

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

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

Generated by Doxygen 1.8.18

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
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
4 File Documentation	11
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()	15
4.1.2.4 hpke_dec()	16
4.1.2.5 hpke_do_kem()	17
4.1.2.6 hpke_enc()	18
4.1.2.7 hpke_enc_evp()	20
4.1.2.8 hpke_enc_int()	21
4.1.2.9 hpke_enc_raw()	22
4.1.2.10 hpke_EVP_PKEY_new_raw_nist_public_key()	23
4.1.2.11 hpke_expand()	24
4.1.2.12 hpke_extract()	25
4.1.2.13 hpke_extract_and_expand()	25
4.1.2.14 hpke_kem_id_check()	26
4.1.2.15 hpke_kem_id_nist_curve()	26
4.1.2.16 hpke_kg()	27
4.1.2.17 hpke_kg_evp()	27
4.1.2.18 hpke_mode_check()	28
4.1.2.19 hpke_prbuf2evp()	28

4.1.2.20 hpke_psk_check()	29
4.1.2.21 hpke_suite_check()	29
4.1.3 Variable Documentation	29
4.1.3.1 hpke_aead_strtab	30
4.1.3.2 hpke_aead_tab	30
4.1.3.3 hpke_kdf_strtab	30
4.1.3.4 hpke_kdf_tab	30
4.1.3.5 hpke_kem_strtab	31
4.1.3.6 hpke_kem_tab	31
4.1.3.7 hpke_mode_strtab	31
4.2 hpke.h File Reference	31
4.2.1 Detailed Description	34
4.2.2 Macro Definition Documentation	35
4.2.2.1 HPKE_A2B	35
4.2.2.2 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()	38
4.2.3.5 hpke_enc_raw()	40
4.2.3.6 hpke_expand()	41
4.2.3.7 hpke_extract()	41
4.2.3.8 hpke_kg()	42
4.2.3.9 hpke_kg_evp()	43
4.2.3.10 hpke_prbuf2evp()	44
4.2.3.11 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

Index	53
--------------	-----------

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

hpke_aead_info_t	Info about an AEAD	5
hpke_kdf_info_t	Info about a KDF	5
hpke_kem_info_t	Info about a KEM	6
hpke_suite_t	Ciphersuite combination	7
hpke_tv_encs_t	Encryption(s) Test Vector structure using field names from published JSON file	7
hpke_tv_s	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:

hpke.c	11
hpke.h	31
hpketv.c	45
hpketv.h	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 algorithm type "EC"/"X25519"/"X448"
- const char * [groupname](#)
string form of EC group needed for NIST curves "P-256"/"p-384"/"p-521"
- int [groupid](#)
NID of KEM.
- const EVP_MD *(* [hash_init_func](#))(void)
the hash alg we're using 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](#)
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 * [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:

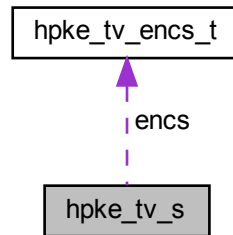
- [hpketv.h](#)

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

```
#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-07"
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"
A label within key_schedule_context.
- #define `HPKE_INFOHASH_LABEL` "info_hash"
A label 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 `PEM_PRIVATEHEADER` "-----BEGIN PRIVATE KEY-----\n"
- #define `PEM_PRIVATEFOOTER` "\n-----END PRIVATE KEY-----\n"

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
- 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)

- 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)
- 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 per draft-05
- 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 privlen, unsigned char *priv, size_t clearlen, unsigned char *clear, size_t aadlen, unsigned char *aad, size_t infolen, unsigned char *info, 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 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_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 privlen, unsigned char *priv, size_t clearlen, unsigned char *clear, size_t aadlen, unsigned char *aad, size_t infolen, unsigned char *info, size_t extsenderpublen, unsigned char *extsenderpub, EVP_PKEY *extsenderpriv, size_t *cipherlen, unsigned char *cipher)

Internal HPKE single-shot encryption function.
- int [hpke_enc_raw](#) (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 extsenderpublen, unsigned char *extsenderpub, size_t rawsenderprivlen, unsigned char *rawsenderpriv, 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 encenlen, 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_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_prbuf2ev](#) (unsigned int kem_id, unsigned char *prbuf, size_t prbuf_len, unsigned char *pubuf, size_t pubuf_len, EVP_PKEY **retpriv)

Variables

- const char * **hpke_mode_strtab** []
- [hpke_aead_info_t](#) **hpke_aead_tab** []
table of AEADs
- const char * **hpke_aead_strtab** []
- [hpke_kem_info_t](#) **hpke_kem_tab** []
table of KEMs
- const char * **hpke_kem_strtab** []
- [hpke_kdf_info_t](#) **hpke_kdf_tab** []
table of KDFs
- const char * **hpke_kdf_strtab** []

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 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

4.1.2.2 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>	is an 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)= e3b0c44298fclcl149afb4c8996fb92427ae41e4649b934ca495991b7852b855
```

Or if you'd like to re-caculate 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, 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 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 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>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>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

4.1.2.5 hpke_do_kem()

```

static int hpke_do_kem (
    int encrypting,
    hpke_suite_t suite,
    EVP_PKEY * key1,
    size_t keylenclen,
    unsigned char * keylenc,
    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 per draft-05

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>key1en</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>key2en</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 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

4.1.2.7 hpke_enc_evpc()

```
int hpke_enc_evpc (
    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 info,
    unsigned char * info,
    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>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 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>	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_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 privlen,
    unsigned char * priv,
    size_t clearlen,
    unsigned char * clear,
    size_t aadlen,
    unsigned char * aad,
    size_t info,
    unsigned char * info,
    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>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>infolen</i>	is the lenght of the info data (can be zero)
<i>info</i>	is the encoded info data (can be NULL)
<i>extsenderpublen</i>	is the length of the input buffer with the 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>	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

4.1.2.9 hpke_enc_raw()

```

int hpke_enc_raw (
    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 info,
size_t extsenderpublen,
unsigned char * extsenderpub,
size_t rawsenderprivlen,
unsigned char * rawsenderpriv,
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 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 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 length of the additional data (can be zero)
<i>aad</i>	is the encoded additional data (can be NULL)
<i>info</i>	is the length 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 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>	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

4.1.2.10 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`

4.1.2.11 hpke_expand()

```
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 )
```

brief 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

4.1.2.12 hpke_extract()

```
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 )
```

brief 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_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_↔</i> <i>_id</i>	is the externally supplied kem_id
--------------------------------------	-----------------------------------

Returns

1 for good, not-1 for error

4.1.2.15 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</i> ↔ <i>_id</i>	is the externally supplied kem_id
----------------------------	-----------------------------------

Returns

1 for NIST, 0 otherwise, -1 for error

4.1.2.16 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

4.1.2.17 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

4.1.2.18 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

4.1.2.19 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 )
```

brief: 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.

4.1.2.20 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.

4.1.2.21 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_strtab

```
const char* hpke_aead_strtab[]
```

Initial value:

```
={  
    NULL,  
    HPKE_AEADSTR_AES128GCM,  
    HPKE_AEADSTR_AES256GCM,  
    HPKE_AEADSTR_CP}  
}
```

4.1.3.2 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

4.1.3.3 hpke_kdf_strtab

```
const char* hpke_kdf_strtab[]
```

Initial value:

```
={  
    NULL,  
    HPKE_KDFSTR_256,  
    HPKE_KDFSTR_384,  
    HPKE_KDFSTR_512}  
}
```

4.1.3.4 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_SHA384, EVP_sha384, 48 },  
    { HPKE_KDF_ID_HKDF_SHA512, EVP_sha512, 64 }  
}
```

table of KDFs

4.1.3.5 hpke_kem_strtab

```
const char* hpke_kem_strtab[]
```

Initial value:

```
={  
    NULL, NULL, NULL, NULL,  
    NULL, NULL, NULL, NULL,  
    NULL, NULL, NULL, NULL,  
    NULL, NULL, NULL, NULL,  
    HPKE_KEMSTR_P256,  
    HPKE_KEMSTR_P384,  
    HPKE_KEMSTR_P521,  
    NULL,  
    NULL, NULL, NULL, NULL,  
    NULL, NULL, NULL, NULL,  
    NULL, NULL, NULL, NULL,  
    HPKE_KEMSTR_X25519,  
    HPKE_KEMSTR_X448,  
    NULL}
```

4.1.3.6 hpke_kem_tab

```
hpke_kem_info_t hpke_kem_tab[]
```

table of KEMs

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

4.1.3.7 hpke_mode_strtab

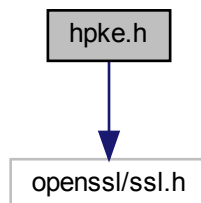
```
const char* hpke_mode_strtab[]
```

Initial value:

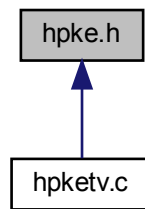
```
={  
    HPKE_MODESTR_BASE,  
    HPKE_MODESTR_PSK,  
    HPKE_MODESTR_AUTH,  
    HPKE_MODESTR_PSKAUTH}
```

4.2 hpke.h File Reference

```
#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_MAXSIZE](#) (40*1024)
40k is more than enough for anyone (using this program:-)
- #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.*

- `#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.

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_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 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, EVP_PKEY *senderpriv, size_t *cipherlen, unsigned char *cipher)
Internal HPKE single-shot encryption function.
- `int hpke_enc_raw` (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 extsenderpublen, unsigned char *extsenderpub, size_t rawsenderprivlen, unsigned char *rawsenderpriv, 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 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_kg_ev` (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_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)
- `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)
- `int hpke_prbuf2ev` (unsigned int kem_id, unsigned char *prbuf, size_t prbuf_len, unsigned char *pubuf, size_t pubuf_len, EVP_PKEY **priv)

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

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-caculate 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,
    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 * 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>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 info,
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 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 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 length of the additional data (can be zero)
<i>aad</i>	is the encoded additional data (can be NULL)
<i>info</i>	is the length 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_enc_evpc()

```

int hpke_enc_evpc (
    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 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>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>infolen</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 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>	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

4.2.3.5 hpke_enc_raw()

```
int hpke_enc_raw (
    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 extsenderpublen,
    unsigned char * extsenderpub,
    size_t rawsenderprivlen,
    unsigned char * rawsenderpriv,
    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>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>infolen</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 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>	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

4.2.3.6 hpke_expand()

```
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 )
```

brief 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

4.2.3.7 hpke_extract()

```
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 )
```

brief 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!

4.2.3.8 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

4.2.3.9 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

4.2.3.10 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 )
```

brief: 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.

4.2.3.11 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
--------------	-----------------------

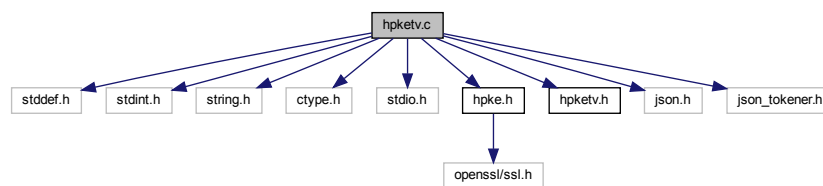
Returns

1 for good/supported, not-1 otherwise

4.3 hpktv.c File Reference

```
#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

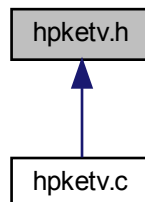
<i>nelems</i>	is the number of array elements
<i>array</i>	is the elements

Returns

1 for good, other for bad

4.4 hpketv.h File Reference

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 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

hpketv.c, 46

hpke.c, 11

hpke_aead_dec, 14
hpke_aead_enc, 15
hpke_aead_strtab, 29
hpke_aead_tab, 30
hpke_ah_decode, 15
hpke_dec, 16
hpke_do_kem, 17
hpke_enc, 18
hpke_enc_ev, 20
hpke_enc_int, 21
hpke_enc_raw, 22
hpke_EVP_PKEY_new_raw_nist_public_key, 23
hpke_expand, 24
hpke_extract, 24
hpke_extract_and_expand, 25
hpke_kdf_strtab, 30
hpke_kdf_tab, 30
hpke_kem_id_check, 26
hpke_kem_id_nist_curve, 26
hpke_kem_strtab, 30
hpke_kem_tab, 31
hpke_kg, 27
hpke_kg_ev, 27
hpke_mode_check, 28
hpke_mode_strtab, 31
hpke_prbuf2ev, 28
hpke_psk_check, 29
hpke_suite_check, 29

hpke.h, 31

HPKE_A2B, 35
hpke_ah_decode, 35
hpke_dec, 36
hpke_enc, 37
hpke_enc_ev, 38
hpke_enc_raw, 39
hpke_expand, 41
hpke_extract, 41
hpke_kg, 42
hpke_kg_ev, 43
hpke_prbuf2ev, 44
hpke_suite_check, 44
HPKE_SUITE_DEFAULT, 35

HPKE_A2B

hpke.h, 35

hpke_aead_dec

hpke.c, 14

hpke_aead_enc

hpke.c, 15

hpke_aead_info_t, 5

hpke_aead_strtab

hpke.c, 29

hpke_aead_tab

hpke.c, 30

hpke_ah_decode

hpke.c, 15

hpke.h, 35

hpke_dec

hpke.c, 16

hpke.h, 36

hpke_do_kem

hpke.c, 17

hpke_enc

hpke.c, 18

hpke.h, 37

hpke_enc_ev

hpke.c, 20

hpke.h, 38

hpke_enc_int

hpke.c, 21

hpke_enc_raw

hpke.c, 22

hpke.h, 39

hpke_EVP_PKEY_new_raw_nist_public_key

hpke.c, 23

hpke_expand

hpke.c, 24

hpke.h, 41

hpke_extract

hpke.c, 24

hpke.h, 41

hpke_extract_and_expand

hpke.c, 25

hpke_kdf_info_t, 5

hpke_kdf_strtab

hpke.c, 30

hpke_kdf_tab

hpke.c, 30

hpke_kem_id_check

hpke.c, 26

hpke_kem_id_nist_curve

hpke.c, 26

hpke_kem_info_t, 6

hpke_kem_strtab

hpke.c, 30

hpke_kem_tab

- hpke.c, [31](#)
- hpke_kg
 - hpke.c, [27](#)
 - hpke.h, [42](#)
- hpke_kg_evp
 - hpke.c, [27](#)
 - hpke.h, [43](#)
- hpke_mode_check
 - hpke.c, [28](#)
- hpke_mode_strtab
 - hpke.c, [31](#)
- hpke_prbuf2evp
 - hpke.c, [28](#)
 - hpke.h, [44](#)
- hpke_psk_check
 - hpke.c, [29](#)
- hpke_suite_check
 - hpke.c, [29](#)
 - 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](#)