Happy Key: HPKE implementation (RFC9180)

https://github.com/sftcd/happykey

Generated by Doxygen 1.9.1

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

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
- const char ∗ **name**
- 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 algtype "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:

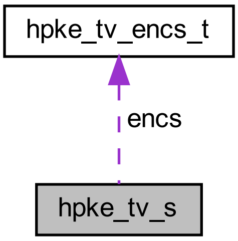
- hpketv.h

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

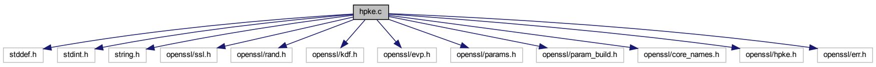
File Documentation

4.1 hpke.c File Reference

An OpenSSL-based HPKE implementation of RFC9180.

```
#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 <openssl/hpke.h>
#include <openssl/err.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_err** { ERR_raise(ERR_LIB_SSL, ERR_R_INTERNAL_ERROR); erv = __LINE__; goto err; }
- #define HPKE_VERLABEL "HPKE-v1"

  version string label
- #define HPKE_SEC41LABEL "KEM"

  "suite_id" label for 4.1
- #define HPKE_SEC51LABEL "HPKE"

  "suite_id" label for 5.1
- #define HPKE_EAE_PRK_LABEL "eae_prk"

  label in ExtractAndExpand
- #define HPKE_PSKIDHASH_LABEL "psk_id_hash"

  in key_schedule_context
- #define HPKE_INFOHASH_LABEL "info_hash"

  in 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

- 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 (OSSL_LIB_CTX ∗libctx, 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 (OSSL_LIB_CTX ∗libctx, 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 AEAD encryption as per the RFC
- static int hpke_extract (OSSL_LIB_CTX ∗libctx, 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 (OSSL_LIB_CTX ∗libctx, 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 (OSSL_LIB_CTX ∗libctx, 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 (OSSL_LIB_CTX ∗libctx, 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_prbuf2evp (OSSL_LIB_CTX ∗libctx, 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_suite_check (hpke_suite_t suite)

  check if a suite is supported locally
- static int hpke_enc_int (OSSL_LIB_CTX ∗libctx, 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 rawsenderprivlen, unsigned char ∗rawsenderpriv, size_t ∗senderpublen, unsigned char ∗senderpub, size_t ∗cipherlen, unsigned char ∗cipher)

  Internal HPKE single-shot encryption function.
- static int hpke_dec_int (OSSL_LIB_CTX ∗libctx, 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.
- static int hpke_kg_evp (OSSL_LIB_CTX ∗libctx, unsigned int mode, hpke_suite_t suite, size_t ∗publen, unsigned char ∗pub, EVP_PKEY ∗∗priv)

  generate a key pair keeping private inside API
- static int hpke_kg (OSSL_LIB_CTX ∗libctx, unsigned int mode, hpke_suite_t suite, size_t ∗publen, unsigned char ∗pub, size_t ∗privlen, unsigned char ∗priv)

  generate a key pair
- static int hpke_random_suite (hpke_suite_t ∗suite)

  randomly pick a suite
- static 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
- static int hpke_str2suite (char ∗suitestr, hpke_suite_t ∗suite)

map a string to a HPKE suite

- static int hpke_expansion (hpke_suite_t suite, size_t clearlen, size_t ∗cipherlen)

  tell the caller how big the cipertext will be
- int **OSSL_HPKE_enc** (OSSL_LIB_CTX ∗libctx, 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)
- int **OSSL_HPKE_enc_evp** (OSSL_LIB_CTX ∗libctx, 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, EVP_PKEY ∗senderpriv, size_t ∗cipherlen, unsigned char ∗cipher)
- int **OSSL_HPKE_dec** (OSSL_LIB_CTX ∗libctx, 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 seqlen, unsigned char ∗seq, size_t ∗clearlen, unsigned char ∗clear)
- int OSSL_HPKE_kg (OSSL_LIB_CTX ∗libctx, unsigned int mode, hpke_suite_t suite, size_t ∗publen, unsigned char ∗pub, size_t ∗privlen, unsigned char ∗priv)

  generate a key pair
- int OSSL_HPKE_kg_evp (OSSL_LIB_CTX ∗libctx, 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 OSSL_HPKE_suite_check (hpke_suite_t suite)

  check if a suite is supported locally
- int OSSL_HPKE_prbuf2evp (OSSL_LIB_CTX ∗libctx, 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 OSSL_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 OSSL_HPKE_str2suite (char ∗str, hpke_suite_t ∗suite)

  map a string to a HPKE suite
- int OSSL_HPKE_expansion (hpke_suite_t suite, size_t clearlen, size_t ∗cipherlen)

  tell the caller how big the cipertext will be

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

4.1.1 Detailed Description

An OpenSSL-based HPKE implementation of RFC9180.

4.1.2 Function Documentation

4.1.2.1 hpke_aead_dec()

```
static int hpke_aead_dec (
             OSSL_LIB_CTX * libctx,
             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

libctx	is the context to use (normally NULL)
suite	is the ciphersuite
key	is the secret
keylen	is the length of the secret
iv	is the initialisation vector
ivlen	is the length of the iv
aad	is the additional authenticated data
aadlen	is the length of the aad
cipher	is obvious
cipherlen	is the ciphertext length
plain	is an output
plainlen	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 (
             OSSL_LIB_CTX * libctx,
             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 AEAD encryption as per the RFC

Parameters

libctx	is the context to use (normally NULL)
suite	is the ciphersuite
key	is the secret
keylen	is the length of the secret
iv	is the initialisation vector
ivlen	is the length of the iv
aad	is the additional authenticated data
aadlen	is the length of the aad
plain	is an output
plainlen	is the length of plain
cipher	is an output
cipherlen	input/output, better be big enough on input, exact on output

Returns

1 for good otherwise bad

4.1.2.3 hpke_dec_int()

```
static int hpke_dec_int (
            OSSL_LIB_CTX * libctx,
            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 )  [static]
```

HPKE single-shot decryption function.

Parameters

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

Returns

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

4.1.2.4 hpke_do_kem()

```
static int hpke_do_kem (
            OSSL_LIB_CTX * libctx,
            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

libctx	is the context to use (normally NULL)
encrypting	is 1 if we're encrypting, 0 for decrypting
suite	is the ciphersuite
key1	is the first key, for which we have the private value
key1enclen	is the length of the encoded form of key1
key1en	is the encoded form of key1
key2	is the peer's key
key2enclen	is the length of the encoded form of key1
key2en	is the encoded form of key1
akey	is the authentication private key
apublen	is the length of the encoded the authentication public key
apub	is the encoded form of the authentication public key
ss	is (a pointer to) the buffer for the shared secret result
sslen	is the size of the buffer (octets-used on exit)

Returns

1 for good, not 1 for not good

4.1.2.5 hpke_enc_int()

```
static int hpke_enc_int (
            OSSL_LIB_CTX * libctx,
            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 rawsenderprivlen,
            unsigned char * rawsenderpriv,
            size_t * senderpublen,
            unsigned char * senderpub,
            size_t * cipherlen,
            unsigned char * cipher )  [static]
```

Internal HPKE single-shot encryption function.

Parameters

libctx	is the context to use (normally NULL)
mode	is the HPKE mode
suite	is the ciphersuite to use
pskid	is the pskid string fpr a PSK mode (can be NULL)
psklen	is the psk length
psk	is the psk
publen	is the length of the recipient public key
pub	is the encoded recipient public key
authprivlen	is the length of the private (authentication) key
authpriv	is the encoded private (authentication) key
authpriv_evp	is the EVP_PKEY∗ form of private (authentication) key
clearlen	is the length of the cleartext
clear	is the encoded cleartext
aadlen	is the lenght of the additional data (can be zero)
aad	is the encoded additional data (can be NULL)
infolen	is the lenght of the info data (can be zero)
info	is the encoded info data (can be NULL)
seqlen	is the length of the sequence data (can be zero)
seq	is the encoded sequence data (can be NULL)
extsenderpublen	length of the input buffer for sender's public key
extsenderpub	is the input buffer for sender public key
extsenderpriv	has the handle for the sender private key
senderpublen	length of the input buffer for sender's public key
senderpub	is the input buffer for ciphertext
cipherlen	is the length of the input buffer for ciphertext
cipher	is the input buffer for ciphertext

Returns

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

4.1.2.6 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

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

Returns

a working EVP_PKEY * or NULL

4.1.2.7 hpke_expand()

```
static int hpke_expand (
             OSSL_LIB_CTX * libctx,
             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]
```

RFC5869 HKDF-Expand.

Parameters

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

Returns

1 for good otherwise bad

4.1.2.8 hpke_expansion()

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

tell the caller how big the cipertext will be

AEAD algorithms add a tag for data authentication. Those are almost always, but not always, 16 octets long, and who knows 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

suite	is the suite to be used
clearlen	is the length of plaintext
cipherlen	points to what'll be ciphertext length

Returns

1 for success, otherwise failure

4.1.2.9 hpke_extract()

```
static int hpke_extract (
            OSSL_LIB_CTX * libctx,
            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

libctx	is the context to use (normally NULL)
suite	is the ciphersuite

Parameters

mode5869	- controls labelling specifics
salt	- surprisingly this is the salt;-)
saltlen	- length of above
label	- label for separation
labellen	- length of above
zz	- the initial key material (IKM)
zzlen	- length of above
secret	- the result of extraction (allocated inside)
secretlen	- 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", I2↩OSP(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!

4.1.2.10 hpke_extract_and_expand()

```
static int hpke_extract_and_expand (
             OSSL_LIB_CTX * libctx,
             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

libctx	is the context to use (normally NULL)
suite	is the ciphersuite
mode5869	- controls labelling specifics
shared_secret	- the initial DH shared secret
shared_secretlen	- length of above
context	- the info
contextlen	- length of above
secret	- the result of extract&expand
secretlen	- buf size on input

Returns

1 for good otherwise bad

4.1.2.11 hpke_good4grease()

```
static 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 )  [static]
```

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

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

Parameters

suite-in	specifies the preferred suite or NULL for a random choice
suite	is the chosen or random suite
pub	a random value of the appropriate length for sender public value
pub_len	is the length of pub (buffer size on input)
cipher	buffer with random value of the appropriate length
cipher_len	is the length of cipher

Returns

1 for success, otherwise failure

4.1.2.12 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

kem← _id	is the externally supplied kem_id

Returns

1 for good, not 1 for error

4.1.2.13 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

kem←_id	is the externally supplied kem_id

Returns

1 for NIST, 0 for good-but-non-NIST, other otherwise

4.1.2.14 hpke_kg()

```
static int hpke_kg (
            OSSL_LIB_CTX * libctx,
            unsigned int mode,
            hpke_suite_t suite,
            size_t * publen,
            unsigned char * pub,
            size_t * privlen,
            unsigned char * priv ) [static]
```

generate a key pair

Parameters

libctx	is the context to use (normally NULL)
mode	is the mode (currently unused)
suite	is the ciphersuite
publen	is the size of the public key buffer (exact length on output)
pub	is the public value
privlen	is the size of the private key buffer (exact length on output)
priv	is the private key

Returns

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

4.1.2.15 hpke_kg_evp()

```
static int hpke_kg_evp (
            OSSL_LIB_CTX * libctx,
            unsigned int mode,
            hpke_suite_t suite,
            size_t * publen,
            unsigned char * pub,
            EVP_PKEY ** priv ) [static]
```

generate a key pair keeping private inside API

Parameters

libctx	is the context to use (normally NULL)
mode	is the mode (currently unused)
suite	is the ciphersuite
publen	is the size of the public key buffer (exact length on output)
pub	is the public value
priv	is the private key pointer

Returns

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

4.1.2.16 hpke_mode_check()

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

check mode is in-range and supported

Parameters

mode	is the caller's chosen mode

Returns

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

4.1.2.17 hpke_prbuf2evp()

```
static int hpke_prbuf2evp (
            OSSL_LIB_CTX * libctx,
            unsigned int kem_id,
            unsigned char * prbuf,
            size_t prbuf_len,
            unsigned char * pubuf,
            size_t pubuf_len,
            EVP_PKEY ** retpriv )  [static]
```

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 encoded private key, and could still have the PEM header or not, and might or might not be base64 encoded. We'll try handle all those options.

Parameters

libctx	is the context to use (normally NULL)
kem_id	is what'd you'd expect (using the HPKE registry values)
prbuf	is the private key buffer
prbuf_len	is the length of that buffer
pubuf	is the public key buffer (if available)
pubuf_len	is the length of that buffer
priv	is a pointer to an EVP_PKEY * for the result

Returns

1 for success, otherwise failure

4.1.2.18 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

mode	is the mode in use
pskid	PSK identifier
psklen	length of PSK
psk	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.

4.1.2.19 hpke_random_suite()

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

randomly pick a suite

Parameters

suite	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.

4.1.2.20 hpke_str2suite()

```
static int hpke_str2suite (
             char * suitestr,
             hpke_suite_t * suite )  [static]
```

map a string to a HPKE suite

Parameters

str	is the string value
suite	is the resulting suite

Returns

1 for success, otherwise failure

4.1.2.21 hpke_suite_check()

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

check if a suite is supported locally

Parameters

suite	is the suite to check

Returns

1 for good/supported, not 1 otherwise

4.1.2.22 OSSL_HPKE_expansion()

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

tell the caller how big the cipertext 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

suite	is the suite to be used
clearlen	is the length of plaintext
cipherlen	points to what'll be ciphertext length

Returns

1 for success, otherwise failure

4.1.2.23 OSSL_HPKE_good4grease()

```
int OSSL_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

suite_in	specifies the preferred suite or NULL for a random choice
suite	is the chosen or random suite
pub	a random value of the appropriate length for a sender public value
pub_len	is the length of pub (buffer size on input)
cipher	is a random value of the appropriate length for a ciphertext
cipher_len	is the length of cipher

Returns

1 for success, otherwise failure

4.1.2.24 OSSL_HPKE_kg()

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

generate a key pair

Parameters

libctx	is the context to use (normally NULL)
mode	is the mode (currently unused)
suite	is the ciphersuite (currently unused)
publen	is the size of the public key buffer (exact length on output)
pub	is the public value
privlen	is the size of the private key buffer (exact length on output)
priv	is the private key

Returns

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

4.1.2.25 OSSL_HPKE_kg_evp()

```
int OSSL_HPKE_kg_evp (
             OSSL_LIB_CTX * libctx,
             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

libctx	is the context to use (normally NULL)
mode	is the mode (currently unused)
suite	is the ciphersuite (currently unused)
publen	is the size of the public key buffer (exact length on output)
pub	is the public value
priv	is the private key handle

Returns

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

4.1.2.26 OSSL_HPKE_prbuf2evp()

```
int OSSL_HPKE_prbuf2evp (
             OSSL_LIB_CTX * libctx,
             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

Parameters

libctx	is the context to use (normally NULL)
kem_id	is what'd you'd expect (using the HPKE registry values)
prbuf	is the private key buffer
prbuf_len	is the length of that buffer
pubuf	is the public key buffer (if available)
pubuf_len	is the length of that buffer
priv	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.

4.1.2.27 OSSL_HPKE_str2suite()

```
int OSSL_HPKE_str2suite (
             char * str,
             hpke_suite_t * suite )
```

map a string to a HPKE suite

Parameters

str	is the string value
suite	is the resulting suite

Returns

1 for success, otherwise failure

4.1.2.28 OSSL_HPKE_suite_check()

```
int OSSL_HPKE_suite_check (
            hpke_suite_t suite )
```

check if a suite is supported locally

Parameters

suite	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, NULL, 0, 0, 0 },
    { HPKE_AEAD_ID_AES_GCM_128, EVP_aes_128_gcm, "AES-128-GCM", 16, 16, 12 },
    { HPKE_AEAD_ID_AES_GCM_256, EVP_aes_256_gcm, "AES-256-GCM", 16, 32, 12 },
    { HPKE_AEAD_ID_CHACHA_POLY1305, EVP_chacha20_poly1305,
        "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

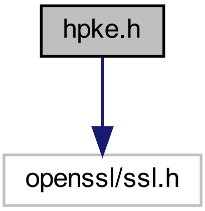
Ok we're wasting space here, but not much and it's ok

4.2 hpke.h File Reference

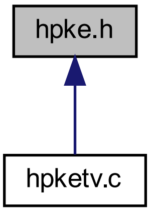
This has the data structures and prototypes (both internal and external) for an OpenSSL-based HPKE implementation of RFC9180.

```
#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 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_12 }
- #define **HPKE_SUITE_TURNITUPTO11** { HPKE_KEM_ID_448, HPKE_KDF_ID_HKDF_SHA512, HPKE_AEAD_ID_CHACHA_POLY1305 }

Functions

- int **OSSL_HPKE_enc** (OSSL_LIB_CTX *libctx, 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)
- int **OSSL_HPKE_enc_evp** (OSSL_LIB_CTX *libctx, 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, EVP_PKEY *senderpriv, size_t *cipherlen, unsigned char *cipher)
- int **OSSL_HPKE_dec** (OSSL_LIB_CTX *libctx, 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 seqlen, unsigned char *seq, size_t *clearlen, unsigned char *clear)
- int OSSL_HPKE_kg (OSSL_LIB_CTX *libctx, unsigned int mode, hpke_suite_t suite, size_t *publen, unsigned char *pub, size_t *privlen, unsigned char *priv)

  generate a key pair
- int OSSL_HPKE_kg_evp (OSSL_LIB_CTX *libctx, 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 OSSL_HPKE_suite_check (hpke_suite_t suite)

  check if a suite is supported locally
- int OSSL_HPKE_prbuf2evp (OSSL_LIB_CTX ∗libctx, 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 OSSL_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 OSSL_HPKE_str2suite (char ∗str, hpke_suite_t ∗suite)

  map a string to a HPKE suite
- int OSSL_HPKE_expansion (hpke_suite_t suite, size_t clearlen, size_t ∗cipherlen)

  tell the caller how big the cipertext will be

4.2.1 Detailed Description

This has the data structures and prototypes (both internal and external) for an OpenSSL-based HPKE implementation of RFC9180.

4.2.2 Macro Definition Documentation

4.2.2.1 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 OSSL_HPKE_expansion()

```
int OSSL_HPKE_expansion (
             hpke_suite_t suite,
             size_t clearlen,
             size_t ∗ cipherlen )
```

tell the caller how big the cipertext 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

suite	is the suite to be used
clearlen	is the length of plaintext
cipherlen	points to what'll be ciphertext length

Returns

1 for success, otherwise failure

4.2.3.2 OSSL_HPKE_good4grease()

```
int OSSL_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

suite_in	specifies the preferred suite or NULL for a random choice
suite	is the chosen or random suite
pub	a random value of the appropriate length for a sender public value
pub_len	is the length of pub (buffer size on input)
cipher	is a random value of the appropriate length for a ciphertext
cipher_len	is the length of cipher

Returns

1 for success, otherwise failure

4.2.3.3 OSSL_HPKE_kg()

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

generate a key pair

Parameters

libctx	is the context to use (normally NULL)
mode	is the mode (currently unused)
suite	is the ciphersuite (currently unused)
publen	is the size of the public key buffer (exact length on output)
pub	is the public value
privlen	is the size of the private key buffer (exact length on output)
priv	is the private key

Returns

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

4.2.3.4 OSSL_HPKE_kg_evp()

```
int OSSL_HPKE_kg_evp (
            OSSL_LIB_CTX * libctx,
            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

libctx	is the context to use (normally NULL)
mode	is the mode (currently unused)
suite	is the ciphersuite (currently unused)
publen	is the size of the public key buffer (exact length on output)
pub	is the public value
priv	is the private key handle

Returns

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

4.2.3.5 OSSL_HPKE_prbuf2evp()

```
int OSSL_HPKE_prbuf2evp (
            OSSL_LIB_CTX * libctx,
            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

Parameters

libctx	is the context to use (normally NULL)
kem_id	is what'd you'd expect (using the HPKE registry values)
prbuf	is the private key buffer
prbuf_len	is the length of that buffer
pubuf	is the public key buffer (if available)
pubuf_len	is the length of that buffer
priv	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.

4.2.3.6 OSSL_HPKE_str2suite()

```
int OSSL_HPKE_str2suite (
            char * str,
            hpke_suite_t * suite )
```

map a string to a HPKE suite

Parameters

str	is the string value
suite	is the resulting suite

Returns

1 for success, otherwise failure

4.2.3.7 OSSL_HPKE_suite_check()

```
int OSSL_HPKE_suite_check (
            hpke_suite_t suite )
```

check if a suite is supported locally

Parameters

suite	is the suite to check

Returns

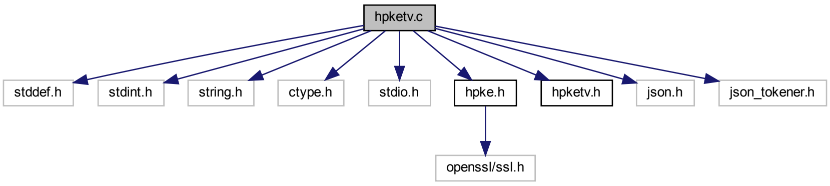
1 for good/supported, not-1 otherwise

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_tokener.h>
```

Include dependency graph for hpketv.c:



Macros

- #define FAIL2BUILD(x) int x;
- #define HPKE_A2B(__c__)

  Map ascii to binary - utility macro used in >1 place.
- #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/te` `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.

4.3.2.2 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.

4.3.3 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

nelems	is the number of array elements
array	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

fname	is the json file
nelems	returns with the number of array elements
array	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

mode	is the selected mode
suite	is the ciphersuite
nelems	is the number of array elements
arr	is the elements
tv	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

nelems	is the number of array elements
array	is the elements

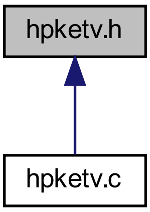
Returns

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/te` `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

nelems	is the number of array elements
array	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

fname	is the json file
nelems	returns with the number of array elements
array	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

mode	is the selected mode
suite	is the ciphersuite
nelems	is the number of array elements
arr	is the elements
tv	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

nelems	is the number of array elements
array	is the elements

Returns

1 for good, other for bad

Index