

NAME

EVP_CIPHER_meth_new, EVP_CIPHER_meth_dup, EVP_CIPHER_meth_free,
 EVP_CIPHER_meth_set_iv_length, EVP_CIPHER_meth_set_flags,
 EVP_CIPHER_meth_set_impl_ctx_size, EVP_CIPHER_meth_set_init,
 EVP_CIPHER_meth_set_do_cipher, EVP_CIPHER_meth_set_cleanup,
 EVP_CIPHER_meth_set_set_asn1_params, EVP_CIPHER_meth_set_get_asn1_params,
 EVP_CIPHER_meth_set_ctrl, EVP_CIPHER_meth_get_init, EVP_CIPHER_meth_get_do_cipher,
 EVP_CIPHER_meth_get_cleanup, EVP_CIPHER_meth_get_set_asn1_params,
 EVP_CIPHER_meth_get_get_asn1_params, EVP_CIPHER_meth_get_ctrl - Routines to build up
 EVP_CIPHER methods

SYNOPSIS

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

The following functions have been deprecated since OpenSSL 3.0, and can be hidden entirely by defining **OPENSSL_API_COMPAT** with a suitable version value, see **openssl_user_macros(7)**:

```
EVP_CIPHER *EVP_CIPHER_meth_new(int cipher_type, int block_size, int key_len);
EVP_CIPHER *EVP_CIPHER_meth_dup(const EVP_CIPHER *cipher);
void EVP_CIPHER_meth_free(EVP_CIPHER *cipher);
```

```
int EVP_CIPHER_meth_set_iv_length(EVP_CIPHER *cipher, int iv_len);
int EVP_CIPHER_meth_set_flags(EVP_CIPHER *cipher, unsigned long flags);
int EVP_CIPHER_meth_set_impl_ctx_size(EVP_CIPHER *cipher, int ctx_size);
int EVP_CIPHER_meth_set_init(EVP_CIPHER *cipher,
    int (*init)(EVP_CIPHER_CTX *ctx,
        const unsigned char *key,
        const unsigned char *iv,
        int enc));
```

```
int EVP_CIPHER_meth_set_do_cipher(EVP_CIPHER *cipher,
    int (*do_cipher)(EVP_CIPHER_CTX *ctx,
        unsigned char *out,
        const unsigned char *in,
        size_t inl));
```

```
int EVP_CIPHER_meth_set_cleanup(EVP_CIPHER *cipher,
    int (*cleanup)(EVP_CIPHER_CTX *));
```

```
int EVP_CIPHER_meth_set_set_asn1_params(EVP_CIPHER *cipher,
    int (*set_asn1_parameters)(EVP_CIPHER_CTX *,
        ASN1_TYPE *));
```

```
int EVP_CIPHER_meth_set_get_asn1_params(EVP_CIPHER *cipher,
```

```

        int (*get_asn1_parameters)(EVP_CIPHER_CTX *,
                                   ASN1_TYPE *);
int EVP_CIPHER_meth_set_ctrl(EVP_CIPHER *cipher,
                             int (*ctrl)(EVP_CIPHER_CTX *, int type,
                                           int arg, void *ptr));

int (*EVP_CIPHER_meth_get_init(const EVP_CIPHER *cipher))(EVP_CIPHER_CTX *ctx,
                                                           const unsigned char *key,
                                                           const unsigned char *iv,
                                                           int enc);
int (*EVP_CIPHER_meth_get_do_cipher(const EVP_CIPHER *cipher))(EVP_CIPHER_CTX *ctx,
                                                                unsigned char *out,
                                                                const unsigned char *in,
                                                                size_t inl);
int (*EVP_CIPHER_meth_get_cleanup(const EVP_CIPHER *cipher))(EVP_CIPHER_CTX *);
int (*EVP_CIPHER_meth_get_set_asn1_params(const EVP_CIPHER *cipher))(EVP_CIPHER_CTX *,
                                                                       ASN1_TYPE *);
int (*EVP_CIPHER_meth_get_get_asn1_params(const EVP_CIPHER *cipher))(EVP_CIPHER_CTX *,
                                                                       ASN1_TYPE *);
int (*EVP_CIPHER_meth_get_ctrl(const EVP_CIPHER *cipher))(EVP_CIPHER_CTX *,
                                                           int type, int arg,
                                                           void *ptr);

```

DESCRIPTION

All of the functions described on this page are deprecated. Applications should instead use the `OSSL_PROVIDER` APIs.

The **EVP_CIPHER** type is a structure for symmetric cipher method implementation.

EVP_CIPHER_meth_new() creates a new **EVP_CIPHER** structure.

EVP_CIPHER_meth_dup() creates a copy of **cipher**.

EVP_CIPHER_meth_free() destroys a **EVP_CIPHER** structure.

EVP_CIPHER_meth_set_iv_length() sets the length of the IV. This is only needed when the implemented cipher mode requires it.

EVP_CIPHER_meth_set_flags() sets the flags to describe optional behaviours in the particular **cipher**. With the exception of cipher modes, of which only one may be present, several flags can be or'd

together. The available flags are:

EVP_CIPH_STREAM_CIPHER, EVP_CIPH_ECB_MODE, EVP_CIPH_CBC_MODE,
EVP_CIPH_CFB_MODE, EVP_CIPH_OFB_MODE, EVP_CIPH_CTR_MODE,
EVP_CIPH_GCM_MODE, EVP_CIPH_CCM_MODE, EVP_CIPH_XTS_MODE,
EVP_CIPH_WRAP_MODE, EVP_CIPH_OCB_MODE, EVP_CIPH_SIV_MODE

The cipher mode.

EVP_CIPH_VARIABLE_LENGTH

This cipher is of variable length.

EVP_CIPH_CUSTOM_IV

Storing and initialising the IV is left entirely to the implementation.

EVP_CIPH_ALWAYS_CALL_INIT

Set this if the implementation's **init()** function should be called even if **key** is **NULL**.

EVP_CIPH_CTRL_INIT

Set this to have the implementation's **ctrl()** function called with command code **EVP_CTRL_INIT** early in its setup.

EVP_CIPH_CUSTOM_KEY_LENGTH

Checking and setting the key length after creating the **EVP_CIPHER** is left to the implementation. Whenever someone uses **EVP_CIPHER_CTX_set_key_length()** on a **EVP_CIPHER** with this flag set, the implementation's **ctrl()** function will be called with the control code **EVP_CTRL_SET_KEY_LENGTH** and the key length in **arg**.

EVP_CIPH_NO_PADDING

Don't use standard block padding.

EVP_CIPH_RAND_KEY

Making a key with random content is left to the implementation. This is done by calling the implementation's **ctrl()** function with the control code **EVP_CTRL_RAND_KEY** and the pointer to the key memory storage in **ptr**.

EVP_CIPH_CUSTOM_COPY

Set this to have the implementation's **ctrl()** function called with command code **EVP_CTRL_COPY** at the end of **EVP_CIPHER_CTX_copy()**. The intended use is for further things to deal with after the implementation specific data block has been copied. The destination **EVP_CIPHER_CTX** is passed to the control with the **ptr** parameter. The implementation specific

data block is reached with **EVP_CIPHER_CTX_get_cipher_data()**.

EVP_CIPH_FLAG_DEFAULT_ASN1

Use the default EVP routines to pass IV to and from ASN.1.

EVP_CIPH_FLAG_LENGTH_BITS

Signals that the length of the input buffer for encryption / decryption is to be understood as the number of bits instead of bytes for this implementation. This is only useful for CFB1 ciphers.

EVP_CIPH_FLAG_CTS

Indicates that the cipher uses ciphertext stealing. This is currently used to indicate that the cipher is a one shot that only allows a single call to **EVP_CipherUpdate()**.

EVP_CIPH_FLAG_CUSTOM_CIPHER

This indicates that the implementation takes care of everything, including padding, buffering and finalization. The EVP routines will simply give them control and do nothing more.

EVP_CIPH_FLAG_AEAD_CIPHER

This indicates that this is an AEAD cipher implementation.

EVP_CIPH_FLAG_TLS1_1_MULTIBLOCK

Allow interleaving of crypto blocks, a particular optimization only applicable to certain TLS ciphers.

EVP_CIPHER_meth_set_impl_ctx_size() sets the size of the EVP_CIPHER's implementation context so that it can be automatically allocated.

EVP_CIPHER_meth_set_init() sets the cipher init function for **cipher**. The cipher init function is called by **EVP_CipherInit()**, **EVP_CipherInit_ex()**, **EVP_EncryptInit()**, **EVP_EncryptInit_ex()**, **EVP_DecryptInit()**, **EVP_DecryptInit_ex()**.

EVP_CIPHER_meth_set_do_cipher() sets the cipher function for **cipher**. The cipher function is called by **EVP_CipherUpdate()**, **EVP_EncryptUpdate()**, **EVP_DecryptUpdate()**, **EVP_CipherFinal()**, **EVP_EncryptFinal()**, **EVP_EncryptFinal_ex()**, **EVP_DecryptFinal()** and **EVP_DecryptFinal_ex()**.

EVP_CIPHER_meth_set_cleanup() sets the function for **cipher** to do extra cleanup before the method's private data structure is cleaned out and freed. Note that the cleanup function is passed a **EVP_CIPHER_CTX ***, the private data structure is then available with **EVP_CIPHER_CTX_get_cipher_data()**. This cleanup function is called by **EVP_CIPHER_CTX_reset()** and **EVP_CIPHER_CTX_free()**.

EVP_CIPHER_meth_set_set_asn1_params() sets the function for **cipher** to set the AlgorithmIdentifier "parameter" based on the passed cipher. This function is called by **EVP_CIPHER_param_to_asn1()**. **EVP_CIPHER_meth_set_get_asn1_params()** sets the function for **cipher** that sets the cipher parameters based on an ASN.1 AlgorithmIdentifier "parameter". Both these functions are needed when there is a need for custom data (more or other than the cipher IV). They are called by **EVP_CIPHER_param_to_asn1()** and **EVP_CIPHER_asn1_to_param()** respectively if defined.

EVP_CIPHER_meth_set_ctrl() sets the control function for **cipher**.

EVP_CIPHER_meth_get_init(), **EVP_CIPHER_meth_get_do_cipher()**, **EVP_CIPHER_meth_get_cleanup()**, **EVP_CIPHER_meth_get_set_asn1_params()**, **EVP_CIPHER_meth_get_get_asn1_params()** and **EVP_CIPHER_meth_get_ctrl()** are all used to retrieve the method data given with the **EVP_CIPHER_meth_set_***() functions above.

RETURN VALUES

EVP_CIPHER_meth_new() and **EVP_CIPHER_meth_dup()** return a pointer to a newly created **EVP_CIPHER**, or NULL on failure. All **EVP_CIPHER_meth_set_***() functions return 1. All **EVP_CIPHER_meth_get_***() functions return pointers to their respective **cipher** function.

SEE ALSO

EVP_EncryptInit(3)

HISTORY

All of these functions were deprecated in OpenSSL 3.0.

The functions described here were added in OpenSSL 1.1.0. The **EVP_CIPHER** structure created with these functions became reference counted in OpenSSL 3.0.

COPYRIGHT

Copyright 2016-2021 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>.