correct implementation when an interface function is invoked. An implementation is represented by an **EC_METHOD** structure.

The creation and destruction of **EC_GROUP** objects is described in *EC_GROUP_new(3)*. Functions for manipulating **EC_GROUP** objects are described in *EC_GROUP_copy(3)*.

Functions for creating, destroying and manipulating **EC_POINT** objects are explained in *EC_POINT_new(3)*, whilst functions for performing mathematical operations and tests on **EC_POINTs** are coverd in *EC_POINT_add(3)*.

For working with private and public keys refer to *EC_KEY_new(3)*. Implementations are covered in *EC_GFp_simple_method(3)*.

For information on encoding and decoding curve parameters to and from ASN1 see *d2i_ECPKParameters(3)*.

SEE ALSO

crypto(3), *EC_GROUP_new(3)*, *EC_GROUP_copy(3)*, *EC_POINT_new(3)*, *EC_POINT_add(3)*, *EC_KEY_new(3)*, *EC_GFp_simple_method(3)*, *d2i_ECPKParameters(3)*