

Happy Key: HPKE implementation (RFC9180)

<https://github.com/sftcd/happykey>

Generated by Doxygen 1.9.1



<b>1 Data Structure Index</b>	<b>1</b>
1.1 Data Structures . . . . .	1
<b>2 File Index</b>	<b>3</b>
2.1 File List . . . . .	3
<b>3 Data Structure Documentation</b>	<b>5</b>
3.1 OSSL_HPKE_SUITE Struct Reference . . . . .	5
3.1.1 Detailed Description . . . . .	5
<b>4 File Documentation</b>	<b>7</b>
4.1 hpke.h File Reference . . . . .	7
4.1.1 Detailed Description . . . . .	10
4.1.2 Macro Definition Documentation . . . . .	10
4.1.2.1 OSSL_HPKE_SUITE_DEFAULT . . . . .	10
4.1.3 Function Documentation . . . . .	11
4.1.3.1 OSSL_HPKE_CTX_export() . . . . .	11
4.1.3.2 OSSL_HPKE_CTX_free() . . . . .	11
4.1.3.3 OSSL_HPKE_CTX_get0_seq() . . . . .	11
4.1.3.4 OSSL_HPKE_CTX_new() . . . . .	13
4.1.3.5 OSSL_HPKE_CTX_set1_authpriv() . . . . .	13
4.1.3.6 OSSL_HPKE_CTX_set1_authpub() . . . . .	14
4.1.3.7 OSSL_HPKE_CTX_set1_psk() . . . . .	14
4.1.3.8 OSSL_HPKE_CTX_set1_senderpriv() . . . . .	15
4.1.3.9 OSSL_HPKE_CTX_set1_seq() . . . . .	15
4.1.3.10 OSSL_HPKE_dec() . . . . .	15
4.1.3.11 OSSL_HPKE_enc() . . . . .	17
4.1.3.12 OSSL_HPKE_expansion() . . . . .	18
4.1.3.13 OSSL_HPKE_good4grease() . . . . .	18
4.1.3.14 OSSL_HPKE_keygen() . . . . .	19
4.1.3.15 OSSL_HPKE_recipient_open() . . . . .	20
4.1.3.16 OSSL_HPKE_sender_seal() . . . . .	21
4.1.3.17 OSSL_HPKE_str2suite() . . . . .	21
4.1.3.18 OSSL_HPKE_suite_check() . . . . .	22
<b>Index</b>	<b>23</b>



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">OSSL_HPKE_SUITE</a>	
Ciphersuite combination . . . . .	<a href="#">5</a>



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">hpke.h</a>	APIs and data structures for HPKE (RFC9180)	7
------------------------	---	---





## Chapter 3

# Data Structure Documentation

### 3.1 OSSL\_HPKE\_SUITE Struct Reference

ciphersuite combination

```
#include <hpke.h>
```

#### Data Fields

- uint16\_t [kem\\_id](#)  
*Key Encryption Method id.*
- uint16\_t [kdf\\_id](#)  
*Key Derivation Function id.*
- uint16\_t [aead\\_id](#)  
*AEAD alg id.*

#### 3.1.1 Detailed Description

ciphersuite combination

The documentation for this struct was generated from the following file:

- [hpke.h](#)



## Chapter 4

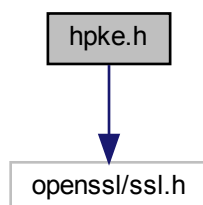
# File Documentation

### 4.1 hpke.h File Reference

APIs and data structures for HPKE (RFC9180).

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

Include dependency graph for hpke.h:



#### Data Structures

- struct [OSSL\\_HPKE\\_SUITE](#)  
*ciphersuite combination*

#### Macros

- #define [OSSL\\_HPKE\\_MODE\\_BASE](#) 0  
*Base mode*
- #define [OSSL\\_HPKE\\_MODE\\_PSK](#) 1  
*Pre-shared key mode.*
- #define [OSSL\\_HPKE\\_MODE\\_AUTH](#) 2  
*Authenticated mode.*

- #define `OSSL_HPKE_MODE_PSKAUTH` 3  
*PSK+authenticated mode.*
- #define `OSSL_HPKE_KEM_ID_RESERVED` 0x0000  
*not used*
- #define `OSSL_HPKE_KEM_ID_P256` 0x0010  
*NIST P-256.*
- #define `OSSL_HPKE_KEM_ID_P384` 0x0011  
*NIST P-256.*
- #define `OSSL_HPKE_KEM_ID_P521` 0x0012  
*NIST P-521.*
- #define `OSSL_HPKE_KEM_ID_25519` 0x0020  
*Curve25519.*
- #define `OSSL_HPKE_KEM_ID_448` 0x0021  
*Curve448.*
- #define `OSSL_HPKE_KDF_ID_RESERVED` 0x0000  
*not used*
- #define `OSSL_HPKE_KDF_ID_HKDF_SHA256` 0x0001  
*HKDF-SHA256.*
- #define `OSSL_HPKE_KDF_ID_HKDF_SHA384` 0x0002  
*HKDF-SHA384.*
- #define `OSSL_HPKE_KDF_ID_HKDF_SHA512` 0x0003  
*HKDF-SHA512.*
- #define `OSSL_HPKE_AEAD_ID_RESERVED` 0x0000  
*not used*
- #define `OSSL_HPKE_AEAD_ID_AES_GCM_128` 0x0001  
*AES-GCM-128.*
- #define `OSSL_HPKE_AEAD_ID_AES_GCM_256` 0x0002  
*AES-GCM-256.*
- #define `OSSL_HPKE_AEAD_ID_CHACHA_POLY1305` 0x0003  
*Chacha20-Poly1305.*
- #define `OSSL_HPKE_AEAD_ID_EXPORTONLY` 0xFFFF  
*export-only fake ID*
- #define `OSSL_HPKE_MODESTR_BASE` "base"  
*base mode (1)*
- #define `OSSL_HPKE_MODESTR_PSK` "psk"  
*psk mode (2)*
- #define `OSSL_HPKE_MODESTR_AUTH` "auth"  
*sender-key pair auth (3)*
- #define `OSSL_HPKE_MODESTR_PSKAUTH` "pskauth"  
*psk+sender-key pair (4)*
- #define `OSSL_HPKE_KEMSTR_P256` "P-256"  
*KEM id 0x10.*
- #define `OSSL_HPKE_KEMSTR_P384` "P-384"  
*KEM id 0x11.*
- #define `OSSL_HPKE_KEMSTR_P521` "P-521"  
*KEM id 0x12.*
- #define `OSSL_HPKE_KEMSTR_X25519` SN\_X25519  
*KEM id 0x20.*
- #define `OSSL_HPKE_KEMSTR_X448` SN\_X448  
*KEM id 0x21.*
- #define `OSSL_HPKE_KDFSTR_256` "hkdf-sha256"

- KDF id 1.*
  - #define `OSSL_HPKE_KDFSTR_384` "hkdf-sha384"
- KDF id 2.*
  - #define `OSSL_HPKE_KDFSTR_512` "hkdf-sha512"
- KDF id 3.*
  - #define `OSSL_HPKE_AEADSTR_AES128GCM` LN\_aes\_128\_gcm
- AEAD id 1.*
  - #define `OSSL_HPKE_AEADSTR_AES256GCM` LN\_aes\_256\_gcm
- AEAD id 2.*
  - #define `OSSL_HPKE_AEADSTR_CP` LN\_chacha20\_poly1305
- AEAD id 3.*
  - #define `OSSL_HPKE_AEADSTR_EXP` "exporter"
- AEAD id 0xff.*
  - #define `OSSL_HPKE_SUITE_DEFAULT`
- Suite constants, use this like: `OSSL_HPKE_SUITE myvar = OSSL_HPKE_SUITE_DEFAULT`;*
  - #define `OSSL_HPKE_MAXSIZE` 512

## Typedefs

- typedef struct ossl\_hpke\_ctx\_st `OSSL_HPKE_CTX`  
*opaque type for HPKE contexts*

## Functions

- `OSSL_HPKE_CTX * OSSL_HPKE_CTX_new` (int mode, `OSSL_HPKE_SUITE` suite, `OSSL_LIB_CTX` \*libctx, const char \*propq)  
*context creator*
- void `OSSL_HPKE_CTX_free` (`OSSL_HPKE_CTX` \*ctx)  
*free up storage for a HPKE context*
- int `OSSL_HPKE_CTX_set1_psk` (`OSSL_HPKE_CTX` \*ctx, const char \*pskid, const unsigned char \*psk, size\_t psklen)  
*set a PSK for an HPKE context*
- int `OSSL_HPKE_CTX_set1_senderpriv` (`OSSL_HPKE_CTX` \*ctx, EVP\_PKEY \*privp)  
*set a sender KEM private key for HPKE*
- int `OSSL_HPKE_CTX_set1_authpriv` (`OSSL_HPKE_CTX` \*ctx, EVP\_PKEY \*privp)  
*set a sender private key for HPKE authenticated modes*
- int `OSSL_HPKE_CTX_set1_authpub` (`OSSL_HPKE_CTX` \*ctx, const unsigned char \*pub, size\_t publen)  
*set a public key for HPKE authenticated modes*
- int `OSSL_HPKE_CTX_get0_seq` (`OSSL_HPKE_CTX` \*ctx, uint64\_t \*seq)  
*ask for the state of the sequence of seal/open calls*
- int `OSSL_HPKE_CTX_set1_seq` (`OSSL_HPKE_CTX` \*ctx, uint64\_t seq)  
*set the sequence value for seal/open calls*
- int `OSSL_HPKE_sender_seal` (`OSSL_HPKE_CTX` \*ctx, unsigned char \*enc, size\_t \*enclen, unsigned char \*ct, size\_t \*ctlen, unsigned char \*pub, size\_t publen, const unsigned char \*info, size\_t infolen, const unsigned char \*aad, size\_t aadlen, const unsigned char \*pt, size\_t ptlen)  
*sender seal function*
- int `OSSL_HPKE_recipient_open` (`OSSL_HPKE_CTX` \*ctx, unsigned char \*pt, size\_t \*ptlen, const unsigned char \*enc, size\_t \*enclen, EVP\_PKEY \*recippriv, const unsigned char \*info, size\_t infolen, const unsigned char \*aad, size\_t aadlen, const unsigned char \*ct, size\_t \*ctlen)  
*recipient open function*

- int [OSSL\\_HPKE\\_CTX\\_export](#) (OSSL\_HPKE\_CTX \*ctx, unsigned char \*secret, size\_t secret\_len, const unsigned char \*label, size\_t labellen)  
*generate a given-length secret based on context and label*
- int [OSSL\\_HPKE\\_keygen](#) (OSSL\_LIB\_CTX \*libctx, const char \*propq, unsigned int mode, [OSSL\\_HPKE\\_SUITE](#) suite, const unsigned char \*ikm, size\_t ikmlen, unsigned char \*pub, size\_t \*publen, EVP\_PKEY \*\*priv)  
*generate a key pair*
- int [OSSL\\_HPKE\\_suite\\_check](#) (OSSL\_HPKE\_SUITE suite)  
*check if a suite is supported locally*
- int [OSSL\\_HPKE\\_good4grease](#) (OSSL\_LIB\_CTX \*libctx, const char \*propq, [OSSL\\_HPKE\\_SUITE](#) \*suite\_in, [OSSL\\_HPKE\\_SUITE](#) \*suite, unsigned char \*pub, size\_t \*pub\_len, unsigned char \*cipher, size\_t cipher\_len)  
*get a (possibly) random suite, public key and ciphertext for GREASERS*
- int [OSSL\\_HPKE\\_str2suite](#) (const char \*str, [OSSL\\_HPKE\\_SUITE](#) \*suite)  
*map a string to a HPKE suite*
- int [OSSL\\_HPKE\\_expansion](#) (OSSL\_HPKE\_SUITE suite, size\_t \*enclen, size\_t clearlen, size\_t \*cipherlen)  
*tell the caller how big the ciphertext will be*
- int [OSSL\\_HPKE\\_enc](#) (OSSL\_LIB\_CTX \*libctx, const char \*propq, unsigned int mode, [OSSL\\_HPKE\\_SUITE](#) suite, const char \*pskid, const unsigned char \*psk, size\_t psklen, const unsigned char \*pub, size\_t publen, const unsigned char \*authpriv, size\_t authprivlen, EVP\_PKEY \*authpriv\_ev, const unsigned char \*clear, size\_t clearlen, const unsigned char \*aad, size\_t aadlen, const unsigned char \*info, size\_t infolen, const unsigned char \*seq, size\_t seqlen, unsigned char \*senderpub, size\_t \*senderpublen, EVP\_PKEY \*senderpriv, unsigned char \*cipher, size\_t \*cipherlen)  
*HPKE single-shot encryption function.*
- int [OSSL\\_HPKE\\_dec](#) (OSSL\_LIB\_CTX \*libctx, const char \*propq, unsigned int mode, [OSSL\\_HPKE\\_SUITE](#) suite, const char \*pskid, const unsigned char \*psk, size\_t psklen, const unsigned char \*pub, size\_t publen, const unsigned char \*priv, size\_t privlen, EVP\_PKEY \*evpriv, const unsigned char \*enc, size\_t enclen, const unsigned char \*cipher, size\_t cipherlen, const unsigned char \*aad, size\_t aadlen, const unsigned char \*info, size\_t infolen, const unsigned char \*seq, size\_t seqlen, unsigned char \*clear, size\_t \*clearlen)  
*HPKE single-shot decryption function.*

### 4.1.1 Detailed Description

APIs and data structures for HPKE (RFC9180).

### 4.1.2 Macro Definition Documentation

#### 4.1.2.1 OSSL\_HPKE\_SUITE\_DEFAULT

```
#define OSSL_HPKE_SUITE_DEFAULT
```

Value:

```
{ \
    OSSL_HPKE_KEM_ID_25519, \
    OSSL_HPKE_KDF_ID_HKDF_SHA256, \
    OSSL_HPKE_AEAD_ID_AES_GCM_128 \
}
```

Suite constants, use this like: [OSSL\\_HPKE\\_SUITE](#) myvar = OSSL\_HPKE\_SUITE\_DEFAULT;

### 4.1.3 Function Documentation

#### 4.1.3.1 OSSL\_HPKE\_CTX\_export()

```
int OSSL_HPKE_CTX_export (
    OSSL_HPKE_CTX * ctx,
    unsigned char * secret,
    size_t secret_len,
    const unsigned char * label,
    size_t labellen )
```

generate a given-length secret based on context and label

##### Parameters

<i>ctx</i>	is the HPKE context
<i>secret</i>	is the resulting secret that will be of length...
<i>secret_len</i>	is the desired output length
<i>label</i>	is a buffer to provide separation between secrets
<i>labellen</i>	is the length of the above

##### Returns

1 for good, 0 for error

The context has to have been used already for one encryption or decryption for this to work (as this is based on the negotiated "exporter\_secret" established via the HPKE operation).

#### 4.1.3.2 OSSL\_HPKE\_CTX\_free()

```
void OSSL_HPKE_CTX_free (
    OSSL_HPKE_CTX * ctx )
```

free up storage for a HPKE context

##### Parameters

<i>ctx</i>	is the pointer to be free'd (can be NULL)
------------	---

#### 4.1.3.3 OSSL\_HPKE\_CTX\_get0\_seq()

```
int OSSL_HPKE_CTX_get0_seq (
    OSSL_HPKE_CTX * ctx,
    uint64_t * seq )
```

ask for the state of the sequence of seal/open calls



## Parameters

<i>ctx</i>	is the pointer for the HPKE context
<i>seq</i>	returns the positive integer sequence number

## Returns

1 for success, 0 for error

The value returned is the next one to be used when sealing or opening (so as we start at zero this will be 1 after the first successful call to seal or open)

seq is a uint64\_t as that's what two other implementations chose

#### 4.1.3.4 OSSL\_HPKE\_CTX\_new()

```
OSSL_HPKE_CTX* OSSL_HPKE_CTX_new (
    int mode,
    OSSL_HPKE_SUITE suite,
    OSSL_LIB_CTX * libctx,
    const char * propq )
```

context creator

## Parameters

<i>mode</i>	is the desired HPKE mode
<i>suite</i>	specifies the KEM, KDF and AEAD to use
<i>libctx</i>	is the library context to use
<i>propq</i>	is a properties string for the library

## Returns

pointer to new context or NULL if error

#### 4.1.3.5 OSSL\_HPKE\_CTX\_set1\_authpriv()

```
int OSSL_HPKE_CTX_set1_authpriv (
    OSSL_HPKE_CTX * ctx,
    EVP_PKEY * privp )
```

set a sender private key for HPKE authenticated modes

## Parameters

<i>ctx</i>	is the pointer for the HPKE context
<i>privp</i>	is an EVP_PKEY form of the private key

**Returns**

1 for success, 0 for error

**4.1.3.6 OSSL\_HPKE\_CTX\_set1\_authpub()**

```
int OSSL_HPKE_CTX_set1_authpub (
    OSSL_HPKE_CTX * ctx,
    const unsigned char * pub,
    size_t publen )
```

set a public key for HPKE authenticated modes

**Parameters**

<i>ctx</i>	is the pointer for the HPKE context
<i>pub</i>	is an buffer form of the public key
<i>publen</i>	is the length of the above

**Returns**

1 for success, 0 for error

In all these APIs public keys are passed as buffers whereas private keys as passed as EVP\_PKEY pointers.

**4.1.3.7 OSSL\_HPKE\_CTX\_set1\_psk()**

```
int OSSL_HPKE_CTX_set1_psk (
    OSSL_HPKE_CTX * ctx,
    const char * pskid,
    const unsigned char * psk,
    size_t psklen )
```

set a PSK for an HPKE context

**Parameters**

<i>ctx</i>	is the pointer for the HPKE context
<i>pskid</i>	is a string identifying the PSK
<i>psk</i>	is the PSK buffer
<i>psklen</i>	is the size of the PSK

**Returns**

1 for success, 0 for error

#### 4.1.3.8 OSSL\_HPKE\_CTX\_set1\_senderpriv()

```
int OSSL_HPKE_CTX_set1_senderpriv (
    OSSL_HPKE_CTX * ctx,
    EVP_PKEY * privp )
```

set a sender KEM private key for HPKE

##### Parameters

<i>ctx</i>	is the pointer for the HPKE context
<i>privp</i>	is an EVP_PKEY form of the private key

##### Returns

1 for success, 0 for error

If no key is set via this API an ephemeral one will be generated in the first seal operation and used until the context is free'd. (Or until a subsequent call to this API replaces the key.) This suits senders who are typically clients.

#### 4.1.3.9 OSSL\_HPKE\_CTX\_set1\_seq()

```
int OSSL_HPKE_CTX_set1_seq (
    OSSL_HPKE_CTX * ctx,
    uint64_t seq )
```

set the sequence value for seal/open calls

##### Parameters

<i>ctx</i>	is the pointer for the HPKE context
<i>seq</i>	set the positive integer sequence number

##### Returns

1 for success, 0 for error

The next seal or open operation will use this value.

#### 4.1.3.10 OSSL\_HPKE\_dec()

```
int OSSL_HPKE_dec (
    OSSL_LIB_CTX * libctx,
    const char * propq,
    unsigned int mode,
    OSSL_HPKE_SUITE suite,
    const char * pskid,
    const unsigned char * psk,
    size_t psklen,
```

```

const unsigned char * pub,
size_t publen,
const unsigned char * priv,
size_t privlen,
EVP_PKEY * evppriv,
const unsigned char * enc,
size_t enclen,
const unsigned char * cipher,
size_t cipherlen,
const unsigned char * aad,
size_t aadlen,
const unsigned char * info,
size_t infolen,
const unsigned char * seq,
size_t seqlen,
unsigned char * clear,
size_t * clearlen )

```

HPKE single-shot decryption function.

#### Parameters

<i>libctx</i>	is the context to use (normally NULL)
<i>propq</i>	is a properties string
<i>mode</i>	is the HPKE mode
<i>suite</i>	is the ciphersuite to use
<i>pskid</i>	is the pskid string for a PSK mode (can be NULL)
<i>psk</i>	is the psk
<i>psklen</i>	is the psk length
<i>pub</i>	is the encoded public (authentication) key
<i>publen</i>	is the length of the public (authentication) key
<i>priv</i>	is the encoded private key
<i>privlen</i>	is the length of the private key
<i>evppriv</i>	is a pointer to an internal form of private key
<i>enc</i>	is the peer's public value
<i>enclen</i>	is the length of the peer's public value
<i>cipher</i>	is the ciphertext
<i>cipherlen</i>	is the length of the ciphertext
<i>aad</i>	is the encoded additional data
<i>aadlen</i>	is the length of the additional data
<i>info</i>	is the encoded info data (can be NULL)
<i>infolen</i>	is the length of the info data (can be zero)
<i>seq</i>	is the encoded sequence data (can be NULL)
<i>seqlen</i>	is the length of the sequence data (can be zero)
<i>clear</i>	is the encoded cleartext
<i>clearlen</i>	length of the input buffer for cleartext

#### Returns

1 for success, other for error (error returns can be non-zero)

## 4.1.3.11 OSSL\_HPKE\_enc()

```

int OSSL_HPKE_enc (
    OSSL_LIB_CTX * libctx,
    const char * propq,
    unsigned int mode,
    OSSL_HPKE_SUITE suite,
    const char * pskid,
    const unsigned char * psk,
    size_t psklen,
    const unsigned char * pub,
    size_t publen,
    const unsigned char * authpriv,
    size_t authprivlen,
    EVP_PKEY * authpriv_evpc,
    const unsigned char * clear,
    size_t clearlen,
    const unsigned char * aad,
    size_t aadlen,
    const unsigned char * info,
    size_t infolen,
    const unsigned char * seq,
    size_t seqlen,
    unsigned char * senderpub,
    size_t * senderpublen,
    EVP_PKEY * senderpriv,
    unsigned char * cipher,
    size_t * cipherlen )

```

HPKE single-shot encryption function.

This function generates an ephemeral ECDH value internally and provides the public component as an output that can be sent to the relevant private key holder along with the ciphertext.

## Parameters

<i>libctx</i>	is the context to use (normally NULL)
<i>propq</i>	is a properties string
<i>mode</i>	is the HPKE mode
<i>suite</i>	is the ciphersuite to use
<i>pskid</i>	is the pskid string for a PSK mode (can be NULL)
<i>psk</i>	is the psk
<i>psklen</i>	is the psk length
<i>pub</i>	is the encoded public key
<i>publen</i>	is the length of the public key
<i>authpriv</i>	is the encoded private (authentication) key
<i>authprivlen</i>	is the length of the private (authentication) key
<i>authpriv_evpc</i>	is the EVP_PKEY* form of private (authentication) key
<i>clear</i>	is the encoded cleartext
<i>clearlen</i>	is the length of the cleartext
<i>aad</i>	is the encoded additional data
<i>aadlen</i>	is the length of the additional data
<i>info</i>	is the encoded info data (can be NULL)
<i>infolen</i>	is the length of the info data (can be zero)
<i>seq</i>	is the encoded sequence data (can be NULL)

**Parameters**

<i>seqlen</i>	is the length of the sequence data (can be zero)
<i>senderpub</i>	is the input buffer for sender public key
<i>senderpublen</i>	length of the input buffer for sender's public key
<i>senderpriv</i>	is the sender's private key (if being re-used)
<i>cipher</i>	is the input buffer for ciphertext
<i>cipherlen</i>	is the length of the input buffer for ciphertext

**Returns**

1 for success, other for error (error returns can be non-zero)

**4.1.3.12 OSSL\_HPKE\_expansion()**

```
int OSSL_HPKE_expansion (
    OSSL_HPKE_SUITE suite,
    size_t * enclen,
    size_t clearlen,
    size_t * cipherlen )
```

tell the caller how big the ciphertext will be

**Parameters**

<i>suite</i>	is the suite to be used
<i>enclen</i>	points to what'll be enc length
<i>clearlen</i>	is the length of plaintext
<i>cipherlen</i>	points to what'll be ciphertext length

**Returns**

1 for success, otherwise failure

**4.1.3.13 OSSL\_HPKE\_good4grease()**

```
int OSSL_HPKE_good4grease (
    OSSL_LIB_CTX * libctx,
    const char * propq,
    OSSL_HPKE_SUITE * suite_in,
    OSSL_HPKE_SUITE * suite,
    unsigned char * pub,
    size_t * pub_len,
    unsigned char * cipher,
    size_t cipher_len )
```

get a (possibly) random suite, public key and ciphertext for GREASERs

## Parameters

<i>libctx</i>	is the context to use (normally NULL)
<i>propq</i>	is a properties string
<i>suite_in</i>	specifies the preferred suite or NULL for a random choice
<i>suite</i>	is the chosen or random suite
<i>pub</i>	a random value of the appropriate length for a sender public value
<i>pub_len</i>	is the length of pub (buffer size on input)
<i>cipher</i>	is a random value of the appropriate length for a ciphertext
<i>cipher_len</i>	is the length of cipher

## Returns

1 for success, otherwise failure

If *suite\_in* is provided that will be used (if supported). If *suite\_in* is NULL, a random suite (from those supported) will be selected. In all cases the output *pub* and *cipher* values will be appropriate random values for the selected suite.

## 4.1.3.14 OSSL\_HPKE\_keygen()

```
int OSSL_HPKE_keygen (
    OSSL_LIB_CTX * libctx,
    const char * propq,
    unsigned int mode,
    OSSL_HPKE_SUITE suite,
    const unsigned char * ikm,
    size_t ikmlen,
    unsigned char * pub,
    size_t * publen,
    EVP_PKEY ** priv )
```

generate a key pair

## Parameters

<i>libctx</i>	is the context to use (normally NULL)
<i>propq</i>	is a properties string
<i>mode</i>	is the mode (currently unused)
<i>suite</i>	is the ciphersuite (currently unused)
<i>ikm</i>	is IKM, if supplied
<i>ikmlen</i>	is the length of IKM, if supplied
<i>pub</i>	is the public value
<i>publen</i>	is the size of the public key buffer (exact length on output)
<i>priv</i>	is the private key

## Returns

1 for success, other for error (error returns can be non-zero)

Used for entities that will later receive HPKE values to decrypt or that want a private key for an AUTH mode. Currently, only the KEM from the suite is significant here. The `pub` output will typically be published so that others can encrypt to the private key holder using HPKE. (Or authenticate HPKE values from that sender.)

#### 4.1.3.15 OSSL\_HPKE\_recipient\_open()

```
int OSSL_HPKE_recipient_open (
    OSSL_HPKE_CTX * ctx,
    unsigned char * pt,
    size_t * ptlen,
    const unsigned char * enc,
    size_t enclen,
    EVP_PKEY * recippriv,
    const unsigned char * info,
    size_t infolen,
    const unsigned char * aad,
    size_t aadlen,
    const unsigned char * ct,
    size_t ctlen )
```

recipient open function

##### Parameters

<i>ctx</i>	is the pointer for the HPKE context
<i>pt</i>	is the plaintext
<i>ptlen</i>	is the size the above
<i>enc</i>	is the sender's ephemeral public value
<i>enclen</i>	is the size the above
<i>recippriv</i>	is the EVP_PKEY form of recipient private value
<i>info</i>	is the info parameter
<i>infolen</i>	is the size the above
<i>aad</i>	is the aad parameter
<i>aadlen</i>	is the size the above
<i>ct</i>	is the ciphertext output
<i>ctlen</i>	is the size the above

##### Returns

1 for success, 0 for error

This can be called once, or multiple, times.

The recipient private key is explicitly set here as recipients are likely to be servers with multiple long(ish) term private keys in memory at once and that may have to e.g. do trial decryptions.

The plaintext output (*pt*) will be smaller than the ciphertext input for all supported suites.



#### 4.1.3.16 OSSL\_HPKE\_sender\_seal()

```
int OSSL_HPKE_sender_seal (
    OSSL_HPKE_CTX * ctx,
    unsigned char * enc,
    size_t * enclen,
    unsigned char * ct,
    size_t * ctlen,
    unsigned char * pub,
    size_t publen,
    const unsigned char * info,
    size_t infolen,
    const unsigned char * aad,
    size_t aadlen,
    const unsigned char * pt,
    size_t ptlen )
```

sender seal function

##### Parameters

<i>ctx</i>	is the pointer for the HPKE context
<i>enc</i>	is the sender's ephemeral public value
<i>enclen</i>	is the size the above
<i>ct</i>	is the ciphertext output
<i>ctlen</i>	is the size the above
<i>pub</i>	is the recipient public key octets
<i>publen</i>	is the size the above
<i>recip</i>	is the EVP_PKEY form of recipient public value
<i>info</i>	is the info parameter
<i>infolen</i>	is the size the above
<i>aad</i>	is the aad parameter
<i>aadlen</i>	is the size the above
<i>pt</i>	is the plaintext
<i>ptlen</i>	is the size the above

##### Returns

1 for success, 0 for error

This can be called once, or multiple, times.

If no KEM private key has been set in the context an ephemeral key will be generated and used for the duration of the context.

The ciphertext buffer (ct) should be big enough to include the AEAD tag generated from encryptions and the *enc* buffer (the ephemeral public key) needs to be big enough for the relevant KEM. `OSSL_HPKE_expansion` can be used to determine the sizes needed.

#### 4.1.3.17 OSSL\_HPKE\_str2suite()

```
int OSSL_HPKE_str2suite (
    const char * str,
    OSSL_HPKE_SUITE * suite )
```

map a string to a HPKE suite

**Parameters**

<i>str</i>	is the string value
<i>suite</i>	is the resulting suite

**Returns**

1 for success, otherwise failure

An example good string is "x25519,hkdf-sha256,aes-128-gcm" Symbols are #define'd for the relevant labels, e.g. OSSL\_HPKE\_KEMSTR\_X25519. Numeric (decimal or hex) values with the relevant IANA codepoint values from RFC9180 may be used, e.g., "0x20,1,1" represents the same suite as the first example.

**4.1.3.18 OSSL\_HPKE\_suite\_check()**

```
int OSSL_HPKE_suite_check (
    OSSL_HPKE_SUITE suite )
```

check if a suite is supported locally

**Parameters**

<i>suite</i>	is the suite to check
--------------	-----------------------

**Returns**

1 for success, other for error (error returns can be non-zero)

# Index

hpke.h, [7](#)

- OSSL\_HPKE\_CTX\_export, [11](#)
- OSSL\_HPKE\_CTX\_free, [11](#)
- OSSL\_HPKE\_CTX\_get0\_seq, [11](#)
- OSSL\_HPKE\_CTX\_new, [13](#)
- OSSL\_HPKE\_CTX\_set1\_authpriv, [13](#)
- OSSL\_HPKE\_CTX\_set1\_authpub, [14](#)
- OSSL\_HPKE\_CTX\_set1\_psk, [14](#)
- OSSL\_HPKE\_CTX\_set1\_senderpriv, [14](#)
- OSSL\_HPKE\_CTX\_set1\_seq, [15](#)
- OSSL\_HPKE\_dec, [15](#)
- OSSL\_HPKE\_enc, [16](#)
- OSSL\_HPKE\_expansion, [18](#)
- OSSL\_HPKE\_good4grease, [18](#)
- OSSL\_HPKE\_keygen, [19](#)
- OSSL\_HPKE\_recipient\_open, [20](#)
- OSSL\_HPKE\_sender\_seal, [20](#)
- OSSL\_HPKE\_str2suite, [21](#)
- OSSL\_HPKE\_suite\_check, [22](#)
- OSSL\_HPKE\_SUITE\_DEFAULT, [10](#)

- OSSL\_HPKE\_CTX\_export
  - hpke.h, [11](#)
- OSSL\_HPKE\_CTX\_free
  - hpke.h, [11](#)
- OSSL\_HPKE\_CTX\_get0\_seq
  - hpke.h, [11](#)
- OSSL\_HPKE\_CTX\_new
  - hpke.h, [13](#)
- OSSL\_HPKE\_CTX\_set1\_authpriv
  - hpke.h, [13](#)
- OSSL\_HPKE\_CTX\_set1\_authpub
  - hpke.h, [14](#)
- OSSL\_HPKE\_CTX\_set1\_psk
  - hpke.h, [14](#)
- OSSL\_HPKE\_CTX\_set1\_senderpriv
  - hpke.h, [14](#)
- OSSL\_HPKE\_CTX\_set1\_seq
  - hpke.h, [15](#)
- OSSL\_HPKE\_dec
  - hpke.h, [15](#)
- OSSL\_HPKE\_enc
  - hpke.h, [16](#)
- OSSL\_HPKE\_expansion
  - hpke.h, [18](#)
- OSSL\_HPKE\_good4grease
  - hpke.h, [18](#)
- OSSL\_HPKE\_keygen
  - hpke.h, [19](#)
- OSSL\_HPKE\_recipient\_open

- hpke.h, [20](#)
- OSSL\_HPKE\_sender\_seal
  - hpke.h, [20](#)
- OSSL\_HPKE\_str2suite
  - hpke.h, [21](#)
- OSSL\_HPKE\_SUITE, [5](#)
- OSSL\_HPKE\_suite\_check
  - hpke.h, [22](#)
- OSSL\_HPKE\_SUITE\_DEFAULT
  - hpke.h, [10](#)