

Happy Key: HPKE implementation (draft-irtf-cfrg-hpke)

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>5</b>
3.1	hpke_aead_info_t Struct Reference . . . . .	5
3.1.1	Detailed Description . . . . .	5
3.2	hpke_kdf_info_t Struct Reference . . . . .	5
3.2.1	Detailed Description . . . . .	6
3.3	hpke_kem_info_t Struct Reference . . . . .	6
3.3.1	Detailed Description . . . . .	6
3.4	hpke_suite_t Struct Reference . . . . .	6
3.4.1	Detailed Description . . . . .	7
3.5	hpke_tv_encs_t Struct Reference . . . . .	7
3.5.1	Detailed Description . . . . .	7
3.6	hpke_tv_s Struct Reference . . . . .	8
3.6.1	Detailed Description . . . . .	9

<b>4 File Documentation</b>	<b>11</b>
4.1 hpke.c File Reference	11
4.1.1 Detailed Description	13
4.1.2 Function Documentation	13
4.1.2.1 figure_contextlen()	13
4.1.2.2 hpke_aead_dec()	13
4.1.2.3 hpke_aead_enc()	14
4.1.2.4 hpke_ah_decode()	15
4.1.2.5 hpke_dec()	16
4.1.2.6 hpke_do_kem()	17
4.1.2.7 hpke_enc()	17
4.1.2.8 hpke_EVP_PKEY_new_raw_nist_private_key()	18
4.1.2.9 hpke_EVP_PKEY_new_raw_nist_public_key()	19
4.1.2.10 hpke_expand()	19
4.1.2.11 hpke_extract()	20
4.1.2.12 hpke_kg()	20
4.1.2.13 hpke_make_context()	21
4.1.2.14 hpke_mode_check()	21
4.1.2.15 hpke_pbuf()	22
4.1.2.16 hpke_psk_check()	22
4.1.2.17 hpke_suite_check()	23
4.1.3 Variable Documentation	23
4.1.3.1 hpke_aead_tab	23
4.1.3.2 hpke_kdf_tab	24
4.1.3.3 hpke_kem_tab	24
4.1.3.4 zero_buf	24
4.2 hpke.h File Reference	25
4.2.1 Detailed Description	26
4.2.2 Macro Definition Documentation	27
4.2.2.1 HPKE_A2B	27

4.2.2.2	HPKE_SUITE_DEFAULT	27
4.2.3	Function Documentation	27
4.2.3.1	hpke_ah_decode()	27
4.2.3.2	hpke_dec()	28
4.2.3.3	hpke_enc()	29
4.2.3.4	hpke_kg()	30
4.3	hpketv.c File Reference	31
4.3.1	Detailed Description	32
4.3.2	Macro Definition Documentation	32
4.3.2.1	FAIL2BUILD	32
4.3.3	Function Documentation	32
4.3.3.1	hpke_tv_free()	32
4.3.3.2	hpke_tv_load()	33
4.3.3.3	hpke_tv_pick()	33
4.3.3.4	hpke_tv_print()	34
4.4	hpketv.h File Reference	34
4.4.1	Detailed Description	35
4.4.2	Typedef Documentation	35
4.4.2.1	hpke_tv_t	35
4.4.3	Function Documentation	35
4.4.3.1	hpke_tv_free()	35
4.4.3.2	hpke_tv_load()	36
4.4.3.3	hpke_tv_pick()	36
4.4.3.4	hpke_tv_print()	37
	<b>Index</b>	<b>39</b>



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">hpke_aead_info_t</a>	Info about an AEAD . . . . .	5
<a href="#">hpke_kdf_info_t</a>	Info about a KDF . . . . .	5
<a href="#">hpke_kem_info_t</a>	Info about a KEM . . . . .	6
<a href="#">hpke_suite_t</a>	Ciphersuite combination . . . . .	6
<a href="#">hpke_tv_encs_t</a>	Encryption(s) Test Vector structure using field names from published JSON file . . . . .	7
<a href="#">hpke_tv_s</a>	HKPE Test Vector structure using field names from published JSON file . . . . .	8





## Chapter 2

# File Index

### 2.1 File List

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

<a href="#">hpke.c</a>	An OpenSSL-based HPKE implementation following draft-irtf-cfrg-hpke . . . . .	11
<a href="#">hpke.h</a>	This has the data structures and prototypes (both internal and external) for an OpenSSL-based HPKE implementation following draft-irtf-cfrg-hpke . . . . .	25
<a href="#">hpketv.c</a>	Implementation related to test vectors for HPKE . . . . .	31
<a href="#">hpketv.h</a>	Header file related to test vectors for HPKE . . . . .	34



## Chapter 3

# Data Structure Documentation

### 3.1 hpke\_aead\_info\_t Struct Reference

info about an AEAD

#### Data Fields

- uint16\_t [aead\\_id](#)  
*code point for aead alg*
- const EVP\_CIPHER \*(\* [aead\\_init\\_func](#) )(void)  
*the aead we're using*
- size\_t [taglen](#)  
*aead tag len*
- size\_t [Nk](#)  
*size of a key for this aead*
- size\_t [Nn](#)  
*length of a nonce for this aead*

#### 3.1.1 Detailed Description

info about an AEAD

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

- [hpke.c](#)

### 3.2 hpke\_kdf\_info\_t Struct Reference

info about a KDF

## Data Fields

- `uint16_t kdf_id`  
*code point for KDF*
- `const EVP_MD *(* hash_init_func )(void)`  
*the hash alg we're using*
- `size_t Nh`  
*length of hash/extract output*

### 3.2.1 Detailed Description

info about a KDF

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

- [hpke.c](#)

## 3.3 hpke\_kem\_info\_t Struct Reference

info about a KEM

## Data Fields

- `uint16_t kem_id`  
*code point for key encipherment method*
- `int groupid`  
*NID of KEM.*
- `size_t Nenc`  
*length of encapsulated key*
- `size_t Npk`  
*length of public key*
- `size_t Npriv`  
*length of raw private key*

### 3.3.1 Detailed Description

info about a KEM

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

- [hpke.c](#)

## 3.4 hpke\_suite\_t 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](#)  
*Authenticated Encryption with Associated Data id.*

### 3.4.1 Detailed Description

ciphersuite combination

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

- [hpke.h](#)

## 3.5 hpke\_tv\_encs\_t Struct Reference

Encryption(s) Test Vector structure using field names from published JSON file.

```
#include <hpketv.h>
```

## Data Fields

- const char \* [aad](#)  
*ascii-hex encoded additional authenticated data*
- const char \* [plaintext](#)  
*aascii-hex encoded plaintext*
- const char \* [ciphertext](#)  
*ascii-hex encoded ciphertext*

### 3.5.1 Detailed Description

Encryption(s) Test Vector structure using field names from published JSON file.

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

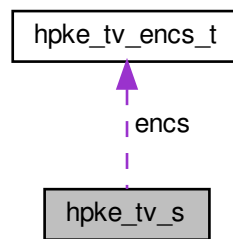
- [hpketv.h](#)

### 3.6 hpke\_tv\_s Struct Reference

HKPE Test Vector structure using field names from published JSON file.

```
#include <hpketv.h>
```

Collaboration diagram for hpke\_tv\_s:



#### Data Fields

- uint8\_t **mode**
- uint16\_t **kdfID**
- uint16\_t **aeadID**
- uint16\_t **kemID**
- const char \* **context**
- const char \* **skI**
- const char \* **pkI**
- const char \* **zz**
- const char \* **secret**
- const char \* **enc**
- const char \* **info**
- const char \* **pskID**
- const char \* **nonce**
- const char \* **key**
- const char \* **pkR**
- const char \* **pkE**
- const char \* **skR**
- const char \* **skE**
- const char \* **psk**
- int **nencs**
- [hpke\\_tv\\_encs\\_t](#) \* **encs**
- void \* **jobj**

*pointer to json-c object into which the char\* pointers above point*

### 3.6.1 Detailed Description

HKPE Test Vector structure using field names from published JSON file.

The jobj field (at the end) is the json-c object from which all these are derived and into which most of the char \* pointers point. When we make an array of [hpke\\_tv\\_s](#) then the same jobj will be pointed at by all, so when it's time to call hpke\_tv\_free then we'll just free one of those using the json-c API.

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

- [hpketv.h](#)





## Chapter 4

# File Documentation

### 4.1 hpke.c File Reference

An OpenSSL-based HPKE implementation following draft-irtf-cfrg-hpke.

```
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include <ctype.h>
#include <openssl/ssl.h>
#include <openssl/rand.h>
#include <openssl/kdf.h>
#include <openssl/evp.h>
#include <openssl/params.h>
#include "hpke.h"
```

Include dependency graph for hpke.c:



### Data Structures

- struct [hpke\\_aead\\_info\\_t](#)  
*info about an AEAD*
- struct [hpke\\_kem\\_info\\_t](#)  
*info about a KEM*
- struct [hpke\\_kdf\\_info\\_t](#)  
*info about a KDF*

### Macros

- #define [CHECK\\_HPKE\\_CTX](#) if ((cp->context)>\*contextlen) { erv=\_\_LINE\_\_; goto err; }  
*make it easier to do repetitive code*
- #define [ISAUTHMODE](#)(xxmode) (xxmode==[HPKE\\_MODE\\_AUTH](#) || xxmode==[HPKE\\_MODE\\_PSKAUTH](#))
- #define [ISPSKMODE](#)(xxmode) (xxmode==[HPKE\\_MODE\\_PSK](#) || xxmode==[HPKE\\_MODE\\_PSKAUTH](#))

## Functions

- int [hpke\\_ah\\_decode](#) (size\_t ahlen, const char \*ah, size\_t \*blen, unsigned char \*\*buf)  
*decode ascii hex to a binary buffer*
- static int [hpke\\_pbuf](#) (FILE \*fout, char \*msg, unsigned char \*buf, size\_t blen)  
*for odd/occasional debugging*
- static int [hpke\\_suite\\_check](#) (hpke\_suite\_t suite)  
*Check if ciphersuite is ok/known to us.*
- static size\_t [figure\\_contextlen](#) (hpke\_suite\_t suite)  
*return the length of the context for this suite*
- static int [hpke\\_aead\\_dec](#) (hpke\_suite\_t suite, unsigned char \*key, size\_t keylen, unsigned char \*iv, size\_t ivlen, unsigned char \*aad, size\_t aadlen, unsigned char \*cipher, size\_t cipherlen, unsigned char \*plain, size\_t \*plainlen)  
*do the AEAD decryption*
- static int [hpke\\_aead\\_enc](#) (hpke\_suite\_t suite, unsigned char \*key, size\_t keylen, unsigned char \*iv, size\_t ivlen, unsigned char \*aad, size\_t aadlen, unsigned char \*plain, size\_t plainlen, unsigned char \*cipher, size\_t \*cipherlen)  
*do the AEAD encryption as per the I-D*
- static int [hpke\\_extract](#) (hpke\_suite\_t suite, const unsigned char \*salt, const size\_t saltlen, const unsigned char \*zz, const size\_t zzlen, unsigned char \*\*secret, const size\_t secretlen)
- static int [hpke\\_expand](#) (hpke\_suite\_t suite, unsigned char \*secret, size\_t secretlen, char \*label, unsigned char \*context, size\_t contextlen, unsigned char \*\*out, size\_t outlen)
- static int [hpke\\_do\\_kem](#) (EVP\_PKEY \*key1, EVP\_PKEY \*key2, unsigned char \*\*zz, size\_t \*zzlen)  
*run the KEM with two keys*
- static int [hpke\\_make\\_context](#) (int mode, hpke\_suite\_t suite, const unsigned char \*enc, const size\_t encenclen, const unsigned char \*pub, const size\_t publen, const unsigned char \*pkl\_hash, const size\_t pkl\_hashlen, const char \*pskid, const unsigned char \*info, const size\_t infolen, unsigned char \*\*context, size\_t \*contextlen)  
*Create context for input to extract/expand.*
- static int [hpke\\_mode\\_check](#) (unsigned int mode)  
*check mode is in-range and supported*
- static int [hpke\\_psk\\_check](#) (unsigned int mode, char \*pskid, size\_t psklen, unsigned char \*psk)  
*check psk params are as per spec*
- static EVP\_PKEY \* [hpke\\_EVP\\_PKEY\\_new\\_raw\\_nist\\_public\\_key](#) (int curve, unsigned char \*buf, size\_t buflen)  
*hpke wrapper to import NIST curve public key as easily as x25519/x448*
- static EVP\_PKEY \* [hpke\\_EVP\\_PKEY\\_new\\_raw\\_nist\\_private\\_key](#) (int curve, unsigned char \*buf, size\_t buflen)  
*hpke wrapper to import NIST curve private key as easily as x25519/x448*
- int [hpke\\_enc](#) (unsigned int mode, hpke\_suite\_t suite, char \*pskid, size\_t psklen, unsigned char \*psk, size\_t publen, unsigned char \*pub, size\_t privlen, unsigned char \*priv, size\_t clearlen, unsigned char \*clear, size\_t aadlen, unsigned char \*aad, size\_t infolen, unsigned char \*info, size\_t senderpublen, unsigned char \*senderpub, size\_t \*cipherlen, unsigned char \*cipher)  
*HPKE single-shot encryption function.*
- int [hpke\\_dec](#) (unsigned int mode, hpke\_suite\_t suite, char \*pskid, size\_t psklen, unsigned char \*psk, size\_t publen, unsigned char \*pub, size\_t privlen, unsigned char \*priv, size\_t encenclen, unsigned char \*enc, size\_t cipherlen, unsigned char \*cipher, size\_t aadlen, unsigned char \*aad, size\_t infolen, unsigned char \*info, size\_t \*clearlen, unsigned char \*clear)  
*HPKE single-shot decryption function.*
- int [hpke\\_kg](#) (unsigned int mode, hpke\_suite\_t suite, size\_t \*publen, unsigned char \*pub, size\_t \*privlen, unsigned char \*priv)  
*generate a key pair*

## Variables

- `hpke_aead_info_t hpke_aead_tab []`  
*table of AEADs*
- `hpke_kem_info_t hpke_kem_tab []`  
*table of KEMs*
- `hpke_kdf_info_t hpke_kdf_tab []`  
*table of KDFs*
- static const unsigned char `zero_buf` [SHA512\_DIGEST\_LENGTH]  
*handy thing to have :-)*

### 4.1.1 Detailed Description

An OpenSSL-based HPKE implementation following draft-irtf-cfrg-hpke.

I plan to use this for my ESNI-enabled OpenSSL build (<https://github.com/sftcd/openssl>) when the time is right.

### 4.1.2 Function Documentation

#### 4.1.2.1 figure\_contextlen()

```
static size_t figure_contextlen (  
    hpke_suite_t suite ) [static]
```

return the length of the context for this suite

#### Parameters

<i>suite</i>	is the ciphersuite to use
--------------	---------------------------

#### Returns

the length (in octets) of the context

#### 4.1.2.2 hpke\_aead\_dec()

```
static int hpke_aead_dec (  
    hpke_suite_t suite,  
    unsigned char * key,  
    size_t keylen,  
    unsigned char * iv,  
    size_t ivlen,
```

```

    unsigned char * aad,
    size_t aadlen,
    unsigned char * cipher,
    size_t cipherlen,
    unsigned char * plain,
    size_t * plainlen ) [static]

```

do the AEAD decryption

#### Parameters

<i>suite</i>	is the ciphersuite
<i>key</i>	is the secret
<i>keylen</i>	is the length of the secret
<i>iv</i>	is the initialisation vector
<i>ivlen</i>	is the length of the iv
<i>aad</i>	is the additional authenticated data
<i>aadlen</i>	is the length of the aad
<i>cipher</i>	is obvious
<i>cipherlen</i>	is the ciphertext length
<i>plain</i>	is an output
<i>plainlen</i>	is an input/output, better be big enough on input, exact on output

#### Returns

1 for good otherwise bad

#### 4.1.2.3 hpke\_aead\_enc()

```

static int hpke_aead_enc (
    hpke_suite_t suite,
    unsigned char * key,
    size_t keylen,
    unsigned char * iv,
    size_t ivlen,
    unsigned char * aad,
    size_t aadlen,
    unsigned char * plain,
    size_t plainlen,
    unsigned char * cipher,
    size_t * cipherlen ) [static]

```

do the AEAD encryption as per the I-D

#### Parameters

<i>suite</i>	is the ciphersuite
<i>key</i>	is the secret
<i>keylen</i>	is the length of the secret
<i>iv</i>	is the initialisation vector

## Parameters

<i>ivlen</i>	is the length of the iv
<i>aad</i>	is the additional authenticated data
<i>aadlen</i>	is the length of the aad
<i>plain</i>	is an output
<i>plainlen</i>	is the length of plain
<i>cipher</i>	is an output
<i>cipherlen</i>	is an input/output, better be big enough on input, exact on output

## Returns

1 for good otherwise bad

## 4.1.2.4 hpke\_ah\_decode()

```
int hpke_ah_decode (
    size_t ahlen,
    const char * ah,
    size_t * blen,
    unsigned char ** buf )
```

decode ascii hex to a binary buffer

Bash command line hashing starting from ascii hex example:

```
$ echo -e "4f6465206f6e2061204772656369616e2055726e" | xxd -r -p | openssl sha256 (stdin)= 55c4040629c64c5efec2f7230407d6
```

The above generates the Hash(info) used in Appendix A.2

If you'd like to regenerate the zero\_sha256 value above, feel free `$ echo -n "" | openssl sha256 echo -n "" | openssl sha256 (stdin)= e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855` Or if you'd like to re-calculate the sha256 of nothing... `SHA256_CTX sha256; SHA256_Init(&sha256); char* buffer = NULL; int bytesRead = 0; SHA256_Update(&sha256, buffer, bytesRead); SHA256_Final(zero_sha256, &sha256);` ...but I've done it for you, so no need:-) `static const unsigned char zero_sha256[SHA256_DIGEST_LENGTH] = { 0xe3, 0xb0, 0xc4, 0x42, 0x98, 0xfc, 0x1c, 0x14, 0x9a, 0xfb, 0xf4, 0xc8, 0x99, 0x6f, 0xb9, 0x24, 0x27, 0xae, 0x41, 0xe4, 0x64, 0x9b, 0x93, 0x4c, 0xa4, 0x95, 0x99, 0x1b, 0x78, 0x52, 0xb8, 0x55};`

## Parameters

<i>ahlen</i>	is the ascii hex string length
<i>ah</i>	is the ascii hex string
<i>blen</i>	is a pointer to the returned binary length
<i>buf</i>	is a pointer to the internally allocated binary buffer

## Returns

1 for good otherwise bad

#### 4.1.2.5 hpke\_dec()

```
int hpke_dec (
    unsigned int mode,
    hpke_suite_t suite,
    char * pskid,
    size_t psklen,
    unsigned char * psk,
    size_t publen,
    unsigned char * pub,
    size_t privlen,
    unsigned char * priv,
    size_t enclen,
    unsigned char * enc,
    size_t cipherlen,
    unsigned char * cipher,
    size_t aadlen,
    unsigned char * aad,
    size_t infolen,
    unsigned char * info,
    size_t * clearlen,
    unsigned char * clear )
```

HPKE single-shot decryption function.

##### Parameters

<i>mode</i>	is the HPKE mode
<i>suite</i>	is the ciphersuite
<i>pskid</i>	is the pskid string for a PSK mode (can be NULL)
<i>psklen</i>	is the psk length
<i>psk</i>	is the psk
<i>publen</i>	is the length of the public (authentication) key
<i>pub</i>	is the encoded public (authentication) key
<i>privlen</i>	is the length of the private key
<i>priv</i>	is the encoded private key
<i>enclen</i>	is the length of the peer's public value
<i>enc</i>	is the peer's public value
<i>cipherlen</i>	is the length of the ciphertext
<i>cipher</i>	is the ciphertext
<i>aadlen</i>	is the length of the additional data
<i>aad</i>	is the encoded additional data
<i>infolen</i>	is the length of the info data (can be zero)
<i>info</i>	is the encoded info data (can be NULL)
<i>clearlen</i>	is the length of the input buffer for cleartext (octets used on output)
<i>clear</i>	is the encoded cleartext

##### Returns

1 for good (OpenSSL style), not-1 for error

#### 4.1.2.6 hpke\_do\_kem()

```
static int hpke_do_kem (
    EVP_PKEY * key1,
    EVP_PKEY * key2,
    unsigned char ** zz,
    size_t * zzlen ) [static]
```

run the KEM with two keys

##### Parameters

<i>key1</i>	is the first key, for which we have the private value
<i>key2</i>	is the peer's key
<i>zz</i>	is (a pointer to) the buffer for the result
<i>zzlen</i>	is the size of the buffer (octets-used on exit)

##### Returns

1 for good, not-1 for not good

#### 4.1.2.7 hpke\_enc()

```
int hpke_enc (
    unsigned int mode,
    hpke_suite_t suite,
    char * pskid,
    size_t psklen,
    unsigned char * psk,
    size_t publen,
    unsigned char * pub,
    size_t privlen,
    unsigned char * priv,
    size_t clearlen,
    unsigned char * clear,
    size_t aadlen,
    unsigned char * aad,
    size_t infolen,
    unsigned char * info,
    size_t * senderpublen,
    unsigned char * senderpub,
    size_t * cipherlen,
    unsigned char * cipher )
```

HPKE single-shot encryption function.

##### Parameters

<i>mode</i>	is the HPKE mode
<i>suite</i>	is the ciphersuite to use
<i>pskid</i>	is the pskid string fpr a PSK mode (can be NULL)

**Parameters**

<i>psklen</i>	is the psk length
<i>psk</i>	is the psk
<i>publen</i>	is the length of the recipient public key
<i>pub</i>	is the encoded recipient public key
<i>privlen</i>	is the length of the private (authentication) key
<i>priv</i>	is the encoded private (authentication) key
<i>clearlen</i>	is the length of the cleartext
<i>clear</i>	is the encoded cleartext
<i>aadlen</i>	is the lenght of the additional data (can be zero)
<i>aad</i>	is the encoded additional data (can be NULL)
<i>info</i>	is the lenght of the info data (can be zero)
<i>info</i>	is the encoded info data (can be NULL)
<i>senderpublen</i>	is the length of the input buffer for the sender's public key (length used on output)
<i>senderpub</i>	is the input buffer for ciphertext
<i>cipherlen</i>	is the length of the input buffer for ciphertext (length used on output)
<i>cipher</i>	is the input buffer for ciphertext

**Returns**

1 for good (OpenSSL style), not-1 for error

**4.1.2.8 hpke\_EVP\_PKEY\_new\_raw\_nist\_private\_key()**

```
static EVP_PKEY* hpke_EVP_PKEY_new_raw_nist_private_key (
    int curve,
    unsigned char * buf,
    size_t buflen ) [static]
```

hpke wrapper to import NIST curve private key as easily as x25519/x448

**Parameters**

<i>curve</i>	is the curve NID
<i>buf</i>	is the binary buffer with the private value
<i>buflen</i>	is the length of the private key buffer

**Returns**

a working EVP\_PKEY \* or NULL

Loads a malarkey required as it turns out You gotta: 1) name group 2) import private 3) then manually re-calc public key before 4) make an EVP\_PKEY



## 4.1.2.9 hpke\_EVP\_PKEY\_new\_raw\_nist\_public\_key()

```
static EVP_PKEY* hpke_EVP_PKEY_new_raw_nist_public_key (
    int curve,
    unsigned char * buf,
    size_t buflen ) [static]
```

hpke wrapper to import NIST curve public key as easily as x25519/x448

## Parameters

<i>curve</i>	is the curve NID
<i>buf</i>	is the binary buffer with the (uncompressed) public value
<i>buflen</i>	is the length of the private key buffer

## Returns

a working EVP\_PKEY \* or NULL

## 4.1.2.10 hpke\_expand()

```
static int hpke_expand (
    hpke_suite_t suite,
    unsigned char * secret,
    size_t secretlen,
    char * label,
    unsigned char * context,
    size_t contextlen,
    unsigned char ** out,
    size_t outlen ) [static]
```

brief RFC5869 HKDF-Expand

## Parameters

<i>suite</i>	is the ciphersuite
<i>secret</i>	- the initial key material (IKM)
<i>secretlen</i>	- length of above
<i>label</i>	- label to prepend to info
<i>context</i>	- the info
<i>contextlen</i>	- length of above
<i>out</i>	- the result of expansion (allocated inside)
<i>outlen</i>	- an input only!

## Returns

1 for good otherwise bad

#### 4.1.2.11 hpke\_extract()

```
static int hpke_extract (
    hpke_suite_t suite,
    const unsigned char * salt,
    const size_t saltlen,
    const unsigned char * zz,
    const size_t zzlen,
    unsigned char ** secret,
    const size_t secretlen ) [static]
```

brief RFC5869 HKDF-Extract

##### Parameters

<i>suite</i>	is the ciphersuite
<i>salt</i>	- surprisingly this is the salt;-)
<i>saltlen</i>	- length of above
<i>zz</i>	- the initial key material (IKM)
<i>zzlen</i>	- length of above
<i>secret</i>	- the result of extraction (allocated inside)
<i>secretlen</i>	- an input only!

##### Returns

1 for good otherwise bad

#### 4.1.2.12 hpke\_kg()

```
int hpke_kg (
    unsigned int mode,
    hpke_suite_t suite,
    size_t * publen,
    unsigned char * pub,
    size_t * privlen,
    unsigned char * priv )
```

generate a key pair

##### Parameters

<i>mode</i>	is the mode (currently unused)
<i>suite</i>	is the ciphersuite (currently unused)
<i>publen</i>	is the size of the public key buffer (exact length on output)
<i>pub</i>	is the public value
<i>privlen</i>	is the size of the private key buffer (exact length on output)
<i>priv</i>	is the private key

## Returns

1 for good (OpenSSL style), not-1 for error

## 4.1.2.13 hpke\_make\_context()

```
static int hpke_make_context (
    int mode,
    hpke_suite_t suite,
    const unsigned char * enc,
    const size_t enclen,
    const unsigned char * pub,
    const size_t publen,
    const unsigned char * pkI_hash,
    const size_t pkI_hashlen,
    const char * pskid,
    const unsigned char * info,
    const size_t infolen,
    unsigned char ** context,
    size_t * contextlen ) [static]
```

Create context for input to extract/expand.

## Parameters

<i>mode</i>	is the HPKE mode
<i>suite</i>	is the ciphersuite to use
<i>enc</i>	is the sender public key
<i>enclen</i>	is the length of the sender public key
<i>pub</i>	is the encoded recipient public key
<i>publen</i>	is the length of the recipient public key
<i>pkI_hash</i>	is the hash of the sender's authentication key (or NULL)
<i>pkI_hashlen</i>	is the length of the pkI_hash
<i>pskid</i>	is a PSK ID string (can be NULL)
<i>info</i>	is buffer of info to bind
<i>infolen</i>	is the length of the buffer of info
<i>context</i>	is a buffer for the resulting context
<i>contextlen</i>	is the size of the buffer and octets-used on exit

## Returns

1 for good, not 1 otherwise

## 4.1.2.14 hpke\_mode\_check()

```
static int hpke_mode_check (
    unsigned int mode ) [static]
```

check mode is in-range and supported

**Parameters**

<i>mode</i>	is the caller's chosen mode
-------------	-----------------------------

**Returns**

1 for good (OpenSSL style), not-1 for error

**4.1.2.15 hpke\_pbuf()**

```
static int hpke_pbuf (
    FILE * fout,
    char * msg,
    unsigned char * buf,
    size_t blen ) [static]
```

for odd/occasional debugging

**Parameters**

<i>fout</i>	is a FILE * to use
<i>msg</i>	is prepended to print
<i>buf</i>	is the buffer to print
<i>blen</i>	is the length of the buffer

**Returns**

1 for success

**4.1.2.16 hpke\_psk\_check()**

```
static int hpke_psk_check (
    unsigned int mode,
    char * pskid,
    size_t psklen,
    unsigned char * psk ) [static]
```

check psk params are as per spec

**Parameters**

<i>mode</i>	is the mode in use
<i>pskid</i>	PSK identifier
<i>psklen</i>	length of PSK
<i>psk</i>	the psk itself

**Returns**

1 for good (OpenSSL style), not-1 for error

If a PSK mode is used both \* pskid and psk must be non-default. Otherwise we ignore the PSK params.

**4.1.2.17 hpke\_suite\_check()**

```
static int hpke_suite_check (
    hpke_suite_t suite ) [static]
```

Check if ciphersuite is ok/known to us.

**Parameters**

<i>suite</i>	is the externally supplied cipheruite
--------------	---------------------------------------

**Returns**

1 for good, not-1 for error

For now, we only recognise HPKE\_SUITE\_DEFAULT

**4.1.3 Variable Documentation****4.1.3.1 hpke\_aead\_tab**

```
hpke_aead_info_t hpke_aead_tab[ ]
```

**Initial value:**

```
={
    { 0, NULL, 0, 0, 0 },
    { HPKE_AEAD_ID_AES_GCM_128, EVP_aes_128_gcm, 16, 16, 12 },
    { HPKE_AEAD_ID_AES_GCM_256, EVP_aes_256_gcm, 16, 32, 12 },
    { HPKE_AEAD_ID_CHACHA_POLY1305, EVP_chacha20_poly1305, 16, 32, 12 }
}
```

table of AEADs

#### 4.1.3.2 hpke\_kdf\_tab

```
hpke_kdf_info_t hpke_kdf_tab[]
```

##### Initial value:

```
= {
    { 0, NULL, 0 },
    { HPKE_KDF_ID_HKDF_SHA256, EVP_sha256, 32 },
    { HPKE_KDF_ID_HKDF_SHA512, EVP_sha512, 64 }
}
```

table of KDFs

#### 4.1.3.3 hpke\_kem\_tab

```
hpke_kem_info_t hpke_kem_tab[]
```

##### Initial value:

```
= {
    { 0, 0, 0, 0 },
    { HPKE_KEM_ID_P256, NID_X9_62_prime256v1, 65, 65, 32 },
    { HPKE_KEM_ID_25519, EVP_PKEY_X25519, 32, 32, 32 },
    { HPKE_KEM_ID_P521, NID_secp521r1, 133, 133, 64 },
    { HPKE_KEM_ID_448, EVP_PKEY_X448, 56, 56, 56 }
}
```

table of KEMs

#### 4.1.3.4 zero\_buf

```
const unsigned char zero_buf[SHA512_DIGEST_LENGTH] [static]
```

##### Initial value:

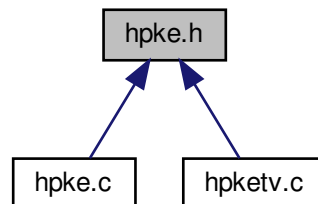
```
= {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
}
```

handy thing to have :-)

## 4.2 hpke.h File Reference

This has the data structures and prototypes (both internal and external) for an OpenSSL-based HPKE implementation following draft-irtf-cfrg-hpke.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [hpke\\_suite\\_t](#)  
*ciphersuite combination*

### Macros

- #define [HPKE\\_MAXSIZE](#) (640\*1024)  
*640k is more than enough for anyone (using this program:-)*
- #define [HPKE\\_MODE\\_BASE](#) 0  
*Base mode (all that we support for now)*
- #define [HPKE\\_MODE\\_PSK](#) 1  
*Pre-shared key mode.*
- #define [HPKE\\_MODE\\_AUTH](#) 2  
*Authenticated mode.*
- #define [HPKE\\_MODE\\_PSKAUTH](#) 3  
*PSK+authenticated mode.*
- #define [HPKE\\_KEM\\_ID\\_RESERVED](#) 0x0000  
*not used*
- #define [HPKE\\_KEM\\_ID\\_P256](#) 0x0001  
*NIST P-256.*
- #define [HPKE\\_KEM\\_ID\\_25519](#) 0x0002  
*Curve25519.*
- #define [HPKE\\_KEM\\_ID\\_P521](#) 0x0003  
*NIST P-521.*
- #define [HPKE\\_KEM\\_ID\\_448](#) 0x0004  
*Curve448.*
- #define [HPKE\\_KEM\\_ID\\_MAX](#) 0x0004  
*Curve448.*

- `#define HPKE_KDF_ID_RESERVED 0x0000`  
*not used*
- `#define HPKE_KDF_ID_HKDF_SHA256 0x0001`  
*HKDF-SHA256.*
- `#define HPKE_KDF_ID_HKDF_SHA512 0x0002`  
*HKDF-SHA512.*
- `#define HPKE_KDF_ID_MAX 0x0002`  
*HKDF-SHA512.*
- `#define HPKE_AEAD_ID_RESERVED 0x0000`  
*not used*
- `#define HPKE_AEAD_ID_AES_GCM_128 0x0001`  
*AES-GCM-128.*
- `#define HPKE_AEAD_ID_AES_GCM_256 0x0002`  
*AES-GCM-256.*
- `#define HPKE_AEAD_ID_CHACHA_POLY1305 0x0003`  
*Chacha20-Poly1305.*
- `#define HPKE_AEAD_ID_MAX 0x0003`  
*Chacha20-Poly1305.*
- `#define HPKE_SUITE_DEFAULT { HPKE_KEM_ID_25519, HPKE_KDF_ID_HKDF_SHA256, HPKE_AEAD_ID_AES_GCM_128 }`
- `#define HPKE_SUITE_TURNITUPTO11 { HPKE_KEM_ID_448, HPKE_KDF_ID_HKDF_SHA512, HPKE_AEAD_ID_CHACHA_POLY1305 }`
- `#define HPKE_A2B(__c__)`  
*Map ascii to binary - utility macro used in > 1 place.*

## Functions

- `int hpke_enc` (unsigned int mode, `hpke_suite_t` suite, char \*pskid, size\_t psklen, unsigned char \*psk, size\_t publen, unsigned char \*pub, size\_t privlen, unsigned char \*priv, size\_t clearlen, unsigned char \*clear, size\_t aadlen, unsigned char \*aad, size\_t infolen, unsigned char \*info, size\_t \*senderpublen, unsigned char \*senderpub, size\_t \*cipherlen, unsigned char \*cipher)  
*HPKE single-shot encryption function.*
- `int hpke_dec` (unsigned int mode, `hpke_suite_t` suite, char \*pskid, size\_t psklen, unsigned char \*psk, size\_t publen, unsigned char \*pub, size\_t privlen, unsigned char \*priv, size\_t encilen, unsigned char \*enc, size\_t cipherlen, unsigned char \*cipher, size\_t aadlen, unsigned char \*aad, size\_t infolen, unsigned char \*info, size\_t \*clearlen, unsigned char \*clear)  
*HPKE single-shot decryption function.*
- `int hpke_kg` (unsigned int mode, `hpke_suite_t` suite, size\_t \*publen, unsigned char \*pub, size\_t \*privlen, unsigned char \*priv)  
*generate a key pair*
- `int hpke_ah_decode` (size\_t ahlen, const char \*ah, size\_t \*blen, unsigned char \*\*buf)  
*decode ascii hex to a binary buffer*

### 4.2.1 Detailed Description

This has the data structures and prototypes (both internal and external) for an OpenSSL-based HPKE implementation following draft-irtf-cfrg-hpke.

I plan to use this for my ESNI-enabled OpenSSL build when the time is right, that's: <https://github.com/sftcd/openssl>)



## 4.2.2 Macro Definition Documentation

### 4.2.2.1 HPKE\_A2B

```
#define HPKE_A2B(
    __c__ )
```

**Value:**

```
(__c__>='0' && __c__<='9' ? (__c__-'0') : \
    (__c__>='A' && __c__<='F' ? (__c__-'A'+10) : \
    (__c__>='a' && __c__<='f' ? (__c__-'a'+10) : 0))
```

Map ascii to binary - utility macro used in >1 place.

### 4.2.2.2 HPKE\_SUITE\_DEFAULT

```
#define HPKE_SUITE_DEFAULT { HPKE_KEM_ID_25519, HPKE_KDF_ID_HKDF_SHA256, HPKE_AEAD_ID_AES_GCM_128 }
```

Two suite constants, use this like:

```
hpke_suite_t myvar = HPKE_SUITE_DEFAULT;
```

## 4.2.3 Function Documentation

### 4.2.3.1 hpke\_ah\_decode()

```
int hpke_ah_decode (
    size_t ahlen,
    const char * ah,
    size_t * blen,
    unsigned char ** buf )
```

decode ascii hex to a binary buffer

**Parameters**

<i>ahlen</i>	is the ascii hex string length
<i>ah</i>	is the ascii hex string
<i>blen</i>	is a pointer to the returned binary length
<i>buf</i>	is a pointer to the internally allocated binary buffer

**Returns**

1 for good (OpenSSL style), not-1 for error

Bash command line hashing starting from ascii hex example:

```
$ echo -e "4f6465206f6e2061204772656369616e2055726e" | xxd -r -p | openssl sha256 (stdin)= 55c4040629c64c5efec2f7230407d6
```

The above generates the Hash(info) used in Appendix A.2

If you'd like to regenerate the zero\_sha256 value above, feel free `$ echo -n "" | openssl sha256 echo -n "" | openssl sha256 (stdin)= e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855` Or if you'd like to re-calculate the sha256 of nothing... `SHA256_CTX sha256; SHA256_Init(&sha256); char* buffer = NULL; int bytesRead = 0; SHA256_Update(&sha256, buffer, bytesRead); SHA256_Final(zero_sha256, &sha256);` ...but I've done it for you, so no need:-) `static const unsigned char zero_sha256[SHA256_DIGEST_LENGTH] = { 0xe3, 0xb0, 0xc4, 0x42, 0x98, 0xfc, 0x1c, 0x14, 0x9a, 0xfb, 0xf4, 0xc8, 0x99, 0x6f, 0xb9, 0x24, 0x27, 0xae, 0x41, 0xe4, 0x64, 0x9b, 0x93, 0x4c, 0xa4, 0x95, 0x99, 0x1b, 0x78, 0x52, 0xb8, 0x55};`

**Parameters**

<i>ahlen</i>	is the ascii hex string length
<i>ah</i>	is the ascii hex string
<i>blen</i>	is a pointer to the returned binary length
<i>buf</i>	is a pointer to the internally allocated binary buffer

**Returns**

1 for good otherwise bad

**4.2.3.2 hpke\_dec()**

```
int hpke_dec (
    unsigned int mode,
    hpke_suite_t suite,
    char * pskid,
    size_t psklen,
    unsigned char * psk,
    size_t publen,
    unsigned char * pub,
    size_t privlen,
    unsigned char * priv,
    size_t enclen,
    unsigned char * enc,
    size_t cipherlen,
    unsigned char * cipher,
    size_t aadlen,
    unsigned char * aad,
    size_t infolen,
    unsigned char * info,
    size_t * clearlen,
    unsigned char * clear )
```

HPKE single-shot decryption function.

## Parameters

<i>mode</i>	is the HPKE mode
<i>suite</i>	is the ciphersuite
<i>pskid</i>	is the pskid string fpr a PSK mode (can be NULL)
<i>psklen</i>	is the psk length
<i>psk</i>	is the psk
<i>publen</i>	is the length of the public (authentication) key
<i>pub</i>	is the encoded public (authentication) key
<i>privlen</i>	is the length of the private key
<i>priv</i>	is the encoded private key
<i>enclen</i>	is the length of the peer's public value
<i>enc</i>	is the peer's public value
<i>cipherlen</i>	is the length of the ciphertext
<i>cipher</i>	is the ciphertext
<i>aadlen</i>	is the lenght of the additional data
<i>aad</i>	is the encoded additional data
<i>infolen</i>	is the lenght of the info data (can be zero)
<i>info</i>	is the encoded info data (can be NULL)
<i>clearlen</i>	is the length of the input buffer for cleartext (octets used on output)
<i>clear</i>	is the encoded cleartext

## Returns

1 for good (OpenSSL style), not-1 for error

## 4.2.3.3 hpke\_enc()

```
int hpke_enc (
    unsigned int mode,
    hpke_suite_t suite,
    char * pskid,
    size_t psklen,
    unsigned char * psk,
    size_t publen,
    unsigned char * pub,
    size_t privlen,
    unsigned char * priv,
    size_t clearlen,
    unsigned char * clear,
    size_t aadlen,
    unsigned char * aad,
    size_t infolen,
    unsigned char * info,
    size_t * senderpublen,
    unsigned char * senderpub,
    size_t * cipherlen,
    unsigned char * cipher )
```

HPKE single-shot encryption function.

## Parameters

<i>mode</i>	is the HPKE mode
<i>suite</i>	is the ciphersuite to use
<i>pskid</i>	is the pskid string fpr a PSK mode (can be NULL)
<i>psklen</i>	is the psk length
<i>psk</i>	is the psk
<i>publen</i>	is the length of the recipient public key
<i>pub</i>	is the encoded recipient public key
<i>privlen</i>	is the length of the private (authentication) key
<i>priv</i>	is the encoded private (authentication) key
<i>clearlen</i>	is the length of the cleartext
<i>clear</i>	is the encoded cleartext
<i>aadlen</i>	is the lenght of the additional data (can be zero)
<i>aad</i>	is the encoded additional data (can be NULL)
<i>info</i>	is the lenght of the info data (can be zero)
<i>info</i>	is the encoded info data (can be NULL)
<i>senderpublen</i>	is the length of the input buffer for the sender's public key (length used on output)
<i>senderpub</i>	is the input buffer for ciphertext
<i>cipherlen</i>	is the length of the input buffer for ciphertext (length used on output)
<i>cipher</i>	is the input buffer for ciphertext

## Returns

1 for good (OpenSSL style), not-1 for error

## 4.2.3.4 hpke\_kg()

```
int hpke_kg (
    unsigned int mode,
    hpke_suite_t suite,
    size_t * publen,
    unsigned char * pub,
    size_t * privlen,
    unsigned char * priv )
```

generate a key pair

## Parameters

<i>mode</i>	is the mode (currently unused)
<i>suite</i>	is the ciphersuite (currently unused)
<i>publen</i>	is the size of the public key buffer (exact length on output)
<i>pub</i>	is the public value
<i>privlen</i>	is the size of the private key buffer (exact length on output)
<i>priv</i>	is the private key

## Returns

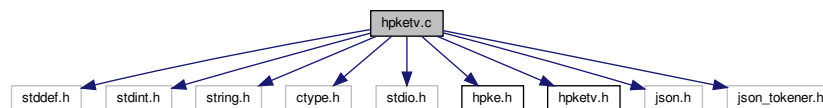
1 for good (OpenSSL style), not-1 for error

## 4.3 hpktv.c File Reference

Implementation related to test vectors for HPKE.

```
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include <ctype.h>
#include <stdio.h>
#include "hpke.h"
#include "hpktv.h"
#include <json.h>
#include <json_tokener.h>
```

Include dependency graph for hpktv.c:



## Macros

- `#define FAIL2BUILD(x) int x;`
- `#define grabnum(_xx) if (!strcmp(key,"#_xx")) { thearr[i]._xx=json_object_get_int(val); }`  
*copy typed/named field from json-c to hpke\_tv\_t*
- `#define grabstr(_xx) if (!strcmp(key,"#_xx")) { thearr[i]._xx=json_object_get_string(val); }`  
*copy typed/named field from json-c to hpke\_tv\_t*
- `#define grabestr(_xx) if (!strcmp(key1,"#_xx")) { encs[j]._xx=json_object_get_string(val1); }`  
*copy typed/named field from json-c to hpke\_tv\_t*
- `#define PRINTIT(_xx) printf("\t"#_xx": %s\n",a->_xx);`  
*print the name of a field and the value of that field*

## Functions

- static char \* **u2c\_transform** (const char \*uncomp)
- int **hpke\_tv\_load** (char \*fname, int \*nelems, **hpke\_tv\_t** \*\*array)  
*load test vectors from json file to array*
- void **hpke\_tv\_free** (int nelems, **hpke\_tv\_t** \*array)  
*free up test vector array*
- void **hpke\_tv\_print** (int nelems, **hpke\_tv\_t** \*array)  
*print test vectors*
- static int **hpke\_tv\_match** (unsigned int mode, **hpke\_suite\_t** suite, **hpke\_tv\_t** \*a)
- int **hpke\_tv\_pick** (unsigned int mode, **hpke\_suite\_t** suite, int nelems, **hpke\_tv\_t** \*arr, **hpke\_tv\_t** \*\*tv)  
*select a test vector to use based on mode and suite*

### 4.3.1 Detailed Description

Implementation related to test vectors for HPKE.

This is compiled in if TESTVECTORS is #define'd, otherwise not.

The overall plan with test vectors is to:

- define data structures here to store the test vectors
- have global variables with the actual data
- have a #ifdef'd command line argument to generate/check a test vector
- have #ifdef'd additional parameters to \_enc/\_dec functions for doing generation/checking

Source for test vectors is: <https://raw.githubusercontent.com/cfrg/draft-irtf-cfrg-hpke/master/test-vectors.json> A copy from 20191126 is also in this repo in test-vectors.json

### 4.3.2 Macro Definition Documentation

#### 4.3.2.1 FAIL2BUILD

```
#define FAIL2BUILD(  
    x ) int x;
```

Crap out if this isn't defined.

### 4.3.3 Function Documentation

#### 4.3.3.1 hpke\_tv\_free()

```
void hpke_tv_free (  
    int nelems,  
    hpke_tv_t * array )
```

free up test vector array

##### Parameters

<i>nelems</i>	is the number of array elements
<i>array</i>	is a guess what?

Caller doesn't need to free "parent" array

#### 4.3.3.2 hpke\_tv\_load()

```
int hpke_tv_load (
    char * fname,
    int * nelems,
    hpke_tv_t ** array )
```

load test vectors from json file to array

##### Parameters

<i>fname</i>	is the json file
<i>nelems</i>	returns with the number of array elements
<i>array</i>	returns with the elements

##### Returns

1 for good, other for bad

#### 4.3.3.3 hpke\_tv\_pick()

```
int hpke_tv_pick (
    unsigned int mode,
    hpke_suite_t suite,
    int nelems,
    hpke_tv_t * arr,
    hpke_tv_t ** tv )
```

select a test vector to use based on mode and suite

##### Parameters

<i>mode</i>	is the selected mode
<i>suite</i>	is the ciphersuite
<i>nelems</i>	is the number of array elements
<i>arr</i>	is the elements
<i>tv</i>	is the chosen test vector (doesn't need to be freed)

##### Returns

1 for good, other for bad

This function will randomly pick a matching test vector that matches the specified criteria.

The string to use is like "0,1,1,2" specifying the mode and suite in the (sorta:-) obvious manner. < array of pointers to matching vectors

#### 4.3.3.4 hpke\_tv\_print()

```
void hpke_tv_print (
    int nelems,
    hpke_tv_t * array )
```

print test vectors

##### Parameters

<i>nelems</i>	is the number of array elements
<i>array</i>	is the elements

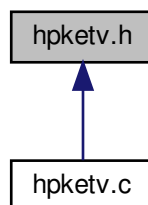
##### Returns

1 for good, other for bad

## 4.4 hpketv.h File Reference

Header file related to test vectors for HPKE.

This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [hpke\\_tv\\_encs\\_t](#)  
*Encryption(s) Test Vector structure using field names from published JSON file.*
- struct [hpke\\_tv\\_s](#)  
*HKPE Test Vector structure using field names from published JSON file.*

## Typedefs

- typedef struct [hpke\\_tv\\_s](#) [hpke\\_tv\\_t](#)  
*HKPE Test Vector structure using field names from published JSON file.*



## Functions

- int `hpke_tv_load` (char \*fname, int \*nelems, `hpke_tv_t` \*\*array)  
*load test vectors from json file to array*
- int `hpke_tv_pick` (unsigned int mode, `hpke_suite_t` suite, int nelems, `hpke_tv_t` \*arr, `hpke_tv_t` \*\*tv)  
*select a test vector to use based on mode and suite*
- void `hpke_tv_free` (int nelems, `hpke_tv_t` \*array)  
*free up test vector array*
- void `hpke_tv_print` (int nelems, `hpke_tv_t` \*array)  
*print test vectors*

### 4.4.1 Detailed Description

Header file related to test vectors for HPKE.

This is compiled in if TESTVECTORS is #define'd, otherwise not.

The overall plan with test vectors is to:

- define data structures here to store the test vectors
- have global variables with the actual data
- have a #ifdef'd command line argument to generate/check a test vector
- have #ifdef'd additional parameters to \_enc/\_dec functions for doing generation/checking

Source for test vectors is: [https://raw.githubusercontent.com/cfrg/draft-irtf-cfrg-hpke/master/test\\_vectors.json](https://raw.githubusercontent.com/cfrg/draft-irtf-cfrg-hpke/master/test_vectors.json) A copy from 20191126 is also in this repo in test-vectors.json

This should only be included if TESTVECTORS is #define'd.

### 4.4.2 Typedef Documentation

#### 4.4.2.1 hpke\_tv\_t

```
typedef struct hpke_tv_s hpke_tv_t
```

HKPE Test Vector structure using field names from published JSON file.

The jobj field (at the end) is the json-c object from which all these are derived and into which most of the char \* pointers point. When we make an array of `hpke_tv_s` then the same jobj will be pointed at by all, so when it's time to call `hpke_tv_free` then we'll just free one of those using the json-c API.

### 4.4.3 Function Documentation

#### 4.4.3.1 hpke\_tv\_free()

```
void hpke_tv_free (
    int nelems,
    hpke_tv_t * array )
```

free up test vector array

**Parameters**

<i>nelems</i>	is the number of array elements
<i>array</i>	is a guess what?

Caller doesn't need to free "parent" array

**4.4.3.2 hpke\_tv\_load()**

```
int hpke_tv_load (
    char * fname,
    int * nelems,
    hpke_tv_t ** array )
```

load test vectors from json file to array

**Parameters**

<i>fname</i>	is the json file
<i>nelems</i>	returns with the number of array elements
<i>array</i>	returns with the elements

**Returns**

1 for good, other for bad

**4.4.3.3 hpke\_tv\_pick()**

```
int hpke_tv_pick (
    unsigned int mode,
    hpke_suite_t suite,
    int nelems,
    hpke_tv_t * arr,
    hpke_tv_t ** tv )
```

select a test vector to use based on mode and suite

**Parameters**

<i>mode</i>	is the selected mode
<i>suite</i>	is the ciphersuite
<i>nelems</i>	is the number of array elements
<i>arr</i>	is the elements
<i>tv</i>	is the chosen test vector (doesn't need to be freed)

**Returns**

1 for good, other for bad

This function will randomly pick a matching test vector that matches the specified criteria.

The string to use is like "0,1,1,2" specifying the mode and suite in the (sorta:-) obvious manner. < array of pointers to matching vectors

**4.4.3.4 hpke\_tv\_print()**

```
void hpke_tv_print (
    int nelems,
    hpke_tv_t * array )
```

print test vectors

**Parameters**

<i>nelems</i>	is the number of array elements
<i>array</i>	is the elements

**Returns**

1 for good, other for bad



# Index

FAIL2BUILD  
  hpke\_tv.c, 32  
figure\_contextlen  
  hpke.c, 13  
  
HPKE\_A2B  
  hpke.h, 27  
HPKE\_SUITE\_DEFAULT  
  hpke.h, 27  
hpke.c, 11  
  figure\_contextlen, 13  
  hpke\_EVP\_PKEY\_new\_raw\_nist\_private\_key, 18  
  hpke\_EVP\_PKEY\_new\_raw\_nist\_public\_key, 18  
  hpke\_aead\_dec, 13  
  hpke\_aead\_enc, 14  
  hpke\_aead\_tab, 23  
  hpke\_ah\_decode, 15  
  hpke\_dec, 15  
  hpke\_do\_kem, 16  
  hpke\_enc, 17  
  hpke\_expand, 19  
  hpke\_extract, 19  
  hpke\_kdf\_tab, 23  
  hpke\_kem\_tab, 24  
  hpke\_kg, 20  
  hpke\_make\_context, 21  
  hpke\_mode\_check, 21  
  hpke\_pbuf, 22  
  hpke\_psk\_check, 22  
  hpke\_suite\_check, 23  
  zero\_buf, 24  
hpke.h, 25  
  HPKE\_A2B, 27  
  HPKE\_SUITE\_DEFAULT, 27  
  hpke\_ah\_decode, 27  
  hpke\_dec, 28  
  hpke\_enc, 29  
  hpke\_kg, 30  
hpke\_EVP\_PKEY\_new\_raw\_nist\_private\_key  
  hpke.c, 18  
hpke\_EVP\_PKEY\_new\_raw\_nist\_public\_key  
  hpke.c, 18  
hpke\_aead\_dec  
  hpke.c, 13  
hpke\_aead\_enc  
  hpke.c, 14  
hpke\_aead\_info\_t, 5  
hpke\_aead\_tab  
  hpke.c, 23  
  hpke.c, 15  
  hpke.h, 27  
hpke\_dec  
  hpke.c, 15  
  hpke.h, 28  
hpke\_do\_kem  
  hpke.c, 16  
hpke\_enc  
  hpke.c, 17  
  hpke.h, 29  
hpke\_expand  
  hpke.c, 19  
hpke\_extract  
  hpke.c, 19  
hpke\_kdf\_info\_t, 5  
hpke\_kdf\_tab  
  hpke.c, 23  
hpke\_kem\_info\_t, 6  
hpke\_kem\_tab  
  hpke.c, 24  
hpke\_kg  
  hpke.c, 20  
  hpke.h, 30  
hpke\_make\_context  
  hpke.c, 21  
hpke\_mode\_check  
  hpke.c, 21  
hpke\_pbuf  
  hpke.c, 22  
hpke\_psk\_check  
  hpke.c, 22  
hpke\_suite\_check  
  hpke.c, 23  
hpke\_suite\_t, 6  
hpke\_tv\_encs\_t, 7  
hpke\_tv\_free  
  hpke\_tv.c, 32  
  hpke\_tv.h, 35  
hpke\_tv\_load  
  hpke\_tv.c, 32  
  hpke\_tv.h, 36  
hpke\_tv\_pick  
  hpke\_tv.c, 33  
  hpke\_tv.h, 36  
hpke\_tv\_print  
  hpke\_tv.c, 33  
  hpke\_tv.h, 37  
hpke\_tv\_s, 8  
hpke\_tv\_t

- hpke\_tv.h, [35](#)
- hpke\_tv.c, [31](#)
  - FAIL2BUILD, [32](#)
  - hpke\_tv\_free, [32](#)
  - hpke\_tv\_load, [32](#)
  - hpke\_tv\_pick, [33](#)
  - hpke\_tv\_print, [33](#)
- hpke\_tv.h, [34](#)
  - hpke\_tv\_free, [35](#)
  - hpke\_tv\_load, [36](#)
  - hpke\_tv\_pick, [36](#)
  - hpke\_tv\_print, [37](#)
  - hpke\_tv\_t, [35](#)
- zero\_buf
  - hpke.c, [24](#)