## NAME

ECDSA_SIG_new, ECDSA_SIG_free, i2d_ECDSA_SIG, d2i_ECDSA_SIG, ECDSA_size, ECDSA_sign_setup, ECDSA_sign, ECDSA_sign_ex, ECDSA_verify, ECDSA_do_sign, ECDSA_do_sign_ex, ECDSA_do_verify - Elliptic Curve Digital Signature Algorithm

## SYNOPSIS

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

ECDSA_SIG* ECDSA_SIG_new(void);
void ECDSA_SIG_free(ECDSA_SIG *sig);
int i2d_ECDSA_SIG(const ECDSA_SIG *sig, unsigned char **pp);
ECDSA_SIG* d2i_ECDSA_SIG(ECDSA_SIG **sig, const unsigned char **pp,
long len);

ECDSA_SIG* ECDSA_do_sign(const unsigned char *dgst, int dgst_len,
EC_KEY *eckey);
ECDSA_SIG* ECDSA_do_sign_ex(const unsigned char *dgst, int dgstlen,
const BIGNUM *kinv, const BIGNUM *rp,
EC_KEY *eckey);
int ECDSA_do_verify(const unsigned char *dgst, int dgst_len,
const ECDSA_SIG *sig, EC_KEY* eckey);
int ECDSA_sign_setup(EC_KEY *eckey, BN_CTX *ctx,
BIGNUM **kinv, BIGNUM **rp);
int ECDSA_sign(int type, const unsigned char *dgst,
int dgstlen, unsigned char *sig,
unsigned int *siglen, EC_KEY *eckey);
int ECDSA_sign_ex(int type, const unsigned char *dgst,
int dgstlen, unsigned char *sig,
unsigned int *siglen, const BIGNUM *kinv,
const BIGNUM *rp, EC_KEY *eckey);
int ECDSA_verify(int type, const unsigned char *dgst,
int dgstlen, const unsigned char *sig,
int siglen, EC_KEY *eckey);
int ECDSA_size(const EC_KEY *eckey);

const ECDSA_METHOD* ECDSA_OpenSSL(void);
void ECDSA_set_default_method(const ECDSA_METHOD *meth);
const ECDSA_METHOD* ECDSA_get_default_method(void);
int ECDSA_set_method(EC_KEY *eckey,const ECDSA_METHOD *meth);

int ECDSA_get_ex_new_index(long argl, void *argp,
CRYPTO_EX_new *new_func,
CRYPTO_EX_dup *dup_func,
CRYPTO_EX_free *free_func);
int ECDSA_set_ex_data(EC_KEY *d, int idx, void *arg);
void* ECDSA_get_ex_data(EC_KEY *d, int idx);
```

## DESCRIPTION

The **ECDSA_SIG** structure consists of two BIGNUMs for the r and s value of a ECDSA signature (see X9.62 or FIPS 186-2).

```
struct
{
BIGNUM *r;
BIGNUM *s;
} ECDSA_SIG;
```

*ECDSA_SIG_new()* allocates a new **ECDSA_SIG** structure (note: this function also allocates the BIGNUMs) and initialize it.

*ECDSA_SIG_free()* frees the **ECDSA_SIG** structure **sig**.

*i2d_ECDSA_SIG()* creates the DER encoding of the ECDSA signature **sig** and writes the encoded signature to ***pp** (note: if **pp** is NULL **i2d_ECDSA_SIG** returns the expected length in bytes of the DER encoded signature). **i2d_ECDSA_SIG** returns the length of the DER encoded signature (or 0 on error).

*d2i_ECDSA_SIG()* decodes a DER encoded ECDSA signature and returns the decoded signature in a newly allocated **ECDSA_SIG** structure. ***sig** poin ts to the buffer containing the DER encoded signature of size **len**.

*ECDSA_size()* returns the maximum length of a DER encoded ECDSA signature created with the private EC key **eckey**.

*ECDSA_sign_setup()* may be used to precompute parts of the signing operation. **eckey** is the private EC key and **ctx** is a pointer to **BN_CTX** structure (or NULL). The precomputed values or returned in **kinv** and **rp** and can be used in a later call to **ECDSA_sign_ex** or **ECDSA_do_sign_ex**.

*ECDSA_sign()* is wrapper function for ECDSA_sign_ex with **kinv** and **rp** set to NULL.

*ECDSA_sign_ex()* computes a digital signature of the **dgstlen** bytes hash value **dgst** using the private EC key **eckey** and the optional pre-computed values **kinv** and **rp**. The DER encoded signatures is stored in **sig** and it's length is returned in **sig_len**. Note: **sig** must point to **ECDSA_size** bytes of memory. The parameter **type** is ignored.

*ECDSA_verify()* verifies that the signature in **sig** of size **siglen** is a valid ECDSA signature of the hash value **dgst** of size **dgstlen** using the public key **eckey**. The parameter**t ype** is ignored.

*ECDSA_do_sign()* is wrapper function for ECDSA_do_sign_ex with **kinv** and **rp** set to NULL.

*ECDSA_do_sign_ex()* computes a digital signature of the **dgst_len** bytes hash value **dgst** using the private key **eckey** and the optional pre-computed values **kinv** and **rp**. The signature is returned in a newly allocated **ECDSA_SIG** structure (or NULL on error).

*ECDSA_do_verify()* verifies that the signature **sig** is a valid ECDSA signature of the hash value **dgst** of size **dgst_len** using the public key **eckey**.

## RETURN VALUES

*ECDSA_size()* returns the maximum length signature or 0 on error.

*ECDSA_sign_setup()* and *ECDSA_sign()* return 1 if successful or 0 on error.

*ECDSA_verify()* and *ECDSA_do_verify()* return 1 for a valid signature, 0 for an invalid signature and -1 on error. The error codes can be obtained by *ERR_get_error(3)*.

## EXAMPLES

Creating a ECDSA signature of given SHA-1 hash value using the named curve secp192k1.

First step: create a EC_KEY object (note: this part is **not** ECDSA specific)

```
int ret;
ECDSA_SIG *sig;
EC_KEY *eckey;
eckey = EC_KEY_new_by_curve_name(NID_secp192k1);
if (eckey == NULL)
{
/* error */
}
if (!EC_KEY_generate_key(eckey))
{
/* error */
}
```

Second step: compute the ECDSA signature of a SHA-1 hash value using **ECDSA_do_sign**

```
sig = ECDSA_do_sign(digest, 20, eckey);
if (sig == NULL)
{
/* error */
}
```

or using **ECDSA_sign**

```
unsigned char *buffer, *pp;
int buf_len;
buf_len = ECDSA_size(eckey);
buffer = OPENSSL_malloc(buf_len);
pp = buffer;
if (!ECDSA_sign(0, dgst, dgstlen, pp, &buf_len, eckey);
{
/* error */
}
```

Third step: verify the created ECDSA signature using **ECDSA_do_verify**

```
ret = ECDSA_do_verify(digest, 20, sig, eckey);
```

or using **ECDSA_verify**

```
ret = ECDSA_verify(0, digest, 20, buffer, buf_len, eckey);
```

and finally evaluate the return value:

```
if (ret == -1)
{
/* error */
}
else if (ret == 0)
{
/* incorrect signature */
}
else /* ret == 1 */
{
/* signature ok */
}
```

## CONFORMING TO

ANSI X9.62, US Federal Information Processing Standard FIPS 186-2 (Digital Signature Standard, DSS)

## SEE ALSO

*dsa(3)*, *rsa(3)*

## HISTORY

The ecdsa implementation was first introduced in OpenSSL 0.9.8

## AUTHOR

Nils Larsch for the OpenSSL project (http://www.openssl.org).