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_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.

3.1.1 Detailed Description

ciphersuite combination

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

- hpke.h

Chapter 4

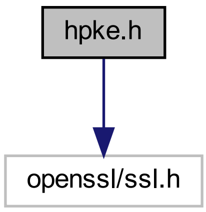
File Documentation

4.1 hpke.h File Reference

APIs and data structures for HPKE (RFC9180).

`#include <openssl/ssl.h>`

Include dependency graph for hpke.h:



Data Structures

- struct hpke_suite_t

    ciphersuite combination

Macros

- #define HPKE_MAXSIZE (2 ∗ 1024) /∗ 2k: enough for anyone :-) ∗/

    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 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

  Suite constants, use this like: hpke_suite_t myvar = HPKE_SUITE_DEFAULT;.
- #define HPKE_SUITE_TURNITUPTO11

  If you like your crypto turned up...

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)

  HPKE single-shot encryption function.
- 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)

  HPKE multi-shot encryption function.
- 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)

  HPKE single-shot decryption function.
- 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 (OSSL_LIB_CTX *libctx, 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

4.1.1 Detailed Description

APIs and data structures for HPKE (RFC9180).

There is only one significant data structure defined here (hpke_suite_t) to represent the KEM, KDF and AEAD algs used. Otherwise, the approach taken is to provide all the API inputs using existing types (buffers, lengths and a few cases of strings or EVP_PKEY pointers.

HPKE key generation functions (`OSSL_HPKE_kg()` and `OSSL_HPKE_kg_evp()`) require a KEM as input and return public and private components of the key.

HPKE encryption supports various "modes" that can optionally bind a pre-shared key (PSK) and/or an authenticatng private value, also generared via `OSSL_HPKE_kg()`, to the encryption operation - `HPKE_MODE_BASE` is the basic mode with neither, while `HPKE_MODE_PSKAUTH` requires both.

An `info` value, known to both encryptor and decryptor can be combined into the key agreement operation. Similarly, additional authenticated data (`aad`) can be combined into the AEAD operation. Applications/protocols using HPKE can use these to authenticate information that won't be part of the encryption.

Where the same public value is used for more than one encryption operation, a sequence number (`seq`) may also be mixed into the key agreement operation.

Single-shot encryption (`OSSL_HPKE_enc()`) requires the mode, suite, public value and cleartext inputs and produces the ciphertext output. The other optional inputs are as described above.

An `OSSL_HPKE_enc_evp()` variant allows the encryptor to re-use its Diffie-Hellman public and private values used in a previous call. The `seq` option is likely also needed in such cases, e.g. as part of some protocol re-try such as the TLS HelloRetryRequest (HRR) case for Encrypted Client Hello.

`OSSL_HPKE_dec()` supports the decryption operation and takes the same kinds of inputs as for encryption with the obvious role-swaps of public and private values.

`OSSL_HPKE_prbuf2evp()` converts a buffer containing a private value into an EVP_PKEY * pointer.

`OSSL_HPKE_suite_check()` can be used to determine if an HPKE suite is supported or not.

`OSSL_HPKE_str2suite()` maps from comma-separated strings, e.g. "x25519,hkdf-sha256,aes128gcm", to an `hpke_suite_t`.

So-called GREASEing (see RFC8701) is a protocol mechanism where phoney values are sent in order to make it less likely that (especially) middleboxes are deployed that only know about "current" protocol options. Protocols using HPKE (such as ECH) make use of this mechanism, but in that case need to produce realistic-looking, though still phoney, values. The `OSSL_HPKE_good4grease()` API can be used to generate such values.

As HPKE encryption uses an AEAD cipher, there is the usual expansion of ciphertext due to the authentication tag. Applications/protocols needing to know the degree of such expansion (whether for GREASEing or memory management) can use the `OSSL_HPKE_expansion()` API.

Many of the APIs defined here also take an `OSSL_LIB_CTX` pointer as input for cases where the default library context is not in use. Return values are always 1 in the case of success, or something else otherwise - note that non-zero failure return values will be seen by callers.

4.1.2 Macro Definition Documentation

4.1.2.1 HPKE_SUITE_DEFAULT

```
#define HPKE_SUITE_DEFAULT
```

Value:

```
{\
    HPKE_KEM_ID_25519, \
    HPKE_KDF_ID_HKDF_SHA256, \
    HPKE_AEAD_ID_AES_GCM_128 \
}
```

Suite constants, use this like: hpke_suite_t myvar = HPKE_SUITE_DEFAULT;.

4.1.2.2 HPKE_SUITE_TURNITUPTO11

```
#define HPKE_SUITE_TURNITUPTO11
```

Value:

```
{ \
    HPKE_KEM_ID_448, \
    HPKE_KDF_ID_HKDF_SHA512, \
    HPKE_AEAD_ID_CHACHA_POLY1305 \
}
```

If you like your crypto turned up...

4.1.3 Function Documentation

4.1.3.1 OSSL_HPKE_dec()

```
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 )
```

HPKE single-shot decryption 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 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 length of the additional data
aad	is the encoded additional data
infolen	is the length 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

4.1.3.2 OSSL_HPKE_enc()

```
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 )
```

HPKE single-shot encryption function.

This function generates an ephemeral ECDH value internally and provides the public component as an output that can be sent to the relevant private key holder along with the ciphertext.

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 public key
pub	is the encoded 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 length of the additional data
aad	is the encoded additional data
infolen	is the length 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)
senderpublen	length of the input buffer for sender's public key
senderpub	is the input buffer for sender public key
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

4.1.3.3 OSSL_HPKE_enc_evp()

```
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 )
```

HPKE multi-shot encryption function.

This function generates a non-ephemeral ECDH value internally and provides the public and private components as outputs. The public part can be sent to the relevant private key holder along with the ciphertext. The private part can be re-used in subequent calls.

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 public key
pub	is the encoded 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

Parameters

clear	is the encoded cleartext
aadlen	is the length of the additional data
aad	is the encoded additional data
infolen	is the length 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)
senderpublen	length of the input buffer for sender's public key
senderpub	is the input buffer for sender public key
senderpriv	is the EVP_PKEY* form of sender key pair
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

4.1.3.4 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

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

4.1.3.5 OSSL_HPKE_good4grease()

```
int OSSL_HPKE_good4grease (
            OSSL_LIB_CTX * libctx,
            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

Parameters

libctx	is the context to use (normally NULL)
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

4.1.3.6 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

Used for entities that will later receive HPKE values to decrypt. Only the KEM from the suite is significant here. The `` `pub `` output will typically be published so that others can encrypt to the private key holder using HPKE. The `priv` output contains the raw private value and hence is sensitive.

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

4.1.3.7 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

Used for entities that will later receive HPKE values to decrypt. Only the KEM from the suite is significant here. The `pub` output will typically be published so that others can encrypt to the private key holder using HPKE. The `priv` output here is in the form of an EVP_PKEY and so the raw private value need not be exposed to the application.

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

4.1.3.8 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.

4.1.3.9 OSSL_HPKE_str2suite()

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

map a string to a HPKE suite

An example good string is "x25519,hkdf-sha256,aes128gcm" Symbols are #define'd for the relevant labels, e.g. HPKE_KEMSTR_X25519. Numeric (decimal or hex) values with the relevant IANA codepoint valus may also be used, e.g., "0x20,1,1" represents the same suite as the first example.

Parameters

str	is the string value
suite	is the resulting suite

Returns

1 for success, otherwise failure

4.1.3.10 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

Index