

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

<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 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 . . . . .               | 7         |
| 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 . . . . .                      | 14        |
| 4.1.2 Function Documentation . . . . .                    | 14        |
| 4.1.2.1 hpke_aead_dec() . . . . .                         | 14        |
| 4.1.2.2 hpke_aead_enc() . . . . .                         | 15        |
| 4.1.2.3 hpke_ah_decode() . . . . .                        | 16        |
| 4.1.2.4 hpke_dec() . . . . .                              | 17        |
| 4.1.2.5 hpke_do_kem() . . . . .                           | 18        |
| 4.1.2.6 hpke_enc() . . . . .                              | 18        |
| 4.1.2.7 hpke_enc_evpc() . . . . .                         | 20        |
| 4.1.2.8 hpke_enc_int() . . . . .                          | 21        |
| 4.1.2.9 hpke_EVP_PKEY_new_raw_nist_public_key() . . . . . | 22        |
| 4.1.2.10 hpke_expand() . . . . .                          | 22        |
| 4.1.2.11 hpke_expansion() . . . . .                       | 23        |
| 4.1.2.12 hpke_extract() . . . . .                         | 24        |
| 4.1.2.13 hpke_extract_and_expand() . . . . .              | 25        |
| 4.1.2.14 hpke_good4grease() . . . . .                     | 25        |
| 4.1.2.15 hpke_kem_id_check() . . . . .                    | 26        |
| 4.1.2.16 hpke_kem_id_nist_curve() . . . . .               | 26        |
| 4.1.2.17 hpke_kg() . . . . .                              | 26        |
| 4.1.2.18 hpke_kg_evpc() . . . . .                         | 27        |
| 4.1.2.19 hpke_mode_check() . . . . .                      | 28        |

|                                      |    |
|--------------------------------------|----|
| 4.1.2.20 hpke_prbuf2evp()            | 28 |
| 4.1.2.21 hpke_psk_check()            | 28 |
| 4.1.2.22 hpke_random_suite()         | 29 |
| 4.1.2.23 hpke_setlibctx()            | 29 |
| 4.1.2.24 hpke_str2suite()            | 30 |
| 4.1.2.25 hpke_suite_check()          | 30 |
| 4.1.3 Variable Documentation         | 30 |
| 4.1.3.1 hpke_aead_tab                | 30 |
| 4.1.3.2 hpke_kdf_tab                 | 31 |
| 4.1.3.3 hpke_kem_tab                 | 31 |
| 4.2 hpke.h File Reference            | 31 |
| 4.2.1 Detailed Description           | 34 |
| 4.2.2 Macro Definition Documentation | 34 |
| 4.2.2.1 HPKE_A2B                     | 35 |
| 4.2.2.2 HPKE_DEFSIZE                 | 35 |
| 4.2.2.3 HPKE_SUITE_DEFAULT           | 35 |
| 4.2.3 Function Documentation         | 35 |
| 4.2.3.1 hpke_ah_decode()             | 35 |
| 4.2.3.2 hpke_dec()                   | 36 |
| 4.2.3.3 hpke_enc()                   | 37 |
| 4.2.3.4 hpke_enc_evp()               | 39 |
| 4.2.3.5 hpke_expansion()             | 40 |
| 4.2.3.6 hpke_good4grease()           | 40 |
| 4.2.3.7 hpke_kg()                    | 41 |
| 4.2.3.8 hpke_kg_evp()                | 42 |
| 4.2.3.9 hpke_prbuf2evp()             | 43 |
| 4.2.3.10 hpke_setlibctx()            | 44 |
| 4.2.3.11 hpke_str2suite()            | 44 |
| 4.2.3.12 hpke_suite_check()          | 44 |
| 4.3 hpketv.c File Reference          | 45 |
| 4.3.1 Detailed Description           | 46 |
| 4.3.2 Macro Definition Documentation | 46 |
| 4.3.2.1 FAIL2BUILD                   | 46 |
| 4.3.3 Function Documentation         | 46 |
| 4.3.3.1 hpke_tv_free()               | 46 |
| 4.3.3.2 hpke_tv_load()               | 47 |
| 4.3.3.3 hpke_tv_pick()               | 47 |
| 4.3.3.4 hpke_tv_print()              | 48 |
| 4.4 hpketv.h File Reference          | 48 |
| 4.4.1 Detailed Description           | 49 |
| 4.4.2 Typedef Documentation          | 49 |
| 4.4.2.1 hpke_tv_t                    | 49 |

---

|                              |           |
|------------------------------|-----------|
| 4.4.3 Function Documentation | 49        |
| 4.4.3.1 hpke_tv_free()       | 49        |
| 4.4.3.2 hpke_tv_load()       | 50        |
| 4.4.3.3 hpke_tv_pick()       | 50        |
| 4.4.3.4 hpke_tv_print()      | 51        |
| <b>Index</b>                 | <b>53</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 . . . . .  | 7 |
| <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 . . . . . | 31 |
| <a href="#">hpketv.c</a> | Implementation related to test vectors for HPKE . . . . .  | 45 |
| <a href="#">hpketv.h</a> | Header file related to test vectors for HPKE . . . . .   | 48 |



## 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*
- const char \* [keytype](#)  
*string form of alg type "EC"/"X25519"/"X448"*
- const char \* [groupname](#)  
*string form of EC group for NIST curves*
- int [groupid](#)  
*NID of KEM.*
- const EVP\_MD \*(\* [hash\\_init\\_func](#) )(void)  
*hash alg for the HKDF*
- size\_t [Nsecret](#)  
*size of secrets*
- 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](#)  
*AEAD alg 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 \* [nonce](#)  
*aascii-hex encoded nonce*
- 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:

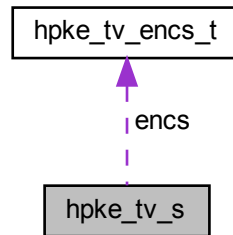
- [hpketv.h](#)

### 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 **kdf\_id**
- uint16\_t **aead\_id**
- uint16\_t **kem\_id**
- const char \* **info**
- const char \* **exporter\_secret**
- const char \* **enc**
- const char \* **key\_schedule\_context**
- const char \* **nonce**
- const char \* **secret**
- const char \* **shared\_secret**
- const char \* **skEm**
- const char \* **skRm**
- const char \* **skSm**
- const char \* **pkEm**
- const char \* **pkRm**
- const char \* **pkSm**
- const char \* **seedE**
- const char \* **seedR**
- const char \* **seedS**
- const char \* **psk\_id**
- 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

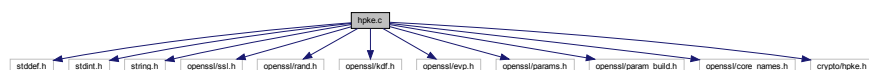
# File Documentation

### 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 <openssl/ssl.h>
#include <openssl/rand.h>
#include <openssl/kdf.h>
#include <openssl/evp.h>
#include <openssl/params.h>
#include <openssl/param_build.h>
#include <openssl/core_names.h>
#include <crypto/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 HPKE_VERLABEL "HPKE-v1"`  
*The version string label.*
- `#define HPKE_SEC41LABEL "KEM"`  
*The "suite\_id" label for 4.1.*
- `#define HPKE_SEC51LABEL "HPKE"`  
*The "suite\_id" label for 5.1.*
- `#define HPKE_EAE_PRK_LABEL "eae_prk"`  
*The label within ExtractAndExpand.*
- `#define HPKE_PSKIDHASH_LABEL "psk_id_hash"`  
*within key\_schedule\_context*
- `#define HPKE_INFOHASH_LABEL "info_hash"`  
*within key\_schedule\_context*
- `#define HPKE_SS_LABEL "shared_secret"`  
*Yet another label.*
- `#define HPKE_NONCE_LABEL "base_nonce"`  
*guess?*
- `#define HPKE_EXP_LABEL "exp"`  
*guess again?*
- `#define HPKE_KEY_LABEL "key"`  
*guess again?*
- `#define HPKE_PSK_HASH_LABEL "psk_hash"`  
*guess again?*
- `#define HPKE_SECRET_LABEL "secret"`  
*guess again?*
- `#define HPKE_5869_MODE_PURE 0`  
*Do "pure" RFC5869.*
- `#define HPKE_5869_MODE_KEM 1`  
*Abide by HPKE section 4.1.*
- `#define HPKE_5869_MODE_FULL 2`  
*Abide by HPKE section 5.1.*
- `#define PEM_PRIVATEHEADER "-----BEGIN PRIVATE KEY-----\n"`
- `#define PEM_PRIVATEFOOTER "\n-----END PRIVATE KEY-----\n"`
- `#define HPKE_MSMATCH(inp, known) (strlen(inp)==strlen(known) && !strcasecmp(inp,known))`

## 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_kem_id_check (uint16_t kem_id)`  
*Check if kem\_id is ok/known to us.*
- `static int hpke_kem_id_nist_curve (uint16_t kem_id)`  
*check if KEM uses NIST curve or not*
- `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 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` (const `hpke_suite_t` suite, const int mode5869, const unsigned char \*salt, const size\_t saltlen, const char \*label, const size\_t labellen, const unsigned char \*ikm, const size\_t ikmlen, unsigned char \*secret, size\_t \*secretlen)  
*RFC5869 HKDF-Extract.*
- static int `hpke_expand` (const `hpke_suite_t` suite, const int mode5869, const unsigned char \*prk, const size\_t prklen, const char \*label, const size\_t labellen, const unsigned char \*info, const size\_t infolen, const uint32\_t L, unsigned char \*out, size\_t \*outlen)  
*RFC5869 HKDF-Expand.*
- static int `hpke_extract_and_expand` (`hpke_suite_t` suite, int mode5869, unsigned char \*shared\_secret, size\_t shared\_secretlen, unsigned char \*context, size\_t contextlen, unsigned char \*secret, size\_t \*secretlen)  
*ExtractAndExpand.*
- static int `hpke_do_kem` (int encrypting, `hpke_suite_t` suite, EVP\_PKEY \*key1, size\_t key1enclen, unsigned char \*key1enc, EVP\_PKEY \*key2, size\_t key2enclen, unsigned char \*key2enc, EVP\_PKEY \*akey, size\_t apublen, unsigned char \*apub, unsigned char \*\*ss, size\_t \*sslen)  
*run the KEM with two keys as required*
- 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 int `hpke_enc_int` (unsigned int mode, `hpke_suite_t` suite, char \*pskid, size\_t psklen, unsigned char \*psk, size\_t publen, unsigned char \*pub, size\_t authprivlen, unsigned char \*authpriv, EVP\_PKEY \*authpriv\_ev, size\_t clearlen, unsigned char \*clear, size\_t aadlen, unsigned char \*aad, size\_t infolen, unsigned char \*info, size\_t seqlen, unsigned char \*seq, size\_t extsenderpublen, unsigned char \*extsenderpub, EVP\_PKEY \*extsenderpriv, size\_t rawsenderprivlen, unsigned char \*rawsenderpriv, size\_t \*senderpublen, unsigned char \*senderpub, size\_t \*cipherlen, unsigned char \*cipher)  
*Internal HPKE single-shot encryption function.*
- 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 authprivlen, unsigned char \*authpriv, EVP\_PKEY \*authpriv\_ev, size\_t clearlen, unsigned char \*clear, size\_t aadlen, unsigned char \*aad, size\_t infolen, unsigned char \*info, size\_t seqlen, unsigned char \*seq, size\_t \*senderpublen, unsigned char \*senderpub, size\_t \*cipherlen, unsigned char \*cipher)  
*HPKE single-shot encryption function.*
- int `hpke_enc_ev` (unsigned int mode, `hpke_suite_t` suite, char \*pskid, size\_t psklen, unsigned char \*psk, size\_t publen, unsigned char \*pub, size\_t authprivlen, unsigned char \*authpriv, EVP\_PKEY \*authpriv\_ev, size\_t clearlen, unsigned char \*clear, size\_t aadlen, unsigned char \*aad, size\_t infolen, unsigned char \*info, size\_t seqlen, unsigned char \*seq, size\_t extsenderpublen, unsigned char \*extsenderpub, EVP\_PKEY \*extsenderpriv, size\_t \*cipherlen, unsigned char \*cipher)  
*Internal 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 authpublen, unsigned char \*authpub, size\_t privlen, unsigned char \*priv, EVP\_PKEY \*evppriv, size\_t encenlen, unsigned char \*enc, size\_t cipherlen, unsigned char \*cipher, size\_t aadlen, unsigned char \*aad, size\_t infolen, unsigned char \*info, size\_t seqlen, unsigned char \*seq, 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_kg_ev` (unsigned int mode, `hpke_suite_t` suite, size\_t \*publen, unsigned char \*pub, EVP\_PKEY \*\*priv)  
*generate a key pair keeping private inside API*
- int `hpke_suite_check` (`hpke_suite_t` suite)

- check if a suite is supported locally*
- int `hpke_prbuf2evp` (unsigned int kem\_id, unsigned char \*prbuf, size\_t prbuf\_len, unsigned char \*pubuf, size\_t pubuf\_len, EVP\_PKEY \*\*retpriv)
  - : map a kem\_id and a private key buffer into an EVP\_PKEY
- static int `hpke_random_suite` (hpke\_suite\_t \*suite)
  - randomly pick a suite
- int `hpke_good4grease` (hpke\_suite\_t \*suite\_in, hpke\_suite\_t suite, unsigned char \*pub, size\_t \*pub\_len, unsigned char \*cipher, size\_t cipher\_len)
  - return a (possibly) random suite, public key, ciphertext for GREASErs
- int `hpke_str2suite` (char \*suitestr, hpke\_suite\_t \*suite)
  - map a string to a HPKE suite
- int `hpke_expansion` (hpke\_suite\_t suite, size\_t clearlen, size\_t \*cipherlen)
  - tell the caller how big the ciphertext will be
- int `hpke_setlibctx` (OSSL\_LIB\_CTX \*libctx)
  - set a non-default OSSL\_LIB\_CTX if needed

## Variables

- static `hpke_aead_info_t hpke_aead_tab` []
  - table of AEADs
- static `hpke_kem_info_t hpke_kem_tab` []
  - table of KEMs
- static `hpke_kdf_info_t hpke_kdf_tab` []
  - table of KDFs
- static OSSL\_LIB\_CTX \* `hpke_libctx` =NULL

## 4.1.1 Detailed Description

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

## 4.1.2 Function Documentation

### 4.1.2.1 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>  | input/output, better be big enough on input, exact on output |

## Returns

1 for good otherwise bad

## 4.1.2.2 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                                 |
| <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> | input/output, better be big enough on input, exact on output |

**Returns**

1 for good otherwise bad

**4.1.2.3 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

Since I always have to reconstruct this again in my head...  
Bash command line hashing starting from ascii hex example:

```
$ echo -e "4f6465206f6e2061204772656369616e2055726e" | \
  xxd -r -p | openssl sha256
(stdin)= 55c4040629c64c5efec2f7230407d612d16289d7c5d7afcf9340280abd2de1ab
```

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)= e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

Or if you'd like to re-caclulate 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

## 4.1.2.4 hpke\_dec()

```

int hpke_dec (
    unsigned int mode,
    hpke_suite_t suite,
    char * pskid,
    size_t psklen,
    unsigned char * psk,
    size_t authpublen,
    unsigned char * authpub,
    size_t privlen,
    unsigned char * priv,
    EVP_PKEY * evppriv,
    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 seqlen,
    unsigned char * seq,
    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>evppriv</i>   | is a pointer to an internal form of 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>seqlen</i>    | is the length of the info data (can be zero)     |
| <i>seq</i>       | is the encoded info data (can be NULL)           |
| <i>clearlen</i>  | length of the input buffer for cleartext         |
| <i>clear</i>     | is the encoded cleartext                         |

**Returns**

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

**4.1.2.5 hpke\_do\_kem()**

```
static int hpke_do_kem (
    int encrypting,
    hpke_suite_t suite,
    EVP_PKEY * key1,
    size_t key1enclen,
    unsigned char * key1enc,
    EVP_PKEY * key2,
    size_t key2enclen,
    unsigned char * key2enc,
    EVP_PKEY * akey,
    size_t apublen,
    unsigned char * apub,
    unsigned char ** ss,
    size_t * sslen ) [static]
```

run the KEM with two keys as required

**Parameters**

|                   |  |
|-------------------|--|
| <i>encrypting</i> | is 1 if we're encrypting, 0 for decrypting                 |
| <i>suite</i>      | is the ciphersuite   |
| <i>key1</i>       | is the first key, for which we have the private value      |
| <i>key1enclen</i> | is the length of the encoded form of key1                  |
| <i>key1enc</i>    | is the encoded form of key1                                |
| <i>key2</i>       | is the peer's key  |
| <i>key2enclen</i> | is the length of the encoded form of key1                  |
| <i>key2enc</i>    | is the encoded form of key1                                |
| <i>akey</i>       | is the authentication private key                          |
| <i>apublen</i>    | is the length of the encoded the authentication public key |
| <i>apub</i>       | is the encoded form of the authentication public key       |
| <i>ss</i>         | is (a pointer to) the buffer for the shared secret result  |
| <i>sslen</i>      | is the size of the buffer (octets-used on exit)            |

**Returns**

1 for good, not-1 for not good

**4.1.2.6 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 authprivlen,
unsigned char * authpriv,
EVP_PKEY * authpriv_evp,
size_t clearlen,
unsigned char * clear,
size_t aadlen,
unsigned char * aad,
size_t infolen,
unsigned char * info,
size_t seqlen,
unsigned char * seq,
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>authprivlen</i>  | is the length of the private (authentication) key      |
| <i>authpriv</i>     | is the encoded private (authentication) key            |
| <i>authpriv_evp</i> | is the EVP_PKEY* form of 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>infolen</i>      | is the lenght of the info data (can be zero)           |
| <i>info</i>         | is the encoded info data (can be NULL)                 |
| <i>seqlen</i>       | is the length of the info data (can be zero)           |
| <i>seq</i>          | is the encoded info data (can be NULL)                 |
| <i>senderpublen</i> | length of the input buffer for the sender's public key |
| <i>senderpub</i>    | is the input buffer for ciphertext                     |
| <i>cipherlen</i>    | length of the input buffer for ciphertext              |
| <i>cipher</i>       | is the input buffer for ciphertext                     |

#### Returns

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

#### 4.1.2.7 hpke\_enc\_evp()

```
int hpke_enc_evp (
    unsigned int mode,
    hpke_suite_t suite,
    char * pskid,
    size_t psklen,
    unsigned char * psk,
    size_t publen,
    unsigned char * pub,
    size_t authprivlen,
    unsigned char * authpriv,
    EVP_PKEY * authpriv_evp,
    size_t clearlen,
    unsigned char * clear,
    size_t aadlen,
    unsigned char * aad,
    size_t infolen,
    unsigned char * info,
    size_t seqlen,
    unsigned char * seq,
    size_t extsenderpublen,
    unsigned char * extsenderpub,
    EVP_PKEY * extsenderpriv,
    size_t * cipherlen,
    unsigned char * cipher )
```

Internal 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>authprivlen</i>  | is the length of the private (authentication) key       |
| <i>authpriv</i>     | is the encoded private (authentication) key             |
| <i>authpriv_evp</i> | is the EVP_PKEY* form of 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>infolen</i>      | is the lenght of the info data (can be zero)            |
| <i>info</i>         | is the encoded info data (can be NULL)                  |
| <i>seqlen</i>       | is the length of the info data (can be zero)            |
| <i>seq</i>          | is the encoded info data (can be NULL)                  |
| <i>senderpublen</i> | length of the input buffer with the sender's public key |
| <i>senderpub</i>    | is the input buffer for sender public key               |
| <i>senderpriv</i>   | has the handle for the sender private key               |
| <i>cipherlen</i>    | length of the input buffer for ciphertext               |
| <i>cipher</i>       | is the input buffer for ciphertext                      |

## Returns

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

## 4.1.2.8 hpke\_enc\_int()

```
static int hpke_enc_int (
    unsigned int mode,
    hpke_suite_t suite,
    char * pskid,
    size_t psklen,
    unsigned char * psk,
    size_t publen,
    unsigned char * pub,
    size_t authprivlen,
    unsigned char * authpriv,
    EVP_PKEY * authpriv_evpc,
    size_t clearlen,
    unsigned char * clear,
    size_t aadlen,
    unsigned char * aad,
    size_t infolen,
    unsigned char * info,
    size_t seqlen,
    unsigned char * seq,
    size_t extsenderpublen,
    unsigned char * extsenderpub,
    EVP_PKEY * extsenderpriv,
    size_t rawsenderprivlen,
    unsigned char * rawsenderpriv,
    size_t * senderpublen,
    unsigned char * senderpub,
    size_t * cipherlen,
    unsigned char * cipher ) [static]
```

Internal 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>authprivlen</i>   | is the length of the private (authentication) key     |
| <i>authpriv</i>      | is the encoded private (authentication) key           |
| <i>authpriv_evpc</i> | is the EVP_PKEY* form of 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)    |

## Parameters

|                        |  |
|------------------------|--|
| <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>seq</i>             | is the length of the info data (can be zero)       |
| <i>seq</i>             | is the encoded info data (can be NULL)             |
| <i>extsenderpublen</i> | length of the input buffer for sender's public key |
| <i>extsenderpub</i>    | is the input buffer for sender public key          |
| <i>extsenderpriv</i>   | has the handle for the sender private key          |
| <i>senderpublen</i>    | length of the input buffer for sender's public key |
| <i>senderpub</i>       | is the input buffer for ciphertext                 |
| <i>cipherlen</i>       | is the length of the input buffer for ciphertext   |
| <i>cipher</i>          | is the input buffer for ciphertext                 |

## Returns

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

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 (
    const hpke_suite_t suite,
    const int mode5869,
```

```

const unsigned char * prk,
const size_t prklen,
const char * label,
const size_t labellen,
const unsigned char * info,
const size_t infoflen,
const uint32_t L,
unsigned char * out,
size_t * outlen ) [static]

```

RFC5869 HKDF-Expand.

#### Parameters

|                   |   |
|-------------------|---|
| <i>suite</i>      | is the ciphersuite                              |
| <i>mode5869</i>   | - controls labelling specifics                  |
| <i>prk</i>        | - the initial pseudo-random key material        |
| <i>prk</i>        | - length of above                               |
| <i>label</i>      | - label to prepend to info                      |
| <i>labellen</i>   | - label to prepend to info                      |
| <i>context</i>    | - the info                                      |
| <i>contextlen</i> | - length of above                               |
| <i>L</i>          | - the length of the output desired              |
| <i>out</i>        | - the result of expansion (allocated by caller) |
| <i>outlen</i>     | - buf size on input                             |

#### Returns

1 for good otherwise bad

#### 4.1.2.11 hpke\_expansion()

```

int hpke_expansion (
    hpke_suite_t suite,
    size_t clearlen,
    size_t * cipherlen )

```

tell the caller how big the ciphertext will be

AEAD algorithms add a tag for data authentication. Those are almost always, but not always, 16 octets long, and who know what'll be true in the future. So this function allows a caller to find out how much data expansion they'll see with a given suite.

#### Parameters

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

**Returns**

1 for success, otherwise failure

**4.1.2.12 hpke\_extract()**

```
static int hpke_extract (
    const hpke_suite_t suite,
    const int mode5869,
    const unsigned char * salt,
    const size_t saltlen,
    const char * label,
    const size_t labellen,
    const unsigned char * ikm,
    const size_t ikmlen,
    unsigned char * secret,
    size_t * secretlen ) [static]
```

RFC5869 HKDF-Extract.

**Parameters**

|                  |   |
|------------------|---|
| <i>suite</i>     | is the ciphersuite                            |
| <i>mode5869</i>  | - controls labelling specifics                |
| <i>salt</i>      | - surprisingly this is the salt;-)            |
| <i>saltlen</i>   | - length of above                             |
| <i>label</i>     | - label for separation                        |
| <i>labellen</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> | - bufsize on input, used size on output       |

**Returns**

1 for good otherwise bad

Mode can be:

- HPKE\_5869\_MODE\_PURE meaning to ignore all the HPKE-specific labelling and produce an output that's RFC5869 compliant (useful for testing and maybe more)
- HPKE\_5869\_MODE\_KEM meaning to follow section 4.1 where the suite\_id is used as: concat("KEM", I2OSP(kem\_id, 2))
- HPKE\_5869\_MODE\_FULL meaning to follow section 5.1 where the suite\_id is used as: concat("HPKE", I2OSP(kem\_id, 2), I2OSP(kdf\_id, 2), I2OSP(aead\_id, 2))

Isn't that a bit of a mess!

## 4.1.2.13 hpke\_extract\_and\_expand()

```
static int hpke_extract_and_expand (
    hpke_suite_t suite,
    int mode5869,
    unsigned char * shared_secret,
    size_t shared_secretlen,
    unsigned char * context,
    size_t contextlen,
    unsigned char * secret,
    size_t * secretlen ) [static]
```

ExtractAndExpand.

## Parameters

|                         |                                |
|-------------------------|--------------------------------|
| <i>suite</i>            | is the ciphersuite             |
| <i>mode5869</i>         | - controls labelling specifics |
| <i>shared_secret</i>    | - the initial DH shared secret |
| <i>shared_secretlen</i> | - length of above              |
| <i>context</i>          | - the info                     |
| <i>contextlen</i>       | - length of above              |
| <i>secret</i>           | - the result of extract&expand |
| <i>secretlen</i>        | - buf size on input            |

## Returns

1 for good otherwise bad

## 4.1.2.14 hpke\_good4grease()

```
int hpke_good4grease (
    hpke_suite_t * suite_in,
    hpke_suite_t suite,
    unsigned char * pub,
    size_t * pub_len,
    unsigned char * cipher,
    size_t cipher_len )
```

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

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

As usual buffers are caller allocated and lengths on input are buffer size.

## Parameters

|                   |  |
|-------------------|--|
| <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 sender public value |
| <i>pub_len</i>    | is the length of pub (buffer size on input)                      |
| <i>cipher</i>     | buffer with random value of the appropriate length               |
| <i>cipher_len</i> | is the length of cipher  |

**Returns**

1 for success, otherwise failure

**4.1.2.15 hpke\_kem\_id\_check()**

```
static int hpke_kem_id_check (
    uint16_t kem_id ) [static]
```

Check if kem\_id is ok/known to us.

**Parameters**

|               |                                   |
|---------------|-----------------------------------|
| <i>kem_id</i> | is the externally supplied kem_id |
|---------------|-----------------------------------|

**Returns**

1 for good, not-1 for error

**4.1.2.16 hpke\_kem\_id\_nist\_curve()**

```
static int hpke_kem_id_nist_curve (
    uint16_t kem_id ) [static]
```

check if KEM uses NIST curve or not

**Parameters**

|               |                                   |
|---------------|-----------------------------------|
| <i>kem_id</i> | is the externally supplied kem_id |
|---------------|-----------------------------------|

**Returns**

1 for NIST, 0 otherwise, -1 for error

**4.1.2.17 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   |
| <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

#### 4.1.2.18 hpke\_kg\_evp()

```
int hpke_kg_evp (   
    unsigned int mode,  
    hpke_suite_t suite,  
    size_t * publen,  
    unsigned char * pub,  
    EVP_PKEY ** priv )
```

generate a key pair keeping private inside API

generate a key pair but keep private inside API

#### Parameters

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

#### Returns

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

#### 4.1.2.19 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

#### 4.1.2.20 hpke\_prbuf2evp()

```
int hpke_prbuf2evp (
    unsigned int kem_id,
    unsigned char * prbuf,
    size_t prbuf_len,
    unsigned char * pubuf,
    size_t pubuf_len,
    EVP_PKEY ** retpriv )
```

: map a kem\_id and a private key buffer into an EVP\_PKEY

Note that the buffer is expected to be some form of the PEM encoded private key, but could still have the PEM header or not, and might or might not be base64 encoded. We'll try handle all those options.

##### Parameters

|                  |   |
|------------------|---|
| <i>kem_id</i>    | is what'd you'd expect (using the HPKE registry values) |
| <i>prbuf</i>     | is the private key buffer                               |
| <i>prbuf_len</i> | is the length of that buffer                            |
| <i>pubuf</i>     | is the public key buffer (if available)                 |
| <i>pubuf_len</i> | is the length of that buffer                            |
| <i>priv</i>      | is a pointer to an EVP_PKEY * for the result            |

##### Returns

1 for success, otherwise failure

#### 4.1.2.21 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.

#### 4.1.2.22 hpke\_random\_suite()

```
static int hpke_random_suite (  
    hpke_suite_t * suite ) [static]
```

randomly pick a suite

#### Parameters

|              |               |
|--------------|---------------|
| <i>suite</i> | is the result |
|--------------|---------------|

#### Returns

1 for success, otherwise failure

If you change the structure of the various \*\_tab arrays then this code will also need change.

#### 4.1.2.23 hpke\_setlibctx()

```
int hpke_setlibctx (  
    OSSL_LIB_CTX * libctx )
```

set a non-default OSSL\_LIB\_CTX if needed

#### Parameters

|            |                       |
|------------|-----------------------|
| <i>ctx</i> | is the context to set |
|------------|-----------------------|

**Returns**

1 for success, otherwise failure

**4.1.2.24 hpke\_str2suite()**

```
int hpke_str2suite (
    char * suitestr,
    hpke_suite_t * 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

**4.1.2.25 hpke\_suite\_check()**

```
int hpke_suite_check (
    hpke_suite_t suite )
```

check if a suite is supported locally

**Parameters**

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

**Returns**

1 for good/supported, not-1 otherwise

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

```
hpke_aead_info_t hpke_aead_tab[] [static]
```

**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[] [static]
```

**Initial value:**

```
={
    { 0, NULL, 0 },
    { HPKE_KDF_ID_HKDF_SHA256, EVP_sha256, 32 },
    { HPKE_KDF_ID_HKDF_SHA384, EVP_sha384, 48 },
    { 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[] [static]
```

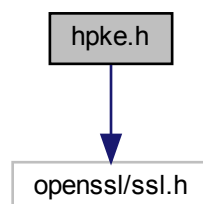
table of KEMs

Ok we're wasting space here, but not much and it's ok

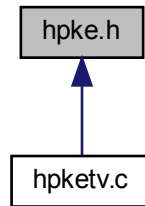
**4.2 hpke.h File Reference**

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

```
#include <openssl/ssl.h>
Include dependency graph for hpke.h:
```



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



## Data Structures

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

## Macros

- #define [HPKE\\_DEFSIZE](#) (40\*1024)  
*default plaintext/ciphertext buffer size e.g.*
- #define [HPKE\\_MAXSIZE](#) 2\*1024 /\* 2k is enough for anyone (using this program:-) \*/  
*biggest/default buffer for keys and internal buffers we use*
- #define [HPKE\\_MODE\\_BASE](#) 0  
*Base mode*
- #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](#) 0x0010  
*NIST P-256.*
- #define [HPKE\\_KEM\\_ID\\_P384](#) 0x0011  
*NIST P-256.*
- #define [HPKE\\_KEM\\_ID\\_P521](#) 0x0012  
*NIST P-521.*
- #define [HPKE\\_KEM\\_ID\\_25519](#) 0x0020  
*Curve25519.*
- #define [HPKE\\_KEM\\_ID\\_448](#) 0x0021  
*Curve448.*
- #define [HPKE\\_KDF\\_ID\\_RESERVED](#) 0x0000  
*not used*

- #define [HPKE\\_KDF\\_ID\\_HKDF\\_SHA256](#) 0x0001  
*HKDF-SHA256.*
- #define [HPKE\\_KDF\\_ID\\_HKDF\\_SHA384](#) 0x0002  
*HKDF-SHA512.*
- #define [HPKE\\_KDF\\_ID\\_HKDF\\_SHA512](#) 0x0003  
*HKDF-SHA512.*
- #define [HPKE\\_KDF\\_ID\\_MAX](#) 0x0003  
*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\\_MODESTR\\_BASE](#) "base"  
*base mode (1), no sender auth*
- #define [HPKE\\_MODESTR\\_PSK](#) "psk"  
*psk mode (2)*
- #define [HPKE\\_MODESTR\\_AUTH](#) "auth"  
*auth (3), with a sender-key pair*
- #define [HPKE\\_MODESTR\\_PSKAUTH](#) "pskauth"  
*psk+sender-key pair (4)*
- #define [HPKE\\_KEMSTR\\_P256](#) "p256"  
*KEM id 0x10.*
- #define [HPKE\\_KEMSTR\\_P384](#) "p384"  
*KEM id 0x11.*
- #define [HPKE\\_KEMSTR\\_P521](#) "p521"  
*KEM id 0x12.*
- #define [HPKE\\_KEMSTR\\_X25519](#) "x25519"  
*KEM id 0x20.*
- #define [HPKE\\_KEMSTR\\_X448](#) "x448"  
*KEM id 0x21.*
- #define [HPKE\\_KDFSTR\\_256](#) "hkdf-sha256"  
*KDF id 1.*
- #define [HPKE\\_KDFSTR\\_384](#) "hkdf-sha384"  
*KDF id 2.*
- #define [HPKE\\_KDFSTR\\_512](#) "hkdf-sha512"  
*KDF id 3.*
- #define [HPKE\\_AEADSTR\\_AES128GCM](#) "aes128gcm"  
*AEAD id 1.*
- #define [HPKE\\_AEADSTR\\_AES256GCM](#) "aes256gcm"  
*AEAD id 2.*
- #define [HPKE\\_AEADSTR\\_CP](#) "chachapoly1305"  
*AEAD id 3.*
- #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 authprivlen, unsigned char \*authpriv, EVP\_PKEY \*authpriv\_evp, size\_t clearlen, unsigned char \*clear, size\_t aadlen, unsigned char \*aad, size\_t inolen, unsigned char \*info, size\_t seqlen, unsigned char \*seq, size\_t \*senderpublen, unsigned char \*senderpub, size\_t \*cipherlen, unsigned char \*cipher)

*HPKE single-shot encryption function.*

- int [hpke\\_enc\\_evp](#) (unsigned int mode, [hpke\\_suite\\_t](#) suite, char \*pskid, size\_t psklen, unsigned char \*psk, size\_t publen, unsigned char \*pub, size\_t authprivlen, unsigned char \*authpriv, EVP\_PKEY \*authpriv\_evp, size\_t clearlen, unsigned char \*clear, size\_t aadlen, unsigned char \*aad, size\_t inolen, unsigned char \*info, size\_t seqlen, unsigned char \*seq, size\_t senderpublen, unsigned char \*senderpub, EVP\_PKEY \*senderpriv, size\_t \*cipherlen, unsigned char \*cipher)

*Internal 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, EVP\_PKEY \*evppriv, size\_t enclen, unsigned char \*enc, size\_t cipherlen, unsigned char \*cipher, size\_t aadlen, unsigned char \*aad, size\_t inolen, unsigned char \*info, size\_t seqlen, unsigned char \*seq, 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\\_kg\\_evp](#) (unsigned int mode, [hpke\\_suite\\_t](#) suite, size\_t \*publen, unsigned char \*pub, EVP\_PKEY \*\*priv)

*generate a key pair but keep private inside API*

- int [hpke\\_ah\\_decode](#) (size\_t ahlen, const char \*ah, size\_t \*blen, unsigned char \*\*buf)

*decode ascii hex to a binary buffer*

- int [hpke\\_suite\\_check](#) ([hpke\\_suite\\_t](#) suite)

*check if a suite is supported locally*

- int [hpke\\_prbuf2evp](#) (unsigned int kem\_id, unsigned char \*prbuf, size\_t prbuf\_len, unsigned char \*pubuf, size\_t pubuf\_len, EVP\_PKEY \*\*priv)

*: map a kem\_id and a private key buffer into an EVP\_PKEY*

- int [hpke\\_good4grease](#) ([hpke\\_suite\\_t](#) \*suite\_in, [hpke\\_suite\\_t](#) 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 [hpke\\_str2suite](#) (char \*str, [hpke\\_suite\\_t](#) \*suite)

*map a string to a HPKE suite*

- int [hpke\\_expansion](#) ([hpke\\_suite\\_t](#) suite, size\_t clearlen, size\_t \*cipherlen)

*tell the caller how big the ciphertext will be*

- int [hpke\\_setlibctx](#) (OSSL\_LIB\_CTX \*libctx)

*set a non-default OSSL\_LIB\_CTX if needed*

### 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.

### 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\_DEFSIZE

```
#define HPKE_DEFSIZE (40*1024)
```

default plaintext/ciphertext buffer size e.g.

if processing stdin

### 4.2.2.3 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

Since I always have to reconstruct this again in my head...  
Bash command line hashing starting from ascii hex example:

```
$ echo -e "4f6465206f6e2061204772656369616e2055726e" | \
  xxd -r -p | openssl sha256
(stdin)= 55c4040629c64c5efec2f7230407d612d16289d7c5d7afcf9340280abd2de1ab
```

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)= e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

Or if you'd like to re-caclulate 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, 0xc4, 0x29, 0xf6, 0xc1, 0xc1, 0x49, 0xaf, 0xbf, 0x4c, 0x89, 0x96, 0xfb, 0x92, 0x42, 0x7a, 0xe4, 0x1e, 0x46, 0x49, 0xb9, 0x34, 0xca, 0x49, 0x59, 0x91, 0xb7, 0x85, 0x2b, 0x85, 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 authpublen,
    unsigned char * authpub,
    size_t privlen,
```

```

    unsigned char * priv,
    EVP_PKEY * evppriv,
    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 seqlen,
    unsigned char * seq,
    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>evppriv</i>   | is a pointer to an internal form of 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>seqlen</i>    | is the length of the info data (can be zero)     |
| <i>seq</i>       | is the encoded info data (can be NULL)           |
| <i>clearlen</i>  | length of the input buffer for cleartext         |
| <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 authprivlen,
    unsigned char * authpriv,
    EVP_PKEY * authpriv_evp,
    size_t clearlen,
    unsigned char * clear,
    size_t aadlen,
    unsigned char * aad,
    size_t infolen,
    unsigned char * info,
    size_t seqlen,
    unsigned char * seq,
    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>authprivlen</i>  | is the length of the private (authentication) key      |
| <i>authpriv</i>     | is the encoded private (authentication) key            |
| <i>authpriv_evp</i> | is the EVP_PKEY* form of 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>infolen</i>      | is the lenght of the info data (can be zero)           |
| <i>info</i>         | is the encoded info data (can be NULL)                 |
| <i>seqlen</i>       | is the length of the info data (can be zero)           |
| <i>seq</i>          | is the encoded info data (can be NULL)                 |
| <i>senderpublen</i> | length of the input buffer for the sender's public key |
| <i>senderpub</i>    | is the input buffer for ciphertext                     |
| <i>cipherlen</i>    | length of the input buffer for ciphertext              |
| <i>cipher</i>       | is the input buffer for ciphertext                     |

#### Returns

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

## 4.2.3.4 hpke\_enc\_evp()

```

int hpke_enc_evp (
    unsigned int mode,
    hpke_suite_t suite,
    char * pskid,
    size_t psklen,
    unsigned char * psk,
    size_t publen,
    unsigned char * pub,
    size_t authprivlen,
    unsigned char * authpriv,
    EVP_PKEY * authpriv_evp,
    size_t clearlen,
    unsigned char * clear,
    size_t aadlen,
    unsigned char * aad,
    size_t infolen,
    unsigned char * info,
    size_t seqlen,
    unsigned char * seq,
    size_t extsenderpublen,
    unsigned char * extsenderpub,
    EVP_PKEY * extsenderpriv,
    size_t * cipherlen,
    unsigned char * cipher )

```

Internal 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>authprivlen</i>  | is the length of the private (authentication) key       |
| <i>authpriv</i>     | is the encoded private (authentication) key             |
| <i>authpriv_evp</i> | is the EVP_PKEY* form of 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>infolen</i>      | is the lenght of the info data (can be zero)            |
| <i>info</i>         | is the encoded info data (can be NULL)                  |
| <i>seqlen</i>       | is the length of the info data (can be zero)            |
| <i>seq</i>          | is the encoded info data (can be NULL)                  |
| <i>senderpublen</i> | length of the input buffer with the sender's public key |
| <i>senderpub</i>    | is the input buffer for sender public key               |
| <i>senderpriv</i>   | has the handle for the sender private key               |
| <i>cipherlen</i>    | length of the input buffer for ciphertext               |
| <i>cipher</i>       | is the input buffer for ciphertext                      |

**Returns**

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

**4.2.3.5 hpke\_expansion()**

```
int hpke_expansion (
    hpke_suite_t suite,
    size_t clearlen,
    size_t * cipherlen )
```

tell the caller how big the ciphertext will be

AEAD algorithms add a tag for data authentication. Those are almost always, but not always, 16 octets long, and who know what'll be true in the future. So this function allows a caller to find out how much data expansion they'll see with a given suite.

**Parameters**

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

**Returns**

1 for success, otherwise failure

**4.2.3.6 hpke\_good4grease()**

```
int hpke_good4grease (
    hpke_suite_t * suite_in,
    hpke_suite_t 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

As usual buffers are caller allocated and lengths on input are buffer size.

**Parameters**

|                   |  |
|-------------------|--|
| <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

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

As usual buffers are caller allocated and lengths on input are buffer size.

**Parameters**

|                   |  |
|-------------------|--|
| <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 sender public value |
| <i>pub_len</i>    | is the length of pub (buffer size on input)                      |
| <i>cipher</i>     | buffer with random value of the appropriate length               |
| <i>cipher_len</i> | is the length of cipher  |

**Returns**

1 for success, otherwise failure

**4.2.3.7 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

**Parameters**

|                |  |
|----------------|--|
| <i>mode</i>    | is the mode (currently unused)                                 |
| <i>suite</i>   | is the ciphersuite   |
| <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

**4.2.3.8 hpke\_kg\_evp()**

```
int hpke_kg_evp (
    unsigned int mode,
    hpke_suite_t suite,
    size_t * publen,
    unsigned char * pub,
    EVP_PKEY ** priv )
```

generate a key pair but keep private inside API

**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>priv</i>   | is the private key handle                                     |

**Returns**

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

generate a key pair but keep private inside API

**Parameters**

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



**Returns**

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

**4.2.3.9 hpke\_prbuf2evp()**

```
int hpke_prbuf2evp (
    unsigned int kem_id,
    unsigned char * prbuf,
    size_t prbuf_len,
    unsigned char * pubuf,
    size_t pubuf_len,
    EVP_PKEY ** retpriv )
```

: map a kem\_id and a private key buffer into an EVP\_PKEY

**Parameters**

|                  |   |
|------------------|---|
| <i>kem_id</i>    | is what'd you'd expect (using the HPKE registry values) |
| <i>prbuf</i>     | is the private key buffer                               |
| <i>prbuf_len</i> | is the length of that buffer                            |
| <i>pubuf</i>     | is the public key buffer (if available)                 |
| <i>pubuf_len</i> | is the length of that buffer                            |
| <i>priv</i>      | is a pointer to an EVP_PKEY * for the result            |

**Returns**

1 for success, otherwise failure

Note that the buffer is expected to be some form of the PEM encoded private key, but could still have the PEM header or not, and might or might not be base64 encoded. We'll try handle all those options.

Note that the buffer is expected to be some form of the PEM encoded private key, but could still have the PEM header or not, and might or might not be base64 encoded. We'll try handle all those options.

**Parameters**

|                  |   |
|------------------|---|
| <i>kem_id</i>    | is what'd you'd expect (using the HPKE registry values) |
| <i>prbuf</i>     | is the private key buffer                               |
| <i>prbuf_len</i> | is the length of that buffer                            |
| <i>pubuf</i>     | is the public key buffer (if available)                 |
| <i>pubuf_len</i> | is the length of that buffer                            |
| <i>priv</i>      | is a pointer to an EVP_PKEY * for the result            |

**Returns**

1 for success, otherwise failure

#### 4.2.3.10 hpke\_setlibctx()

```
int hpke_setlibctx (
    OSSL_LIB_CTX * libctx )
```

set a non-default OSSL\_LIB\_CTX if needed

##### Parameters

|            |                       |
|------------|-----------------------|
| <i>ctx</i> | is the context to set |
|------------|-----------------------|

##### Returns

1 for success, otherwise failure

#### 4.2.3.11 hpke\_str2suite()

```
int hpke_str2suite (
    char * suitestr,
    hpke_suite_t * 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

#### 4.2.3.12 hpke\_suite\_check()

```
int hpke_suite_check (
    hpke_suite_t suite )
```

check if a suite is supported locally

##### Parameters

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

## Returns

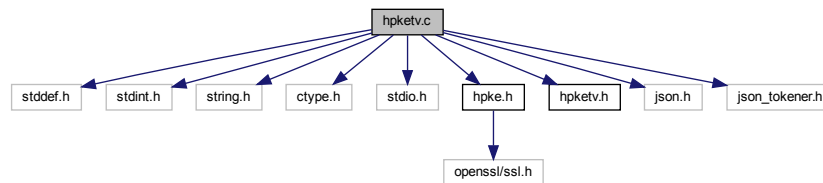
1 for good/supported, not-1 otherwise

## 4.3 hpketv.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 "hpketv.h"
#include <json.h>
#include <json_tokenizer.h>
```

Include dependency graph for hpketv.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> The latest copy from that repo 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                 |

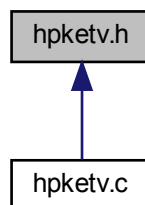
##### Returns

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) The latest copy from that repo 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, 46

## hpke.c, 11

hpke\_aead\_dec, 14  
hpke\_aead\_enc, 15  
hpke\_aead\_tab, 30  
hpke\_ah\_decode, 16  
hpke\_dec, 16  
hpke\_do\_kem, 18  
hpke\_enc, 18  
hpke\_enc\_ev, 19  
hpke\_enc\_int, 21  
hpke\_EVP\_PKEY\_new\_raw\_nist\_public\_key, 22  
hpke\_expand, 22  
hpke\_expansion, 23  
hpke\_extract, 24  
hpke\_extract\_and\_expand, 24  
hpke\_good4grease, 25  
hpke\_kdf\_tab, 31  
hpke\_kem\_id\_check, 26  
hpke\_kem\_id\_nist\_curve, 26  
hpke\_kem\_tab, 31  
hpke\_kg, 26  
hpke\_kg\_ev, 27  
hpke\_mode\_check, 27  
hpke\_prbuf2ev, 28  
hpke\_psk\_check, 28  
hpke\_random\_suite, 29  
hpke\_setlibctx, 29  
hpke\_str2suite, 30  
hpke\_suite\_check, 30

## hpke.h, 31

HPKE\_A2B, 34  
hpke\_ah\_decode, 35  
hpke\_dec, 36  
HPKE\_DEFSIZE, 35  
hpke\_enc, 37  
hpke\_enc\_ev, 38  
hpke\_expansion, 40  
hpke\_good4grease, 40  
hpke\_kg, 41  
hpke\_kg\_ev, 42  
hpke\_prbuf2ev, 43  
hpke\_setlibctx, 43  
hpke\_str2suite, 44  
hpke\_suite\_check, 44  
HPKE\_SUITE\_DEFAULT, 35

## HPKE\_A2B

hpke.h, 34

## hpke\_aead\_dec

hpke.c, 14

## hpke\_aead\_enc

hpke.c, 15

## hpke\_aead\_info\_t, 5

## hpke\_aead\_tab

hpke.c, 30

## hpke\_ah\_decode

hpke.c, 16

hpke.h, 35

## hpke\_dec

hpke.c, 16

hpke.h, 36

## HPKE\_DEFSIZE

hpke.h, 35

## hpke\_do\_kem

hpke.c, 18

## hpke\_enc

hpke.c, 18

hpke.h, 37

## hpke\_enc\_ev

hpke.c, 19

hpke.h, 38

## hpke\_enc\_int

hpke.c, 21

## hpke\_EVP\_PKEY\_new\_raw\_nist\_public\_key

hpke.c, 22

## hpke\_expand

hpke.c, 22

## hpke\_expansion

hpke.c, 23

hpke.h, 40

## hpke\_extract

hpke.c, 24

## hpke\_extract\_and\_expand

hpke.c, 24

## hpke\_good4grease

hpke.c, 25

hpke.h, 40

## hpke\_kdf\_info\_t, 5

## hpke\_kdf\_tab

hpke.c, 31

## hpke\_kem\_id\_check

hpke.c, 26

## hpke\_kem\_id\_nist\_curve

hpke.c, 26

## hpke\_kem\_info\_t, 6

## hpke\_kem\_tab

hpke.c, 31

- hpke\_kg
  - hpke.c, [26](#)
  - hpke.h, [41](#)
- hpke\_kg\_evp
  - hpke.c, [27](#)
  - hpke.h, [42](#)
- hpke\_mode\_check
  - hpke.c, [27](#)
- hpke\_prbuf2evp
  - hpke.c, [28](#)
  - hpke.h, [43](#)
- hpke\_psk\_check
  - hpke.c, [28](#)
- hpke\_random\_suite
  - hpke.c, [29](#)
- hpke\_setlibctx
  - hpke.c, [29](#)
  - hpke.h, [43](#)
- hpke\_str2suite
  - hpke.c, [30](#)
  - hpke.h, [44](#)
- hpke\_suite\_check
  - hpke.c, [30](#)
  - hpke.h, [44](#)
- HPKE\_SUITE\_DEFAULT
  - hpke.h, [35](#)
- hpke\_suite\_t, [7](#)
- hpke\_tv\_encs\_t, [7](#)
- hpke\_tv\_free
  - hpketv.c, [46](#)
  - hpketv.h, [49](#)
- hpke\_tv\_load
  - hpketv.c, [46](#)
  - hpketv.h, [50](#)
- hpke\_tv\_pick
  - hpketv.c, [47](#)
  - hpketv.h, [50](#)
- hpke\_tv\_print
  - hpketv.c, [47](#)
  - hpketv.h, [51](#)
- hpke\_tv\_s, [8](#)
- hpke\_tv\_t
  - hpketv.h, [49](#)
- hpketv.c, [45](#)
  - FAIL2BUILD, [46](#)
  - hpke\_tv\_free, [46](#)
  - hpke\_tv\_load, [46](#)
  - hpke\_tv\_pick, [47](#)
  - hpke\_tv\_print, [47](#)
- hpketv.h, [48](#)
  - hpke\_tv\_free, [49](#)
  - hpke\_tv\_load, [50](#)
  - hpke\_tv\_pick, [50](#)
  - hpke\_tv\_print, [51](#)
  - hpke\_tv\_t, [49](#)