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_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.

3.1.1 Detailed Description

ciphersuite combination

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

- hpke.h

3.2 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

3.2.1 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

3.3 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 from which we derived this

3.3.1 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:

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_PSK_AUTH 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_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_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_SUITE_DEFAULT { HPKE_KEM_ID_25519, HPKE_KDF_ID_HKDF_SHA256, HPKE_AEA↩ D_ID_AES_GCM_128 }

Functions

- 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_enc (unsigned int mode, hpke_suite_t suite, size_t publen, unsigned char ∗pub, 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, 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_pbuf (FILE *fout, char *msg, unsigned char *buf, size_t blen)

  for odd/occasional debugging

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_GCM_128 }
```

A suite constant (the only one supported for now:-) Use this as follows:

```
hpke_suit_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
ahstr	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
ahstr	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

zero for error, 1 for success

4.2.3.2 hpke_dec()

```
int hpke_dec (
            unsigned int mode,
            hpke_suite_t suite,
            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 to use
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,
             size_t recippublen,
             unsigned char * recippub,
             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
recippublen	is the length of the recipient public key
recippub	is the encoded recipient public 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

< Our error return value - 1 is success

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

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

< Our error return value - 1 is success

4.2.3.5 hpke_pbuf()

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

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 good (OpenSSL style), not-1 for error

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

4.3 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 **grabnum**(_xx)
- #define **grabstr**(_xx)
- #define **grabestr**(_xx)
- #define **PRINTIT**(_xx) printf("\t"#_xx": %s\n",a->_xx);

Functions

- assert ("TESTVECTORS not defined which is bad")
- 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
- int hpke_tv_pick (int nelems, hpke_tv_t *arr, char *selector, 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 grabestr

```
#define grabestr(
               _xx )
```

Value:

```
if (!strcmp(key1,""#_xx"")) { \
                      encs[j]._xx=json_object_get_string(val1); \
                    }
```

4.3.2.2 grabnum

```
#define grabnum(
               _xx )
```

Value:

```
if (!strcmp(key,""#_xx"")) { \
                      thearr[i]._xx=json_object_get_int(val); \
                    }
```

4.3.2.3 grabstr

```
#define grabstr(
               _xx )
```

Value:

```
if (!strcmp(key,""#_xx"")) { \
                      thearr[i]._xx=json_object_get_string(val); \
                    }
```

4.3.3 Function Documentation

4.3.3.1 assert()

```
assert (
             "TESTVECTORS not defined which is bad"  )
```

Crap out if this isn't defined.

4.3.3.2 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?

Returns

1 for good, other for bad

Caller doesn't need to free "parent" array

4.3.3.3 hpke_tv_load()

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

load test vectors from json file to array

Parameters

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

Returns

1 for good, other for bad

4.3.3.4 hpke_tv_pick()

```
int hpke_tv_pick (
            int nelems,
            hpke_tv_t * arr,
            char * selector,
            hpke_tv_t ** tv )
```

select a test vector to use based on mode and suite

Parameters

nelems	is the number of array elements
array	is the elements
selector	is a string to use
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.

4.3.3.5 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 (int nelems, hpke_tv_t ∗arr, char ∗selector, 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?

Returns

1 for good, other for bad

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

filename	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 (
            int nelems,
            hpke_tv_t * arr,
            char * selector,
            hpke_tv_t ** tv )
```

select a test vector to use based on mode and suite

Parameters

nelems	is the number of array elements
array	is the elements
selector	is a string to use
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.

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