



*Full credit is given to the above companies including the OS that this PDF file was generated!*

## **Red Hat Enterprise Linux Release 9.2 Manual Pages on 'EVP\_PKEY\_asn1\_set\_param\_check.3ossl' command**

```
$ man EVP_PKEY_asn1_set_param_check.3ossl
```

```
EVP_PKEY_ASN1_METHOD(3ossl)    OpenSSL    EVP_PKEY_ASN1_METHOD(3ossl)
```

### NAME

EVP\_PKEY\_ASN1\_METHOD, EVP\_PKEY\_asn1\_new, EVP\_PKEY\_asn1\_copy, EVP\_PKEY\_asn1\_free, EVP\_PKEY\_asn1\_add0, EVP\_PKEY\_asn1\_add\_alias, EVP\_PKEY\_asn1\_set\_public, EVP\_PKEY\_asn1\_set\_private, EVP\_PKEY\_asn1\_set\_param, EVP\_PKEY\_asn1\_set\_free, EVP\_PKEY\_asn1\_set\_ctrl, EVP\_PKEY\_asn1\_set\_item, EVP\_PKEY\_asn1\_set\_siginf, EVP\_PKEY\_asn1\_set\_check, EVP\_PKEY\_asn1\_set\_public\_check, EVP\_PKEY\_asn1\_set\_param\_check, EVP\_PKEY\_asn1\_set\_security\_bits, EVP\_PKEY\_asn1\_set\_set\_priv\_key, EVP\_PKEY\_asn1\_set\_set\_pub\_key, EVP\_PKEY\_asn1\_set\_get\_priv\_key, EVP\_PKEY\_asn1\_set\_get\_pub\_key, EVP\_PKEY\_get0\_asn1 - manipulating and registering EVP\_PKEY\_ASN1\_METHOD structure

### SYNOPSIS

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

typedef struct evp_pkey_asn1_method_st EVP_PKEY_ASN1_METHOD;

EVP_PKEY_ASN1_METHOD *EVP_PKEY_asn1_new(int id, int flags,
                                       const char *pem_str,
                                       const char *info);

void EVP_PKEY_asn1_copy(EVP_PKEY_ASN1_METHOD *dst,
                       const EVP_PKEY_ASN1_METHOD *src);

void EVP_PKEY_asn1_free(EVP_PKEY_ASN1_METHOD *ameth);

int EVP_PKEY_asn1_add0(const EVP_PKEY_ASN1_METHOD *ameth);
```

```

int EVP_PKEY_asn1_add_alias(int to, int from);

void EVP_PKEY_asn1_set_public(EVP_PKEY_ASN1_METHOD *ameth,
    int (*pub_decode) (EVP_PKEY *pk,
        const X509_PUBKEY *pub),
    int (*pub_encode) (X509_PUBKEY *pub,
        const EVP_PKEY *pk),
    int (*pub_cmp) (const EVP_PKEY *a,
        const EVP_PKEY *b),
    int (*pub_print) (BIO *out,
        const EVP_PKEY *pkey,
        int indent, ASN1_PCTX *pctx),
    int (*pkey_size) (const EVP_PKEY *pk),
    int (*pkey_bits) (const EVP_PKEY *pk));

void EVP_PKEY_asn1_set_private(EVP_PKEY_ASN1_METHOD *ameth,
    int (*priv_decode) (EVP_PKEY *pk,
        const PKCS8_PRIV_KEY_INFO
        *p8inf),
    int (*priv_encode) (PKCS8_PRIV_KEY_INFO *p8,
        const EVP_PKEY *pk),
    int (*priv_print) (BIO *out,
        const EVP_PKEY *pkey,
        int indent,
        ASN1_PCTX *pctx));

void EVP_PKEY_asn1_set_param(EVP_PKEY_ASN1_METHOD *ameth,
    int (*param_decode) (EVP_PKEY *pkey,
        const unsigned char **pder,
        int derlen),
    int (*param_encode) (const EVP_PKEY *pkey,
        unsigned char **pder),
    int (*param_missing) (const EVP_PKEY *pk),
    int (*param_copy) (EVP_PKEY *to,
        const EVP_PKEY *from),
    int (*param_cmp) (const EVP_PKEY *a,

```

```

        const EVP_PKEY *b),
    int (*param_print) (BIO *out,
        const EVP_PKEY *pkey,
        int indent,
        ASN1_PCTX *pctx));
void EVP_PKEY_asn1_set_free(EVP_PKEY_ASN1_METHOD *ameth,
    void (*pkey_free) (EVP_PKEY *pkey));
void EVP_PKEY_asn1_set_ctrl(EVP_PKEY_ASN1_METHOD *ameth,
    int (*pkey_ctrl) (EVP_PKEY *pkey, int op,
        long arg1, void *arg2));
void EVP_PKEY_asn1_set_item(EVP_PKEY_ASN1_METHOD *ameth,
    int (*item_verify) (EVP_MD_CTX *ctx,
        const ASN1_ITEM *it,
        void *asn,
        X509_ALGOR *a,
        ASN1_BIT_STRING *sig,
        EVP_PKEY *pkey),
    int (*item_sign) (EVP_MD_CTX *ctx,
        const ASN1_ITEM *it,
        void *asn,
        X509_ALGOR *alg1,
        X509_ALGOR *alg2,
        ASN1_BIT_STRING *sig));
void EVP_PKEY_asn1_set_siginf(EVP_PKEY_ASN1_METHOD *ameth,
    int (*siginf_set) (X509_SIG_INFO *siginf,
        const X509_ALGOR *alg,
        const ASN1_STRING *sig));
void EVP_PKEY_asn1_set_check(EVP_PKEY_ASN1_METHOD *ameth,
    int (*pkey_check) (const EVP_PKEY *pk));
void EVP_PKEY_asn1_set_public_check(EVP_PKEY_ASN1_METHOD *ameth,
    int (*pkey_pub_check) (const EVP_PKEY *pk));
void EVP_PKEY_asn1_set_param_check(EVP_PKEY_ASN1_METHOD *ameth,
    int (*pkey_param_check) (const EVP_PKEY *pk));

```

```

void EVP_PKEY_asn1_set_security_bits(EVP_PKEY_ASN1_METHOD *ameth,
                                     int (*pkey_security_bits) (const EVP_PKEY
                                                                *pk));

void EVP_PKEY_asn1_set_set_priv_key(EVP_PKEY_ASN1_METHOD *ameth,
                                     int (*set_priv_key) (EVP_PKEY *pk,
                                                         const unsigned char
                                                                *priv,
                                                         size_t len));

void EVP_PKEY_asn1_set_set_pub_key(EVP_PKEY_ASN1_METHOD *ameth,
                                    int (*set_pub_key) (EVP_PKEY *pk,
                                                         const unsigned char *pub,
                                                         size_t len));

void EVP_PKEY_asn1_set_get_priv_key(EVP_PKEY_ASN1_METHOD *ameth,
                                     int (*get_priv_key) (const EVP_PKEY *pk,
                                                         unsigned char *priv,
                                                         size_t *len));

void EVP_PKEY_asn1_set_get_pub_key(EVP_PKEY_ASN1_METHOD *ameth,
                                    int (*get_pub_key) (const EVP_PKEY *pk,
                                                         unsigned char *pub,
                                                         size_t *len));

const EVP_PKEY_ASN1_METHOD *EVP_PKEY_get0_asn1(const EVP_PKEY *pkey);

```

## DESCRIPTION

EVP\_PKEY\_ASN1\_METHOD is a structure which holds a set of ASN.1 conversion, printing and information methods for a specific public key algorithm.

There are two places where the EVP\_PKEY\_ASN1\_METHOD objects are stored:

one is a built-in array representing the standard methods for different algorithms, and the other one is a stack of user-defined application-specific methods, which can be manipulated by using

EVP\_PKEY\_asn1\_add0(3).

## Methods

The methods are the underlying implementations of a particular public key algorithm present by the EVP\_PKEY object.

```

int (*pub_decode) (EVP_PKEY *pk, const X509_PUBKEY *pub);
int (*pub_encode) (X509_PUBKEY *pub, const EVP_PKEY *pk);
int (*pub_cmp) (const EVP_PKEY *a, const EVP_PKEY *b);
int (*pub_print) (BIO *out, const EVP_PKEY *pkey, int indent,
                 ASN1_PCTX *pctx);

```

The `pub_decode()` and `pub_encode()` methods are called to decode / encode X509\_PUBKEY ASN.1 parameters to / from pk. They MUST return 0 on error, 1 on success. They're called by `X509_PUBKEY_get0(3)` and `X509_PUBKEY_set(3)`.

The `pub_cmp()` method is called when two public keys are to be compared. It MUST return 1 when the keys are equal, 0 otherwise. It's called by `EVP_PKEY_eq(3)`.

The `pub_print()` method is called to print a public key in humanly readable text to out, indented indent spaces. It MUST return 0 on error, 1 on success. It's called by `EVP_PKEY_print_public(3)`.

```

int (*priv_decode) (EVP_PKEY *pk, const PKCS8_PRIV_KEY_INFO *p8inf);
int (*priv_encode) (PKCS8_PRIV_KEY_INFO *p8, const EVP_PKEY *pk);
int (*priv_print) (BIO *out, const EVP_PKEY *pkey, int indent,
                 ASN1_PCTX *pctx);

```

The `priv_decode()` and `priv_encode()` methods are called to decode / encode PKCS8\_PRIV\_KEY\_INFO from private key to / from pk. They MUST return 0 on error, 1 on success. They're called by `EVP_PKCS82PKEY(3)` and `EVP_PKEY2PKCS8(3)`.

The `priv_print()` method is called to print a private key in humanly readable text to out, indented indent spaces. It MUST return 0 on error, 1 on success. It's called by `EVP_PKEY_print_private(3)`.

```

int (*pkey_size) (const EVP_PKEY *pk);
int (*pkey_bits) (const EVP_PKEY *pk);
int (*pkey_security_bits) (const EVP_PKEY *pk);

```

The `pkey_size()` method returns the key size in bytes. It's called by `EVP_PKEY_get_size(3)`.

The `pkey_bits()` method returns the key size in bits. It's called by `EVP_PKEY_get_bits(3)`.

```

int (*param_decode) (EVP_PKEY *pkey,
                    const unsigned char **pder, int derlen);
int (*param_encode) (const EVP_PKEY *pkey, unsigned char **pder);
int (*param_missing) (const EVP_PKEY *pk);
int (*param_copy) (EVP_PKEY *to, const EVP_PKEY *from);
int (*param_cmp) (const EVP_PKEY *a, const EVP_PKEY *b);
int (*param_print) (BIO *out, const EVP_PKEY *pkey, int indent,
                  ASN1_PCTX *pctx);

```

The param\_decode() and param\_encode() methods are called to decode / encode DER formatted parameters to / from pk. They MUST return 0 on error, 1 on success. They're called by PEM\_read\_bio\_Parameters(3) and the file: OSSL\_STORE\_LOADER(3).

The param\_missing() method returns 0 if a key parameter is missing, otherwise 1. It's called by EVP\_PKEY\_missing\_parameters(3).

The param\_copy() method copies key parameters from from to to. It MUST return 0 on error, 1 on success. It's called by EVP\_PKEY\_copy\_parameters(3).

The param\_cmp() method compares the parameters of keys a and b. It MUST return 1 when the keys are equal, 0 when not equal, or a negative number on error. It's called by EVP\_PKEY\_parameters\_eq(3).

The param\_print() method prints the private key parameters in humanly readable text to out, indented indent spaces. It MUST return 0 on error, 1 on success. It's called by EVP\_PKEY\_print\_params(3).

```

int (*sig_print) (BIO *out,
                const X509_ALGOR *sigalg, const ASN1_STRING *sig,
                int indent, ASN1_PCTX *pctx);

```

The sig\_print() method prints a signature in humanly readable text to out, indented indent spaces. sigalg contains the exact signature algorithm. If the signature in sig doesn't correspond to what this method expects, X509\_signature\_dump() must be used as a last resort.

It MUST return 0 on error, 1 on success. It's called by X509\_signature\_print(3).

```

void (*pkey_free) (EVP_PKEY *pkey);

```

The `pkey_free()` method helps freeing the internals of `pkey`. It's called by `EVP_PKEY_free(3)`, `EVP_PKEY_set_type(3)`, `EVP_PKEY_set_type_str(3)`, and `EVP_PKEY_assign(3)`.

```
int (*pkey_ctrl) (EVP_PKEY *pkey, int op, long arg1, void *arg2);
```

The `pkey_ctrl()` method adds extra algorithm specific control. It's called by `EVP_PKEY_get_default_digest_nid(3)`,

```
EVP_PKEY_set1_encoded_public_key(3),
```

```
EVP_PKEY_get1_encoded_public_key(3), PKCS7_SIGNER_INFO_set(3),
```

```
PKCS7_RECIP_INFO_set(3), ...
```

```
int (*old_priv_decode) (EVP_PKEY *pkey,
```

```
    const unsigned char **pder, int derlen);
```

```
int (*old_priv_encode) (const EVP_PKEY *pkey, unsigned char **pder);
```

The `old_priv_decode()` and `old_priv_encode()` methods decode / encode they private key `pkey` from / to a DER formatted array. These are exclusively used to help decoding / encoding older (pre PKCS#8) PEM formatted encrypted private keys. `old_priv_decode()` MUST return 0 on error, 1 on success. `old_priv_encode()` MUST the return same kind of values as `i2d_PrivateKey()`. They're called by `d2i_PrivateKey(3)` and `i2d_PrivateKey(3)`.

```
int (*item_verify) (EVP_MD_CTX *ctx, const ASN1_ITEM *it, void *asn,
```

```
    X509_ALGOR *a, ASN1_BIT_STRING *sig, EVP_PKEY *pkey);
```

```
int (*item_sign) (EVP_MD_CTX *ctx, const ASN1_ITEM *it, void *asn,
```

```
    X509_ALGOR *alg1, X509_ALGOR *alg2,
```

```
    ASN1_BIT_STRING *sig);
```

The `item_sign()` and `item_verify()` methods make it possible to have algorithm specific signatures and verification of them.

`item_sign()` MUST return one of:

<=0 error

1 `item_sign()` did everything, OpenSSL internals just needs to pass the signature length back.

2 `item_sign()` did nothing, OpenSSL internal standard routines are expected to continue with the default signature production.

3 `item_sign()` set the algorithm identifier `algor1` and `algor2`, OpenSSL

internals should just sign using those algorithms.

item\_verify() MUST return one of:

<=0 error

1 item\_sign() did everything, OpenSSL internals just needs to pass the signature length back.

2 item\_sign() did nothing, OpenSSL internal standard routines are expected to continue with the default signature production.

item\_verify() and item\_sign() are called by ASN1\_item\_verify(3) and

ASN1\_item\_sign(3), and by extension, X509\_verify(3),

X509\_REQ\_verify(3), X509\_sign(3), X509\_REQ\_sign(3), ...

```
int (*siginf_set) (X509_SIG_INFO *siginf, const X509_ALGOR *alg,
                  const ASN1_STRING *sig);
```

The siginf\_set() method is used to set custom X509\_SIG\_INFO parameters.

It MUST return 0 on error, or 1 on success. It's called as part of

X509\_check\_purpose(3), X509\_check\_ca(3) and X509\_check\_issued(3).

```
int (*pkey_check) (const EVP_PKEY *pk);
```

```
int (*pkey_public_check) (const EVP_PKEY *pk);
```

```
int (*pkey_param_check) (const EVP_PKEY *pk);
```

The pkey\_check(), pkey\_public\_check() and pkey\_param\_check() methods

are used to check the validity of pk for key-pair, public component and

parameters, respectively. They MUST return 0 for an invalid key, or 1

for a valid key. They are called by EVP\_PKEY\_check(3),

EVP\_PKEY\_public\_check(3) and EVP\_PKEY\_param\_check(3) respectively.

```
int (*set_priv_key) (EVP_PKEY *pk, const unsigned char *priv, size_t len);
```

```
int (*set_pub_key) (EVP_PKEY *pk, const unsigned char *pub, size_t len);
```

The set\_priv\_key() and set\_pub\_key() methods are used to set the raw

private and public key data for an EVP\_PKEY. They MUST return 0 on

error, or 1 on success. They are called by

EVP\_PKEY\_new\_raw\_private\_key(3), and EVP\_PKEY\_new\_raw\_public\_key(3)

respectively.

```
size_t (*dirty) (const EVP_PKEY *pk);
```

```
void (*export_to) (const EVP_PKEY *pk, EVP_KEYMGMT *keymgmt);
```

dirty\_cnt() returns the internal key's dirty count. This can be used

to synchronise different copies of the same keys.

The `export_to()` method exports the key material from the given key to a provider, through the `EVP_KEYMGMT(3)` interface, if that provider supports importing key material.

## Functions

`EVP_PKEY_asn1_new()` creates and returns a new `EVP_PKEY_ASN1_METHOD` object, and associates the given `id`, `flags`, `pem_str` and `info`. `id` is a NID, `pem_str` is the PEM type string, `info` is a descriptive string. The following flags are supported:

`ASN1_PKEY_SIGPARAM_NULL`

If `ASN1_PKEY_SIGPARAM_NULL` is set, then the signature algorithm parameters are given the type `V_ASN1_NULL` by default, otherwise they will be given the type `V_ASN1_UNDEF` (i.e. the parameter is omitted).

See `X509_ALGOR_set0(3)` for more information.

`EVP_PKEY_asn1_copy()` copies an `EVP_PKEY_ASN1_METHOD` object from `src` to `dst`. This function is not thread safe, it's recommended to only use this when initializing the application.

`EVP_PKEY_asn1_free()` frees an existing `EVP_PKEY_ASN1_METHOD` pointed by `ameth`.

`EVP_PKEY_asn1_add0()` adds `ameth` to the user defined stack of methods unless another `EVP_PKEY_ASN1_METHOD` with the same NID is already there.

This function is not thread safe, it's recommended to only use this when initializing the application.

`EVP_PKEY_asn1_add_alias()` creates an alias with the NID to for the `EVP_PKEY_ASN1_METHOD` with NID from unless another `EVP_PKEY_ASN1_METHOD` with the same NID is already added. This function is not thread safe, it's recommended to only use this when initializing the application.

`EVP_PKEY_asn1_set_public()`, `EVP_PKEY_asn1_set_private()`,

`EVP_PKEY_asn1_set_param()`, `EVP_PKEY_asn1_set_free()`,

`EVP_PKEY_asn1_set_ctrl()`, `EVP_PKEY_asn1_set_item()`,

`EVP_PKEY_asn1_set_siginf()`, `EVP_PKEY_asn1_set_check()`,

`EVP_PKEY_asn1_set_public_check()`, `EVP_PKEY_asn1_set_param_check()`,

`EVP_PKEY_asn1_set_security_bits()`, `EVP_PKEY_asn1_set_set_priv_key()`,

EVP\_PKEY\_asn1\_set\_set\_pub\_key(), EVP\_PKEY\_asn1\_set\_get\_priv\_key() and EVP\_PKEY\_asn1\_set\_get\_pub\_key() set the diverse methods of the given EVP\_PKEY\_ASN1\_METHOD object.  
EVP\_PKEY\_get0\_asn1() finds the EVP\_PKEY\_ASN1\_METHOD associated with the key pkey.

## RETURN VALUES

EVP\_PKEY\_asn1\_new() returns NULL on error, or a pointer to an EVP\_PKEY\_ASN1\_METHOD object otherwise.  
EVP\_PKEY\_asn1\_add0() and EVP\_PKEY\_asn1\_add\_alias() return 0 on error, or 1 on success.  
EVP\_PKEY\_get0\_asn1() returns NULL on error, or a pointer to a constant EVP\_PKEY\_ASN1\_METHOD object otherwise.

## HISTORY

The signature of the pub\_decode functional argument of EVP\_PKEY\_asn1\_set\_public() has changed in OpenSSL 3.0 so its pub parameter is now constified.

## COPYRIGHT

Copyright 2017-2021 The OpenSSL Project Authors. All Rights Reserved.  
Licensed under the Apache License 2.0 (the "License"). You may not use this file except in compliance with the License. You can obtain a copy in the file LICENSE in the source distribution or at <https://www.openssl.org/source/license.html>.

3.0.7                    2023-07-13    EVP\_PKEY\_ASN1\_METHOD(3ossl)