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

Generated by Doxygen 1.8.13

Contents

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
- size_t taglen

  aead tag len
- size_t Nk

  size of a key for this aead
- size_t Nn

  length of a nonce for this aead

3.1.1 Detailed Description

info about an AEAD

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

- hpke.c

3.2 hpke_kdf_info_t Struct Reference

info about a KDF

Data Fields

- uint16_t kdf_id

  code point for KDF
- const EVP_MD ∗(∗ hash_init_func)(void)

  the hash alg we're using
- size_t Nh

  length of hash/extract output

3.2.1 Detailed Description

info about a KDF

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

- hpke.c

3.3 hpke_kem_info_t Struct Reference

info about a KEM

Data Fields

- uint16_t kem_id

  code point for key encipherment method
- int groupid

  NID of KEM.
- size_t Nenc

  length of encapsulated key
- size_t Npk

  length of public key

3.3.1 Detailed Description

info about a KEM

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

- hpke.c

3.4 hpke_suite_t Struct Reference

ciphersuite combination

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

Data Fields

- uint16_t kem_id

  Key Encryption Method id.
- uint16_t kdf_id

  Key Derivation Function id.
- uint16_t aead_id

  Authenticated Encryption with Associated Data id.

3.4.1 Detailed Description

ciphersuite combination

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

- hpke.h

3.5 hpke_tv_encs_t Struct Reference

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

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

Data Fields

- const char ∗ aad

  ascii-hex encoded additional authenticated data
- const char ∗ plaintext

  aascii-hex encoded plaintext
- const char ∗ ciphertext

  ascii-hex encoded ciphertext

3.5.1 Detailed Description

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

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

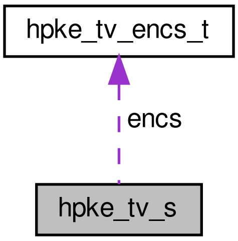
- hpketv.h

3.6 hpke_tv_s Struct Reference

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

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

Collaboration diagram for hpke_tv_s:



Data Fields

- uint8_t **mode**
- uint16_t **kdfID**
- uint16_t **aeadID**
- uint16_t **kemID**
- const char ∗ **context**
- const char ∗ **skI**
- const char ∗ **pkI**
- const char ∗ **zz**
- const char ∗ **secret**
- const char ∗ **enc**
- const char ∗ **info**
- const char ∗ **pskID**
- const char ∗ **nonce**
- const char ∗ **key**
- const char ∗ **pkR**
- const char ∗ **pkE**
- const char ∗ **skR**
- const char ∗ **skE**
- const char ∗ **psk**
- int **nencs**
- hpke_tv_encs_t ∗ **encs**
- void ∗ jobj

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

3.6.1 Detailed Description

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

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

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

- hpketv.h

Chapter 4

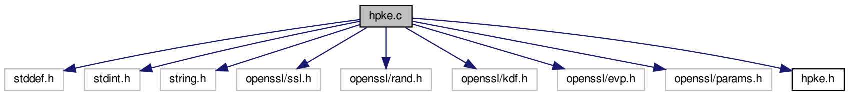
File Documentation

4.1 hpke.c File Reference

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

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

Include dependency graph for hpke.c:



Data Structures

- struct hpke_aead_info_t

  info about an AEAD
- struct hpke_kem_info_t

  info about a KEM
- struct hpke_kdf_info_t

  info about a KDF

Macros

- #define HPKE_A2B(__c__)

  Map ascii to binary.
- #define CHECK_HPKE_CTX if ((cp-∗context)>∗contextlen) { erv=__LINE__; goto err; }

  make it easier to do repetitive code
- #define **ISAUTHMODE**(xxmode) (xxmode==HPKE_MODE_AUTH || xxmode==HPKE_MODE_PSKAUTH)
- #define **ISPSKMODE**(xxmode) (xxmode==HPKE_MODE_PSK || xxmode==HPKE_MODE_PSKAUTH)

Functions

- int hpke_ah_decode (size_t ahlen, const char ∗ah, size_t ∗blen, unsigned char ∗∗buf)

    decode ascii hex to a binary buffer
- static int hpke_pbuf (FILE ∗fout, char ∗msg, unsigned char ∗buf, size_t blen)

    for odd/occasional debugging
- static int hpke_suite_check (hpke_suite_t suite)

    Check if ciphersuite is ok/known to us.
- static size_t figure_contextlen (hpke_suite_t suite)

    return the length of the context for this suite
- static int hpke_aead_dec (hpke_suite_t suite, unsigned char ∗key, size_t keylen, unsigned char ∗iv, size←↩ _t ivlen, unsigned char ∗aad, size_t aadlen, unsigned char ∗cipher, size_t cipherlen, unsigned char ∗plain, size_t ∗plainlen)

    do the AEAD decryption
- static int hpke_aead_enc (hpke_suite_t suite, unsigned char ∗key, size_t keylen, unsigned char ∗iv, size←↩ _t ivlen, unsigned char ∗aad, size_t aadlen, unsigned char ∗plain, size_t plainlen, unsigned char ∗cipher, size_t ∗cipherlen)

    do the AEAD encryption as per the I-D
- static int hpke_extract (hpke_suite_t suite, const unsigned char ∗salt, const size_t saltlen, const unsigned char ∗zz, const size_t zzlen, unsigned char ∗∗secret, const size_t secretlen)
- static int hpke_expand (hpke_suite_t suite, unsigned char ∗secret, size_t secretlen, char ∗label, unsigned char ∗context, size_t contextlen, unsigned char ∗∗out, size_t outlen)
- static int hpke_do_kem (EVP_PKEY ∗key1, EVP_PKEY ∗key2, unsigned char ∗∗zz, size_t ∗zzlen)

    run the KEM with two keys
- static int hpke_make_context (int mode, hpke_suite_t suite, const unsigned char ∗enc, const size_t enclen, const unsigned char ∗pub, const size_t publen, const unsigned char ∗pkI_hash, const size_t pkI_←↩ hashlen, const char ∗pskid, const unsigned char ∗info, const size_t infolen, unsigned char ∗∗context, size_t ∗contextlen)

    Create context for input to extract/expand.
- static int hpke_mode_check (unsigned int mode)

    check mode is in-range and supported
- static int hpke_psk_check (unsigned int mode, char ∗pskid, size_t psklen, unsigned char ∗psk)

    check psk params are as per spec
- int hpke_enc (unsigned int mode, hpke_suite_t suite, char ∗pskid, size_t psklen, unsigned char ∗psk, size←↩ _t publen, unsigned char ∗pub, size_t privlen, unsigned char ∗priv, size_t clearlen, unsigned char ∗clear, size_t aadlen, unsigned char ∗aad, size_t infolen, unsigned char ∗info, size_t ∗senderpublen, unsigned char ∗senderpub, size_t ∗cipherlen, unsigned char ∗cipher)

    HPKE single-shot encryption function.
- int hpke_dec (unsigned int mode, hpke_suite_t suite, char ∗pskid, size_t psklen, unsigned char ∗psk, size_t publen, unsigned char ∗pub, size_t privlen, unsigned char ∗priv, size_t 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

Variables

- hpke_aead_info_t hpke_aead_tab [ ]

    table of AEADs
- hpke_kem_info_t hpke_kem_tab [ ]

table of KEMs

- hpke_kdf_info_t hpke_kdf_tab [ ]

  table of KDFs

- static const unsigned char zero_buf [SHA512_DIGEST_LENGTH]

  handy thing to have :-)

4.1.1 Detailed Description

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

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

4.1.2 Macro Definition Documentation

4.1.2.1 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.

Bash command line hashing starting from ascii hex example:

$ echo -e "4f6465206f6e20612047726565636961e2055726e" | xxd -r -p | openssl sha256 (stdin)= 55c4040629c64c5efec2f7230407d61

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

If you'd like to regenerate the zero_sha256 value above, feel free $ echo -n "" | openssl sha256 echo -n "" | openssl sha256 (stdin)= e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855 Or if you'd like to re-caclulate the sha256 of nothing... SHA256_CTX sha256; SHA256_Init(&sha256); char∗ buffer = NULL; int bytesRead = 0; SHA256_Update(&sha256, buffer, bytesRead); SHA256_Final(zero_sha256, &sha256); ...but I've done it for you, so no need:-) static const unsigned char zero_sha256[SHA256_DIGEST_LENGTH] = { 0xe3, 0xb0, 0xc4, 0x42, 0x98, 0xfc, 0x1c, 0x14, 0x9a, 0xfb, 0xf4, 0xc8, 0x99, 0x6f, 0xb9, 0x24, 0x27, 0xae, 0x41, 0xe4, 0x64, 0x9b, 0x93, 0x4c, 0xa4, 0x95, 0x99, 0x1b, 0x78, 0x52, 0xb8, 0x55};

4.1.3 Function Documentation

4.1.3.1 figure_contextlen()

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

return the length of the context for this suite

Parameters

suite	is the ciphersuite to use

Returns

the length (in octets) of the context

4.1.3.2 hpke_aead_dec()

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

do the AEAD decryption

Parameters

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	is an input/output, better be big enough on input, exact on output

Returns

1 for good otherwise bad

4.1.3.3 hpke_aead_enc()

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

do the AEAD encryption as per the I-D

Parameters

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	is an input/output, better be big enough on input, exact on output

Returns

1 for good otherwise bad

4.1.3.4 hpke_ah_decode()

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

decode ascii hex to a binary buffer

Parameters

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

Returns

1 for good otherwise bad

4.1.3.5 hpke_dec()

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

HPKE single-shot decryption function.

Parameters

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
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)
clearlen	is the length of the input buffer for cleartext (octets used on output)
clear	is the encoded cleartext

Returns

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

4.1.3.6 hpke_do_kem()

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

run the KEM with two keys

Parameters

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

Returns

1 for good, not-1 for not good

4.1.3.7 hpke_enc()

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

HPKE single-shot encryption function.

Parameters

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
privlen	is the length of the private (authentication) key
priv	is the encoded 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)
senderpublen	is the length of the input buffer for the sender's public key (length used on output)
senderpub	is the input buffer for ciphertext
cipherlen	is the length of the input buffer for ciphertext (length used on output)
cipher	is the input buffer for ciphertext

Returns

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

4.1.3.8 hpke_expand()

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

brief RFC5869 HKDF-Expand

Parameters

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

Returns

1 for good otherwise bad

4.1.3.9 hpke_extract()

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

brief RFC5869 HKDF-Extract

Parameters

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

Returns

1 for good otherwise bad

4.1.3.10 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

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

4.1.3.11 hpke_make_context()

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

Create context for input to extract/expand.

Parameters

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

Returns

1 for good, not 1 otherwise

4.1.3.12 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

4.1.3.13 hpke_pbuf()

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

for odd/occasional debugging

Parameters

fout	is a FILE ∗ to use
msg	is prepended to print
buf	is the buffer to print
blen	is the length of the buffer

Returns

1 for success

4.1.3.14 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.

4.1.3.15 hpke_suite_check()

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

Check if ciphersuite is ok/known to us.

Parameters

suite	is the externally supplied cipheruite

Returns

1 for good, not-1 for error

For now, we only recognise HPKE_SUITE_DEFAULT

4.1.4 Variable Documentation

4.1.4.1 hpke_aead_tab

```
hpke_aead_info_t hpke_aead_tab[ ]
```

Initial value:

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

table of AEADs

4.1.4.2 hpke_kdf_tab

```
hpke_kdf_info_t hpke_kdf_tab[ ]
```

Initial value:

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

table of KDFs

4.1.4.3 hpke_kem_tab

```
hpke_kem_info_t hpke_kem_tab[ ]
```

Initial value:

```
={
    { 0, 0, 0, 0 },
    { HPKE_KEM_ID_P256, NID_secp256k1, 32, 32 },

    { HPKE_KEM_ID_25519, EVP_PKEY_X25519, 32, 32 },
    { HPKE_KEM_ID_P521, NID_secp521r1, 65, 65 },
    { HPKE_KEM_ID_448, EVP_PKEY_X448, 56, 56 }
}
```

table of KEMs

4.1.4.4 zero_buf

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

Initial value:

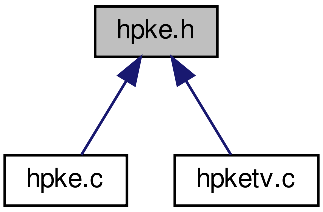
```
= {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00,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

handy thing to have :-)

4.2 hpke.h File Reference

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

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



Data Structures

- struct hpke_suite_t

    ciphersuite combination

Macros

- #define HPKE_MAXSIZE (640∗1024)

    640k is more than enough for anyone (using this program:-)
- #define HPKE_MODE_BASE 0

    Base mode (all that we support for now)
- #define HPKE_MODE_PSK 1

    Pre-shared key mode.
- #define HPKE_MODE_AUTH 2

    Authenticated mode.
- #define HPKE_MODE_PSKAUTH 3

    PSK+authenticated mode.
- #define HPKE_KEM_ID_RESERVED 0x0000

    not used
- #define HPKE_KEM_ID_P256 0x0001

    NIST P-256.
- #define HPKE_KEM_ID_25519 0x0002

    Curve25519.
- #define HPKE_KEM_ID_P521 0x0003

    NIST P-521.
- #define HPKE_KEM_ID_448 0x0004

    Curve448.
- #define HPKE_KEM_ID_MAX 0x0004

    Curve448.

- #define HPKE_KDF_ID_RESERVED 0x0000

  not used
- #define HPKE_KDF_ID_HKDF_SHA256 0x0001

  HKDF-SHA256.
- #define HPKE_KDF_ID_HKDF_SHA512 0x0002

  HKDF-SHA512.
- #define HPKE_KDF_ID_MAX 0x0002

  HKDF-SHA512.
- #define HPKE_AEAD_ID_RESERVED 0x0000

  not used
- #define HPKE_AEAD_ID_AES_GCM_128 0x0001

  AES-GCM-128.
- #define HPKE_AEAD_ID_AES_GCM_256 0x0002

  AES-GCM-256.
- #define HPKE_AEAD_ID_CHACHA_POLY1305 0x0003

  Chacha20-Poly1305.
- #define HPKE_AEAD_ID_MAX 0x0003

  Chacha20-Poly1305.
- #define HPKE_SUITE_DEFAULT { HPKE_KEM_ID_25519, HPKE_KDF_ID_HKDF_SHA256, HPKE_AEAD_ID_AES_GCM_128 }
- #define **HPKE_SUITE_TURNITUPTO11** { HPKE_KEM_ID_448, HPKE_KDF_ID_HKDF_SHA512, HPKE_AEAD_ID_CHACHA_POLY1305 }

Functions

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

  HPKE single-shot encryption function.
- int hpke_dec (unsigned int mode, hpke_suite_t suite, char ∗pskid, size_t psklen, unsigned char ∗psk, size_t publen, unsigned char ∗pub, size_t privlen, unsigned char ∗priv, size_t 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_ah_decode (size_t ahlen, const char ∗ah, size_t ∗blen, unsigned char ∗∗buf)

  decode ascii hex to a binary buffer

4.2.1 Detailed Description

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

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

4.2.2 Macro Definition Documentation

4.2.2.1 HPKE_SUITE_DEFAULT

```
#define HPKE_SUITE_DEFAULT { HPKE_KEM_ID_25519, HPKE_KDF_ID_HKDF_SHA256, HPKE_AEAD_ID_AES_GC←↩
M_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

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

Returns

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

Parameters

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

Returns

1 for good otherwise bad

4.2.3.2 hpke_dec()

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

HPKE single-shot decryption function.

Parameters

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
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)
clearlen	is the length of the input buffer for cleartext (octets used on output)
clear	is the encoded cleartext

Returns

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

4.2.3.3 hpke_enc()

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

HPKE single-shot encryption function.

Parameters

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
privlen	is the length of the private (authentication) key
priv	is the encoded 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)
senderpublen	is the length of the input buffer for the sender's public key (length used on output)
senderpub	is the input buffer for ciphertext
cipherlen	is the length of the input buffer for ciphertext (length used on output)
cipher	is the input buffer for ciphertext

Returns

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

4.2.3.4 hpke_kg()

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

generate a key pair

Parameters

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

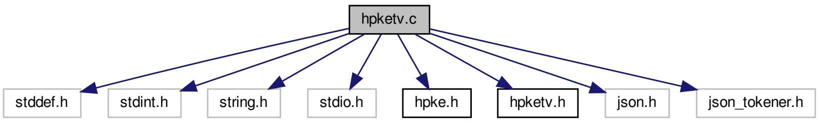
1 for good (OpenSSL style), not-1 for error

4.3 hpketv.c File Reference

Implementation related to test vectors for HPKE.

```
#include <stddef.h>
#include <stdint.h>
#include <string.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 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

- 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` `json` A copy from 20191126 is are 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

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

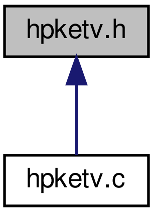
Returns

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/test` `json` A copy from 20191126 is are 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