

# **Linux Kernel Crypto API User Space Interface Library**

**Stephan Mueller**

**`smueller@chronox.de`**

# **Linux Kernel Crypto API User Space Interface Library**

by Stephan Mueller

1.2.0 Edition

Copyright © 2014 Stephan Mueller

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more details see the file COPYING in the source distribution of Linux.

# Table of Contents

<b>1. libkcapi - Linux Kernel Crypto API User Space Interface Library .....</b>	<b>1</b>
Version Number Schema .....	1
Purpose Of AF_ALG .....	1
<b>2. Programming Guidelines .....</b>	<b>3</b>
Convenience Functions .....	3
Synchronous Symmetric Cipher API .....	3
Asynchronous Symmetric Cipher API .....	3
AEAD Cipher API .....	4
Aynchronous AEAD Cipher API .....	4
AEAD Memory Structure .....	4
Message Digest API .....	6
Asymmetric Cipher API .....	6
Zero Copy .....	6
Memory Allocation .....	6
Asynchronous I/O Use Cases and Libkcapi .....	6
Multiple Staged Cipher Operations .....	7
Multiple Separate Cipher Operations .....	7
Kernel Interfaces .....	8
Kernel Configuration .....	8
Example Code .....	9
<b>3. Programming Interface .....</b>	<b>11</b>
Common API .....	11
kcapi_set_verbosity .....	11
kcapi_versionstring .....	11
kcapi_version .....	12
kcapi_pad_iv .....	13
kcapi_memset_secure .....	13
kcapi_handle_reinit .....	14
Symmetric Cipher API - Generic .....	15
kcapi_cipher_init .....	15
kcapi_cipher_destroy .....	16
kcapi_cipher_setkey .....	16
kcapi_cipher_ivsize .....	17
kcapi_cipher_blocksize .....	18
Synchronous Symmetric Cipher API - One Shot .....	18
kcapi_cipher_encrypt .....	18
kcapi_cipher_decrypt .....	19
Symmetric Cipher API - Convenience .....	20
kcapi_cipher_enc_aes_cbc .....	21
kcapi_cipher_dec_aes_cbc .....	22
kcapi_cipher_enc_aes_ctr .....	23
kcapi_cipher_dec_aes_ctr .....	24
Asynchronous Symmetric Cipher API - One Shot .....	25
kcapi_cipher_encrypt_aio .....	25
kcapi_cipher_decrypt_aio .....	26
Synchronous Symmetric Cipher API - Stream .....	27
kcapi_cipher_stream_init_enc .....	27
kcapi_cipher_stream_init_dec .....	28
kcapi_cipher_stream_update .....	29
kcapi_cipher_stream_update_last .....	30
kcapi_cipher_stream_op .....	31
AEAD Cipher API - Generic .....	32
kcapi_aead_init .....	33
kcapi_aead_destroy .....	33
kcapi_aead_setkey .....	34
kcapi_aead_setassoclen .....	35
kcapi_aead_settaglen .....	35
kcapi_aead_ivsize .....	36
kcapi_aead_blocksize .....	36
kcapi_aead_authsize .....	37
kcapi_aead_inbuflen_enc .....	37

kcapi_aead_inbuflen_dec .....	38
kcapi_aead_outbuflen_enc .....	39
kcapi_aead_outbuflen_dec .....	40
kcapi_aead_ccm_nonce_to_iv .....	40
kcapi_aead_getdata_input .....	41
kcapi_aead_getdata_output .....	42
Synchronous AEAD Cipher API - One Shot .....	43
kcapi_aead_encrypt .....	43
kcapi_aead_decrypt .....	45
Asynchronous AEAD Cipher API - One Shot .....	46
kcapi_aead_encrypt_aio .....	46
kcapi_aead_decrypt_aio .....	47
Synchronous AEAD Cipher API - Stream .....	48
kcapi_aead_stream_init_enc .....	48
kcapi_aead_stream_init_dec .....	49
kcapi_aead_stream_update .....	50
kcapi_aead_stream_update_last .....	52
kcapi_aead_stream_op .....	53
Message Digest Cipher API - Generic .....	53
kcapi_md_init .....	53
kcapi_md_destroy .....	54
kcapi_md_setkey .....	55
kcapi_md_digestsize .....	55
Message Digest Cipher API - One Shot .....	56
kcapi_md_digest .....	56
Message Digest Cipher API - Convenience .....	57
kcapi_md_sha1 .....	57
kcapi_md_sha224 .....	58
kcapi_md_sha256 .....	59
kcapi_md_sha384 .....	59
kcapi_md_sha512 .....	60
kcapi_md_hmac_sha1 .....	61
kcapi_md_hmac_sha224 .....	62
kcapi_md_hmac_sha256 .....	63
kcapi_md_hmac_sha384 .....	64
kcapi_md_hmac_sha512 .....	65
Message Digest Cipher API - Stream .....	65
kcapi_md_update .....	66
kcapi_md_final .....	66
Random Number API .....	67
kcapi_rng_init .....	67
kcapi_rng_destroy .....	68
kcapi_rng_seed .....	68
kcapi_rng_generate .....	69
kcapi_rng_seedsize .....	70
Random Number API - Convenience .....	70
kcapi_rng_get_bytes .....	70
Asymmetric Cipher API - Generic .....	71
kcapi_akcipher_init .....	71
kcapi_akcipher_destroy .....	72
kcapi_akcipher_setkey .....	72
kcapi_akcipher_setpubkey .....	73
Synchronous asymmetric Cipher API - One Shot .....	74
kcapi_akcipher_encrypt .....	74
kcapi_akcipher_decrypt .....	75
kcapi_akcipher_sign .....	76
kcapi_akcipher_verify .....	77
Asynchronous asymmetric Cipher API - One Shot .....	78
kcapi_akcipher_encrypt_aio .....	78
kcapi_akcipher_decrypt_aio .....	79
kcapi_akcipher_sign_aio .....	80
kcapi_akcipher_verify_aio .....	81
Asymmetric Cipher API - Stream .....	82
kcapi_akcipher_stream_init_enc .....	82

kcapi_akcipher_stream_init_dec .....	83
kcapi_akcipher_stream_init_sgn .....	84
kcapi_akcipher_stream_init_vfy .....	84
kcapi_akcipher_stream_update .....	85
kcapi_akcipher_stream_op .....	86
Key Protocol Primitives API - Generic .....	87
kcapi_kpp_init .....	87
kcapi_kpp_destroy .....	88
kcapi_kpp_dh_setparam_pkcs3 .....	88
kcapi_kpp_ecdh_setcurve .....	89
kcapi_kpp_setkey .....	90
Synchronous Key Protocol Primitives API - One Shot .....	91
kcapi_kpp_keygen .....	91
kcapi_kpp_ssgen .....	92
Asynchronous Key Protocol Primitives API - One Shot .....	93
kcapi_kpp_keygen_aio .....	93
kcapi_kpp_ssgen_aio .....	94
Key Derivation Functions .....	95
kcapi_kdf_dpi .....	95
kcapi_kdf_fb .....	96
kcapi_kdf_ctr .....	97
kcapi_pbkdf .....	98
kcapi_pbkdf_iteration_count .....	99
kcapi_hkdf .....	99



# Chapter 1. libkcapi - Linux Kernel Crypto API User Space Interface Library

This documentation applies to version 1.2.0.

The Linux kernel exports a network interface of type AF\_ALG to allow user space to utilize the kernel crypto API.

libkcapi uses this network interface and exports easy to use APIs so that a developer does not need to consider the low-level network interface handling.

The library does not implement any cipher algorithms. All consumer requests are sent to the kernel for processing. Results from the kernel crypto API are returned to the consumer via the library API.

The kernel interface and therefore this library can be used by unprivileged processes.

## Version Number Schema

The version numbers for this library have the following schema: MAJOR.MINOR.PATCHLEVEL

Changes in the major number implies API and ABI incompatible changes, or functional changes that require consumer to be updated (as long as this number is zero, the API is not considered stable and can change without a bump of the major version).

Changes in the minor version are API compatible, but the ABI may change. Functional enhancements only are added. Thus, a consumer can be left unchanged if enhancements are not considered. The consumer only needs to be recompiled.

Patchlevel changes are API / ABI compatible. No functional changes, no enhancements are made. This release is a bug fix release only. The consumer can be left unchanged and does not need to be recompiled.

## Purpose Of AF\_ALG

With the presence of numerous user space cryptographic libraries, one may ask why is there a need for the kernel to expose its kernel crypto API to user space. As there are system calls and potentially memory copies needed before a cipher can be invoked, it should be typically slower than user space shared libraries.

There are several reasons for AF\_ALG:

- The first and most important item is the access to hardware accelerators and hardware devices whose technical interface can only be accessed from the kernel mode / supervisor state of the processor. Such support cannot be used from user space except through AF\_ALG.
- When using user space libraries, all key material and other cryptographic sensitive parameters remains in the calling application's memory even when the application supplied the information to the library. When using AF\_ALG, the key material and other sensitive parameters are handed to the kernel. The calling application now can reliably erase that information from its memory and just use the cipher handle to perform the cryptographic operations. If the application is cracked an attacker cannot obtain the key material.
- On memory constrained systems like embedded systems, the additional memory footprint of a user space cryptographic library may be too much. As the kernel requires the kernel crypto API to be present, reusing existing code should reduce the memory footprint.





## Chapter 2. Programming Guidelines

A consumer has to use the `kcapi.h` header file to link with `libkcapi`. The linking has to be performed using `-lkcap`.

In case a consumer does not want a shared library, the `libkcapi` C file and header file can also just be copied to the consumer code and compiled along with it.

A general requirement must be observed: setting of keys must be performed before any operation. Re-setting of keys is only permissible once all data in flight (sent to the kernel but the kernel's result is not yet obtained) is processed, i.e. no data is in flight any more.

### Convenience Functions

To support various use cases, the API provided with `libkcapi` is extensive. Though, some developers want to simply use a given cipher without any specific details. To accommodate such users, `libkcapi` provides convenience functions or convenience wrappers.

The convenience functions provide exactly one function call to perform one complete cipher operation, such as an AES CBC encryption operation or a SHA-256 hashing. The caller only needs to provide the input and output buffers of his data. The entire intrinsic operation of `libkcapi` is hidden from the user.

Convenience functions are provided for the different cipher types. They are clearly marked in the API specification below.

### Synchronous Symmetric Cipher API

Symmetric ciphers can be used in the following different ways:

- One-shot API: The one-shot API performs an encryption operation with one API call. With that API call, the caller provides the input data and immediately receives the output from the cipher operation.
- Stream API: With the stream API, the caller can implement independent calls to send data to the kernel and receive data from the kernel. Multiple send calls can be inter-mixed with multiple receive calls. If the kernel buffer is full, the caller of a subsequent send call will be put to sleep. Conversely, if the buffer is empty, a caller trying to read data resulting from a cipher operation will be put to sleep. Sleeping callers will be woken up by the kernel once buffer space becomes available or data becomes available, respectively. The detached nature of the sending/receiving operation allows the implementation of multi-threaded applications where one or more threads send data and one or more threads receive data. The threads must operate on the same cipher handle. However, access to that cipher handle does not need to be serialized when the stream API calls are invoked as the API calls only read the cipher handle.

### Asynchronous Symmetric Cipher API

In addition to the symmetric cipher API, an asynchronous API is offered with the `kcapi_cipher_*_aio` and `kcapi_aead_*_aio` API calls. The concept of that API is to perform parallel operations of multiple encryption or decryption data streams.

To use the AIO API, the caller must use the `KCAPI_INIT_AIO` with the `kcapi_cipher_init` function call to set up all additional logistics for handling AIO. That means, users which are not interested in AIO will not suffer from the additional memory overhead including the time required to allocate that memory required for AIO.

This implies that the asynchronous API handles the scatter-gather lists referenced by the IOVECs differently compared to the synchronous APIs. Whereas the synchronous API references different parts of plaintext or ciphertext that are processed with one cipher operation, the IOVECs of the asynchronous API references plaintext or ciphertext where each IOVEC is processed with an independent cipher operation. I.e. when using AES-CBC with the synchronous API and the scatter-gather lists, all input data is sent to one invocation of the AES-CBC cipher. Conversely, the asynchronous API invokes one individual AES-CBC operation for each individual IOVEC.

The asynchronous API is designed to perform an in-place operation where the buffers for the input data are used to store the output data.

The asynchronous API in libkcapi as well as the kernel has a higher overhead for setting the cipher operation up. That means that if the caller only uses a one IOVEC with one associated cipher operation, the asynchronous API is expected to be slower compared to the synchronous API. But already with two or three combined cipher operations, the AIO API should be faster than the synchronous API. You may test the difference in performance with the test/kcapi test application by using the options -f for measuring the time of cipher operations in nanoseconds, -d for the number of parallel invocations and -x 1 for a symmetric one-shot cipher invocation and -x 9 for an asymmetric cipher operation with the given input data.

The kernel offers the AIO interface since kernel version 4.1 (symmetric ciphers) and 4.7 (AEAD ciphers). The libkcapi implements a transparent fallback to use the synchronous cipher API in case the AIO support is not present for the current kernel. This allows the calling users to be agnostic of the kernel support. Nonetheless, libkcapi will report the lack of AIO support if AIO is requested as the fallback implementation has a slight performance overhead.

## AEAD Cipher API

AEAD ciphers implement a very similar API approach as the symmetric ciphers:

- One-shot API: The one-shot API performs an encryption operation with one API call. With that API call, the caller provides the input data and immediately receives the output from the cipher operation.
- Stream API: With the stream API, the caller can implement independent calls to send data to the kernel and receive data from the kernel. However, unlike the symmetric cipher API, one AEAD cipher operation must be considered as one unit as the integrity value is calculated for one encryption or decryption operation. The caller can use multiple calls to provide the input data. The last chunk of data must be sent to the kernel with the API call marking the last submission. Then, the cipher operation can be triggered with the `recvmsg` invocation. It is possible to implement a multi-threaded application as the thread triggering the cipher operation is put to sleep until the last block is received. Once the last block is received, the caller waiting on the cipher operation is woken up to obtain the data.

## Asynchronous AEAD Cipher API

Similarly to the symmetric cipher API, the AEAD API supports asynchronous operation as well. The same concept regarding the IOVECs applies as discussed for the asynchronous symmetric cipher API above.

## AEAD Memory Structure

When using the stream API for AEAD, the caller must observe a particular order of data components. It is permissible that for each of the following data components multiple send calls are used. But in total, all send calls must send the AEAD data in the requested sequence. That sequence has changed with kernel 4.9. The following sequence is applicable to kernel versions up to and including 4.8:

1. Associated Authentication Data: The AAD must be provided as a first chunk.
2. Plaintext / Ciphertext: Following the AAD, the entire plaintext or ciphertext is provided that shall be encrypted and integrity protected or decrypted and whose integrity shall be verified.
3. Authentication Tag: Regardless of an encryption or decryption, the authentication tag memory must be provided.

The caller must provide memory that is identical in size for the input and output data, even parts of the memory is unused. For example, for encryption, the AEAD cipher operation only needs the AAD and the plaintext. Nonetheless, the interface requires that the memory is big enough to hold the tag as well. This requirement particularly aids the in-place cipher operation.

Starting with kernel 4.9, the interface changed slightly such that the authentication tag memory is only needed in the output buffer for encryption and in the input buffer for decryption.

To allow the calling application to be agnostic about the differences in the kernel interface, the calling application is offered additional API calls which should be used as follows:

1. Obtain the required input buffer length for the cryptographic operation using the calls `kcapi_aead_inbuflen_enc` or `kcapi_aead_inbuflen_dec`.
2. Obtain the required output buffer length for the cryptographic operation using the APIs of `kcapi_aead_outbuflen_enc` or `kcapi_aead_outbuflen_dec`.
3. For an in-place operation with a linear buffer, do the following (for an example, see `test/kcapi-main.c:cavs_aead()`):
  - a. allocate memory that is `max(inbuflen, outbuflen)`,
  - b. call to `kcapi_aead_getdata_input` and `kcapi_aead_getdata_output` with the allocated memory pointer to obtain the pointers into that allocated memory where the AAD, plaintext / ciphertext and tag is to be provided,
  - c. fill these AAD, plaintext/ciphertext and tag pointers with the respective data if they are non-NULL -- note, a NULL pointer may be returned for the tag pointer,
  - d. invoke the crypto operation with the pointer to the allocated buffer and `inbuflen` is supplied to the.
4. For separate, potentially non-contiguous buffers, do the following (for an example, see `test/kcapi-main.c:cavs_aead_stream()`):
  - a. ensure that your total buffer size for input and output complies with the result from the buffer lengths supplied by the aforementioned API calls,
  - b. call to `kcapi_aead_getdata_input` and `kcapi_aead_getdata_output` with NULL pointers for the memory buffers to obtain the lengths for the AEAD data components,
  - c. initialize the IOVECs and/or invoke the stream API with the independent buffers with the AAD, plaintext/ciphertext and tag if the associated length values are non-zero.

If the caller chooses to not implement an in-place operation, the kernel will copy the AAD data into the output buffer, so that the destination buffer will hold the ciphertext or plaintext, the AAD data and the authentication tag (encryption only). The memory structure of the destination buffer is identical to the source buffer. (This is currently not yet implemented for all ciphers and will be fixed in future kernel versions.)

## Message Digest API

Again, like for the symmetric ciphers, the message digest API implements the one-shot and the stream use cases. In addition, convenience wrapper functions for SHA-1 through SHA-512 are provided where the caller only provides its input data and the return buffer for obtaining a message digest or keyed message digest.

## Asymmetric Cipher API

The asymmetric cipher API provides access to the raw asymmetric operations (i.e. modular exponentiation).

## Zero Copy

When using the one-shot API for symmetric ciphers, AEAD ciphers, as well as message digests, the library uses the zero copy interface to provide the input data to the kernel. That means, the kernel operates on the user space pages.

To ensure the efficiency of this zero copy approach, the caller should use a page-aligned data buffer for the input data. Non-aligned buffers would work also, but the kernel would need to perform more page accesses, lowering the throughput. Such an aligned buffer can be created, for example, using the following call - the value 4096 should be the size of one page on the system:

```
unsigned char buf[4096] __attribute__((__aligned__(4096)));

unsigned char *buf;
posix_memalign((void *)&buf, PAGE_SIZE, buflen);
```

## Memory Allocation

The library libkcapi uses the data structure `struct kcapi_handle` as the cipher handle that allows the consumer to operate with the various function calls of this library.

Unlike other crypto libraries, libkcapi does not allocate any memory or performs operations that implies memory allocation. `struct kcapi_handle` only holds pointers to the consumer-provided buffers with sensitive data. That means that the buffers holding sensitive data like keys are under full control of the consumer. Therefore, this library does not offer any memory allocation or secure memory clearing functions.

The consumer must ensure that the memory is appropriately sanitized. The caller does not need to sanitize `struct kcapi_handle` as it does not contain any sensitive data.

## Asynchronous I/O Use Cases and Libkcapi

The kernel crypto API user space interface supports different use cases with the asynchronous I/O operations which are illustrated in the following sections. These sections also illustrate the API calls to be used to follow the respective use cases.

All APIs that perform synchronous operation do not have different purposes and thus do not require special precautions when using them.

The different use cases round asynchronous I/O revolve around different ways how to send data to the kernel and to retrieve processed data.

### Multiple Staged Cipher Operations

Using the `kcapi_cipher_*_aio` and `kcapi_aead_*_aio` API calls, a caller can supply one or more IOVECs of data to the kernel. However, the caller can only supply one IV to the kernel.

The API calls only allow specifying one integer defining the number of IOVECs in the arrays of the input data as well as the output data. The libkcapi library uses the input and output IOVECs as pairs. I.e. the first IOVEC of the input array relate to the first IOVEC of the output array, and so on.

The kernel invokes the cipher operation when a `recvmsg` system call is processed. The AIO handling transforms each output IOVEC into one separate invocation of the `recvmsg` handler that processes the data submitted with the corresponding input IOVEC. This means, each output IOVEC will trigger one cipher operation. When multiple IOVECs are processed by the kernel's AIO handling, all resulting `recvmsg` calls are invoked with the in the data same order specified by the list of IOVECs.

WARNING: Currently, it is not guaranteed that the drivers perform the proper serialization of the parallel processing of the different IOVECs. For example, when providing two IOVECs, they may be both using the initially set IV. Thus, they are not chained. User space is able to serialize the AIO operation in this case by invoking the AIO API calls with input/output IOVEC arrays holding one entry each only. However, this would imply that this type of invocation will not be different from a synchronous invocation.

### Multiple Separate Cipher Operations

The kernel and thus libkcapi supports the use case where several of the aforementioned multiple staged cipher operations are can be performed in parallel which are totally isolated from each other. In this case, different IVs are used.

Using the `kcapi_handle_reinit` libkcapi API call, the caller can obtain a new cipher handle from an existing handle. Both share the same key and cipher -- the kernel crypto API maintains the same TFM data structure for both. However, both cipher handles can now encrypt or decrypt data completely isolated from each other. Specifically, the following data of a cipher operation is isolated between the different cipher handles -- this is all data that is not set with a `setsockopt(2)` system call:

- Input: plaintext (encryption) or ciphertext (decryption)
- Output: plaintext (decryption) or ciphertext (encryption)
- IV
- AEAD: associated authenticated data (AAD) and its length
- AEAD: tag

The following data is shared between the different cipher handles -- this covers all data that can be set with a `setsockopt(2)` system call:

- Key

- AEAD: Tag length

This means that the "multiple staged cipher operations" discussed above can be performed with each cipher handle independently.

The call `kcapi_handle_reinit` does not open another socket, but implies that only a new `accept(2)` system call is performed.

## Kernel Interfaces

Depending on the version of your kernel, some of the kernel interfaces the library depends on are not available. When using the respective library API functions, an error is returned during initialization of the cipher handle. The following interfaces are available:

- `kcapi_md_*` usable since kernel version 3.0
- `kcapi_cipher_*` usable since kernel version 3.0
- `kcapi_rng_*` kernel interface integrated into kernel version 4.0
- `kcapi_aead_*` kernel interface added to cryptodev-2.6 tree and should be usable with kernel version 4.2.
- `kcapi_akcipher_*` kernel interface is discussed for inclusion to the cryptodev-2.6 kernel tree.

## Kernel Configuration

To use `libkcapi`, the following kernel options need to be enabled:

- `CONFIG_CRYPTO_USER` enables the `NETLINK_CRYPTO` interface to allow obtaining information about the loaded ciphers. When compiled as module in older kernels (pre 3.18) the resulting `crypto_user` kernel module must be loaded manually.
- `CONFIG_CRYPTO_USER_API` enables the core functionality of the user space interface handler.
- `CONFIG_CRYPTO_USER_API_HASH` enables the "hash" interface (i.e. allows the use of all message digest and keyed message digest ciphers).
- `CONFIG_CRYPTO_USER_API_SKCIPHER` enables the "skcipher" interface to use symmetric cipher algorithms.
- `CONFIG_CRYPTO_USER_API_AEAD` enables the "aead" interface to use AEAD cipher algorithms. This support is currently discussed on LKML and therefore not present in the mainline kernel.
- `CONFIG_CRYPTO_USER_API_RNG` enables the "rng" interface to use the random number generators.
- `CONFIG_CRYPTO_USER_API_AKCIPHER` enables the "akcipher" interface to use the asymmetric ciphers. This support is currently discussed on LKML and therefore not present in the mainline kernel.

In addition, the following patch must be applied if a kernel less than 3.19-rc1 or the cryptodev-2.6 kernel tree is used:  
<https://git.kernel.org/cgit/linux/kernel/git/herbert/cryptodev-2.6.git/commit/?id=5d4a5e770d97d87082067886e7097c920b338da5>

In addition, the following patch must be applied if a kernel less than 3.19-rc1 or the cryptodev-2.6 kernel tree is used:  
<https://git.kernel.org/cgit/linux/kernel/git/herbert/cryptodev-2.6.git/commit/?id=af8e80731a94ff9de9508b01d9e5d931d538dc6b>

In addition, the following patch must be applied if a kernel less than 3.19-rc1 or the cryptodev-2.6 kernel tree is used:

<https://git.kernel.org/cgit/linux/kernel/git/herbert/cryptodev-2.6.git/commit/?id=25fb8638e919bc7431a73f2fb4a9713818ae2c9d>

## **Example Code**

Example code covering all available API calls is provided with the test code in the test/ directory.





## Chapter 3. Programming Interface

### Common API

The following API calls are common to all cipher types.

#### **kcapi\_set\_verbosity**

**LINUX**

libkcapi Manual December 2020

##### **Name**

`kcapi_set_verbosity` — set the verbosity level of the library

##### **Synopsis**

```
void kcapi_set_verbosity (enum kcapi_verbosity level);
```

##### **Arguments**

*level*

[in] verbosity level:

##### **LOG\_ERR**

only log error messages (default)

##### **LOG\_WARN**

log warnings and error messages

##### **LOG\_VERBOSE**

log verbose messages, warnings and error messages

##### **LOG\_DEBUG**

log all details of library operation

#### **kcapi\_versionstring**

**LINUX**

## Name

`kcapi_versionstring` — obtain version string of kcapi library

## Synopsis

```
void kcapi_versionstring (char * buf, uint32_t buflen);
```

## Arguments

*buf*

[out] buffer to place version string into

*buflen*

[in] length of buffer

## kcapi\_version

### LINUX

## Name

`kcapi_version` — return machine-usable version number of kcapi library

## Synopsis

```
uint32_t kcapi_version ( void );
```

## Arguments

*void*

no arguments

## Description

The function returns a version number that is monotonic increasing for newer versions. The version numbers are multiples of 100. For example, version 1.2.3 is converted to 1020300 -- the last two digits are reserved for future use.

The result of this function can be used in comparing the version number in a calling program if version-specific calls need to be made.

*return* Version number of kcapi library

## kcapi\_pad\_iv

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_pad\_iv — realign the IV as necessary for cipher

#### Synopsis

```
int kcapi_pad_iv (struct kcapi_handle * handle, const uint8_t * iv,
uint32_t ivlen, uint8_t ** newiv, uint32_t * newivlen);
```

#### Arguments

*handle*

[in] cipher handle

*iv*

[in] current IV buffer

*ivlen*

[in] length of IV buffer

*newiv*

[out] buffer of aligned IV

*newivlen*

[out] length of newly aligned IV

#### Description

The function pads the least significant bits of the provided IV up to the block size of the cipher with zeros. In case the provided IV is longer than the block size, the least significant bits are truncated to the block size.

The function allocates memory for newiv in case the return code indicates success. The consumer must free the memory after use.

*return* 0 for success; a negative errno-style error code if an error occurred

## kcapi\_memset\_secure

### LINUX

## Name

`kcapi_memset_secure` — `memset` implementation that will not be optimized away by the compiler

## Synopsis

```
void kcapi_memset_secure (void * s, int c, uint32_t n);
```

## Arguments

*s*

[in] see `memset(3)`

*c*

[in] see `memset(3)`

*n*

[in] see `memset(3)`

## Description

The parameters, he logic and the return code is identical to `memset(3)`.

## kcapi\_handle\_reinit

### LINUX

## Name

`kcapi_handle_reinit` — re-initialize a new kernel interface

## Synopsis

```
int kcapi_handle_reinit (struct kcapi_handle ** newhandle, struct  
kcapi_handle * existing, uint32_t flags);
```

## Arguments

*newhandle*

[out] cipher handle filled during the call

*existing*

[in] existing cipher handle from which a new handle shall be re-initialized

*flags*

[in] flags specifying the type of cipher handle

## Description

The kernel crypto API interface operates with two types of file descriptors, the TFM file descriptor and the OP file descriptor.

The TFM file descriptor receives the cipher-operation static information: the key, and the AEAD tag size.

The OP file descriptor receives the volatile data, such as the plaintext / ciphertext, the IV, or the AEAD AD size.

The kernel crypto API AF\_ALG interface supports the concept that one TFM file descriptor can operate with multiple OP file descriptors. The different OP file descriptors can perform completely separate cipher operations using the same key which can execute in parallel. The parallel execution can be performed in the same or different process threads.

`kcapi_handle_reinit` function allows the allocation of a new cipher handle with a new OP file descriptor but using the same TFM file descriptor. To obtain a reference to the TFM file descriptor, an *existing* cipher handle is used as source. `kcapi_handle_reinit` can be invoked multiple times. Each resulting cipher handle must be deallocated with `kcapi_cipher_destroy`. The deallocation ensures that the TFM resource is only released if the last handle using this TFM resource is released.

*return* 0 upon success; -EINVAL - accept syscall failed -ENOMEM - cipher handle cannot be allocated

## Symmetric Cipher API - Generic

These generic API for symmetric ciphers calls are to be used for both, the one-shot and the stream encryption/decryption operations.

API function calls used to invoke symmetric ciphers.

## kcapi\_cipher\_init

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_cipher_init` — initialize cipher handle

### Synopsis

```
int kcapi_cipher_init (struct kcapi_handle ** handle, const char *
ciphername, uint32_t flags);
```

## Arguments

*handle*

[out] cipher handle filled during the call

*ciphername*

[in] kernel crypto API cipher name as specified in /proc/crypto

*flags*

[in] flags specifying the type of cipher handle

## Description

This function provides the initialization of a symmetric cipher handle and establishes the connection to the kernel.

On success, a pointer to `kcapi_handle` object is returned in `*handle`. Function `kcapi_cipher_destroy` should be called afterwards to free resources.

*return* 0 upon success; -ENOENT - algorithm not available; -EOPNOTSUPP - AF\_ALG family not available; -EINVAL - accept syscall failed -ENOMEM - cipher handle cannot be allocated

## kcapi\_cipher\_destroy

**LINUX**

libkcapi Manual December 2020

### Name

`kcapi_cipher_destroy` — close the cipher handle and release resources

### Synopsis

```
void kcapi_cipher_destroy (struct kcapi_handle * handle);
```

## Arguments

*handle*

[in] cipher handle to release

## kcapi\_cipher\_setkey

**LINUX**

libkcapi ManualDecember 2020

**Name**`kcapi_cipher_setkey` — set the key for the cipher handle**Synopsis**

```
int kcapi_cipher_setkey (struct kcapi_handle * handle, const uint8_t
* key, uint32_t keylen);
```

**Arguments***handle*

[in] cipher handle

*key*

[in] key buffer

*keylen*

[in] length of key buffer

**Description**

With this function, the caller sets the key for subsequent encryption or decryption operations.

After the caller provided the key, the caller may securely destroy the key as it is now maintained by the kernel.

*return* 0 upon success (in case of an akcipher handle, a positive integer is returned that denominates the maximum output size of the cryptographic operation -- this value must be used as the size of the output buffer for one cryptographic operation); a negative errno-style error code if an error occurred

**kcapi\_cipher\_ivsize****LINUX**

libkcapi ManualDecember 2020

**Name**`kcapi_cipher_ivsize` — return size of IV required for cipher**Synopsis**

```
uint32_t kcapi_cipher_ivsize (struct kcapi_handle * handle);
```

## Arguments

*handle*  
[in] cipher handle

## Description

*return* > 0 specifying the IV size; 0 on error

## kcapi\_cipher\_blocksize

### LINUX

libkcapi Manual December 2020

## Name

kcapi\_cipher\_blocksize — return size of one block of the cipher

## Synopsis

```
uint32_t kcapi_cipher_blocksize (struct kcapi_handle * handle);
```

## Arguments

*handle*  
[in] cipher handle

## Description

*return* > 0 specifying the block size; 0 on error

## Synchronous Symmetric Cipher API - One Shot

## kcapi\_cipher\_encrypt

### LINUX

libkcapi Manual December 2020

## Name

kcapi\_cipher\_encrypt — encrypt data (synchronous one shot)



## Synopsis

```
int32_t kcap_i_cipher_encrypt (struct kcap_i_handle * handle, const
uint8_t * in, uint32_t inlen, const uint8_t * iv, uint8_t * out,
uint32_t outlen, int access);
```

## Arguments

*handle*

[in] cipher handle

*in*

[in] plaintext data buffer

*inlen*

[in] length of in buffer

*iv*

[in] IV to be used for cipher operation

*out*

[out] ciphertext data buffer

*outlen*

[in] length of out buffer

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

It is perfectly legal to use the same buffer as the plaintext and ciphertext pointers. That would mean that after the encryption operation, the plaintext is overwritten with the ciphertext.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

The IV buffer must be exactly `kcap_i_cipher_ivsize` bytes in size.

*return* number of bytes encrypted upon success; a negative errno-style error code if an error occurred

## kcap\_i\_cipher\_decrypt

**LINUX**

## Name

`kcapi_cipher_decrypt` — decrypt data (synchronous one shot)

## Synopsis

```
int32_t kcapi_cipher_decrypt (struct kcapi_handle * handle, const
uint8_t * in, uint32_t inlen, const uint8_t * iv, uint8_t * out,
uint32_t outlen, int access);
```

## Arguments

*handle*

[in] cipher handle

*in*

[in] ciphertext data buffer

*inlen*

[in] length of in buffer

*iv*

[in] IV to be used for cipher operation

*out*

[out] plaintext data buffer

*outlen*

[in] length of out bufferS

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

It is perfectly legal to use the same buffer as the plaintext and ciphertext pointers. That would mean that after the encryption operation, the ciphertext is overwritten with the plaintext.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

The IV buffer must be exactly `kcapi_cipher_ivsize` bytes in size.

*return* number of bytes decrypted upon success; a negative errno-style error code if an error occurred

## Symmetric Cipher API - Convenience

### kcapi\_cipher\_enc\_aes\_cbc

#### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_cipher\_enc\_aes\_cbc — Convenience function for AES CBC encryption

#### Synopsis

```
int32_t kcapi_cipher_enc_aes_cbc (const uint8_t * key, uint32_t
keylen, const uint8_t * in, uint32_t inlen, const uint8_t * iv,
uint8_t * out, uint32_t outlen);
```

#### Arguments

*key*

[in] key buffer

*keylen*

[in] length of key buffer

*in*

[in] plaintext data buffer

*inlen*

[in] length of in buffer

*iv*

[in] IV to be used for cipher operation

*out*

[out] ciphertext data buffer

*outlen*

[in] length of out buffer

#### Description

The convenience function performs an AES CBC encryption operation using the provided key, the given input buffer and the given IV. The output is stored in the out buffer.

Note, AES CBC requires an input data that is a multiple of 16 bytes. If you have data that is not guaranteed to be multiples of 16 bytes, either add zero bytes at the end of the buffer to pad it up to a multiple of 16 bytes. Otherwise, the CTR mode encryption operation may be usable.

The output buffer must be at least as large as the input buffer.

The IV must be exactly 16 bytes in size.

The AES type (AES-128, AES-192 or AES-256) is determined by the size of the given key. If the key is 16 bytes long, AES-128 is used. A 24 byte key implies AES-192 and a 32 byte key implies AES-256.

*return* number of bytes generated upon success; a negative errno-style error code if an error occurred

## kcapi\_cipher\_dec\_aes\_cbc

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_cipher\_dec\_aes\_cbc — Convenience function for AES CBC decryption

#### Synopsis

```
int32_t kcapi_cipher_dec_aes_cbc (const uint8_t * key, uint32_t
keylen, const uint8_t * in, uint32_t inlen, const uint8_t * iv,
uint8_t * out, uint32_t outlen);
```

#### Arguments

*key*

[in] key buffer

*keylen*

[in] length of key buffer

*in*

[in] ciphertext data buffer

*inlen*

[in] length of in buffer

*iv*

[in] IV to be used for cipher operation

*out*

[out] plaintext data buffer

*outlen*

[in] length of out buffer

## Description

The convenience function performs an AES CBC decryption operation using the provided key, the given input buffer and the given IV. The output is stored in the out buffer.

Note, AES CBC requires an input data that is a multiple of 16 bytes. If you have data that is not guaranteed to be multiples of 16 bytes, either add zero bytes at the end of the buffer to pad it up to a multiple of 16 bytes. Otherwise, the CTR mode encryption operation may be usable.

The output buffer must be at least as large as the input buffer.

The IV must be exactly 16 bytes in size.

The AES type (AES-128, AES-192 or AES-256) is determined by the size of the given key. If the key is 16 bytes long, AES-128 is used. A 24 byte key implies AES-192 and a 32 byte key implies AES-256.

*return* number of bytes generated upon success; a negative errno-style error code if an error occurred

## kcapi\_cipher\_enc\_aes\_ctr

### LINUX

libkcapi Manual December 2020

### Name

kcapi\_cipher\_enc\_aes\_ctr — Convenience function for AES CTR encryption

### Synopsis

```
int32_t kcapi_cipher_enc_aes_ctr (const uint8_t * key, uint32_t
keylen, const uint8_t * in, uint32_t inlen, const uint8_t * ctr,
uint8_t * out, uint32_t outlen);
```

### Arguments

*key*

[in] key buffer

*keylen*

[in] length of key buffer

*in*

[in] plaintext data buffer

*inlen*

[in] length of in buffer

*ctr*

[in] start counter value to be used for cipher operation

*out*

[out] ciphertext data buffer

*outlen*

[in] length of out buffer

## Description

The convenience function performs an AES counter mode encryption operation using the provided key, the given input buffer and the given IV. The output is stored in the out buffer.

The input buffer can be of arbitrary length.

The output buffer must be at least as large as the input buffer.

The start counter can contain all zeros (not a NULL buffer!) and must be exactly 16 bytes in size.

The AES type (AES-128, AES-192 or AES-256) is determined by the size of the given key. If the key is 16 bytes long, AES-128 is used. A 24 byte key implies AES-192 and a 32 byte key implies AES-256.

*return* number of bytes generated upon success; a negative errno-style error code if an error occurred

## kcapi\_cipher\_dec\_aes\_ctr

### LINUX

libkcapi Manual December 2020

### Name

kcapi\_cipher\_dec\_aes\_ctr — Convenience function for AES CTR decryption

### Synopsis

```
int32_t kcapi_cipher_dec_aes_ctr (const uint8_t * key, uint32_t
keylen, const uint8_t * in, uint32_t inlen, const uint8_t * ctr,
uint8_t * out, uint32_t outlen);
```

### Arguments

*key*

[in] key buffer

*keylen*

[in] length of key buffer

*in*

[in] ciphertext data buffer

*inlen*

[in] length of in buffer

*ctr*

[in] start counter value to be used for cipher operation

*out*

[out] plaintext data buffer

*outlen*

[in] length of out buffer

## Description

The convenience function performs an AES counter mode encryption operation using the provided key, the given input buffer and the given IV. The output is stored in the out buffer.

The input buffer can be of arbitrary length.

The output buffer must be at least as large as the input buffer.

The start counter can contain all zeros (not a NULL buffer!) and must be exactly 16 bytes in size.

The AES type (AES-128, AES-192 or AES-256) is determined by the size of the given key. If the key is 16 bytes long, AES-128 is used. A 24 byte key implies AES-192 and a 32 byte key implies AES-256.

*return* number of bytes generated upon success; a negative errno-style error code if an error occurred

## Asynchronous Symmetric Cipher API - One Shot

### kcapi\_cipher\_encrypt\_aio

#### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_cipher\_encrypt\_aio — encrypt data (asynchronous one shot)

#### Synopsis

```
int32_t kcapi_cipher_encrypt_aio (struct kcapi_handle * handle,
struct iovec * iniov, struct iovec * outiov, uint32_t iovlen, const
uint8_t * iv, int access);
```

## Arguments

*handle*

[in] cipher handle

*iniov*

[in] head of scatter-gather list array holding the plaintext

*outiov*

[out] head of scatter-gather list of the destination buffers filled with ciphertext

*iovlen*

[in] number of scatter-gather list entries

*iv*

[in] IV to be used for cipher operation

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

The individual scatter-gather list entries are processed with separate invocations of the the given cipher.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

The IV buffer must be exactly `kcapi_cipher_ivsize` bytes in size.

*return* number of bytes encrypted upon success; a negative errno-style error code if an error occurred

## kcapi\_cipher\_decrypt\_aio

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_cipher_decrypt_aio` — decrypt data (asynchronous one shot)

### Synopsis

```
int32_t kcapi_cipher_decrypt_aio (struct kcapi_handle * handle,
struct iovec * iniov, struct iovec * outiov, uint32_t iovlen, const
uint8_t * iv, int access);
```



## Arguments

*handle*

[in] cipher handle

*iniov*

[in] head of scatter-gather list array holding the ciphertext

*outiov*

[out] head of scatter-gather list with the destination buffers for the plaintext

*iovlen*

[in] number of scatter-gather list entries

*iv*

[in] IV to be used for cipher operation

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

The individual scatter-gather list entries are processed with separate invocations of the the given cipher.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

The IV buffer must be exactly `kcapi_cipher_ivsize` bytes in size.

*return* number of bytes decrypted upon success; a negative errno-style error code if an error occurred

## Synchronous Symmetric Cipher API - Stream

The stream API requires that first the cipher operation type is set with the `init` call, followed by an arbitrary number and mix of the `update` and `op` calls.

### `kcapi_cipher_stream_init_enc`

#### LINUX

libkcapi Manual December 2020

#### Name

`kcapi_cipher_stream_init_enc` — start an encryption operation (stream)

## Synopsis

```
int32_t kcap_i_cipher_stream_init_enc (struct kcap_i_handle * handle,  
const uint8_t * iv, struct iovec * iov, uint32_t iovlen);
```

## Arguments

*handle*

[in] cipher handle

*iv*

[in] IV to be used for cipher operation

*iov*

[in] scatter/gather list with data to be encrypted. This is the pointer to the first iov entry if an array of iov entries is supplied. See `sendmsg(2)` for details on how iov is to be used. This pointer may be NULL if no data to be encrypted is available at the point of the call.

*iovlen*

[in] number of scatter/gather list elements. If iov is NULL, this value must be zero.

## Description

A stream encryption operation is started with this call. Multiple successive `kcap_i_cipher_stream_update` function calls can be invoked to send more plaintext data to be encrypted. The kernel buffers the input until `kcap_i_cipher_stream_op` picks up the encrypted data. Once plaintext is encrypted during the `kcap_i_cipher_stream_op` it is removed from the kernel buffer.

The function calls of `kcap_i_cipher_stream_update` and `kcap_i_cipher_stream_op` can be mixed, even by multiple threads of an application.

The IV buffer must be exactly `kcap_i_cipher_ivsize` bytes in size.

*return* number of bytes sent to the kernel upon success; a negative errno-style error code if an error occurred

## kcap\_i\_cipher\_stream\_init\_dec

### LINUX

libkcap\_i Manual December 2020

## Name

`kcap_i_cipher_stream_init_dec` — start a decryption operation (stream)

## Synopsis

```
int32_t kcap_i_cipher_stream_init_dec (struct kcap_i_handle * handle,
const uint8_t * iv, struct iovec * iov, uint32_t iovlen);
```

## Arguments

*handle*

[in] cipher handle

*iv*

[in] IV to be used for cipher operation

*iov*

[in] scatter/gather list with data to be encrypted. This is the pointer to the first iov entry if an array of iov entries is supplied. See `sendmsg(2)` for details on how iov is to be used. This pointer may be NULL if no data to be encrypted is available at the point of the call.

*iovlen*

[in] number of scatter/gather list elements. If iov is NULL, this value must be zero.

## Description

A stream decryption operation is started with this call. Multiple successive `kcap_i_cipher_stream_update` function calls can be invoked to send more ciphertext data to be decrypted. The kernel buffers the input until `kcap_i_cipher_stream_op` picks up the decrypted data. Once ciphertext is decrypted during the `kcap_i_cipher_stream_op` it is removed from the kernel buffer.

The function calls of `kcap_i_cipher_stream_update` and `kcap_i_cipher_stream_op` can be mixed, even by multiple threads of an application.

The IV buffer must be exactly `kcap_i_cipher_ivsize` bytes in size.

*return* number of bytes sent to the kernel upon success; a negative errno-style error code if an error occurred

## kcap\_i\_cipher\_stream\_update

**LINUX**

libkcap\_i Manual December 2020

### Name

`kcap_i_cipher_stream_update` — send more data for processing (stream)

## Synopsis

```
int32_t kcapi_cipher_stream_update (struct kcapi_handle * handle,  
struct iovec * iov, uint32_t iovlen);
```

## Arguments

*handle*

[in] cipher handle

*iov*

[in] scatter/gather list with data to be processed by the cipher operation.

*iovlen*

[in] number of scatter/gather list elements.

## Description

Using this function call, more plaintext for encryption or ciphertext for decryption can be submitted to the kernel.

This function may cause the caller to sleep if the kernel buffer holding the data is getting full. The process will be woken up once more buffer space becomes available by calling `kcapi_cipher_stream_op`.

## Note

with the separate API calls of `kcapi_cipher_stream_update` and `kcapi_cipher_stream_op` a multi-threaded application can be implemented where one thread sends data to be processed and one thread picks up data processed by the cipher operation.

## IMPORTANT NOTE

The kernel will only process `sysconf(_SC_PAGESIZE) * ALG_MAX_PAGES` at one time. If your input data is larger than this threshold, you MUST segment it into chunks of at most `sysconf(_SC_PAGESIZE) * ALG_MAX_PAGES` and invoke the `kcapi_cipher_stream_update` on that segment followed by `kcapi_cipher_stream_op` before the next chunk is processed. If this rule is not obeyed, the thread invoking `kcapi_cipher_stream_update` will be put to sleep until another thread invokes `kcapi_cipher_stream_op`.

## WARNING

The memory referenced by *iov* is not accessed by the kernel during this call. The memory is first accessed when `kcapi_cipher_stream_op` is called. Thus, you MUST make sure that the referenced memory is still present at the time `kcapi_cipher_stream_op` is called.

*return* number of bytes sent to the kernel upon success; a negative errno-style error code if an error occurred

## kcapi\_cipher\_stream\_update\_last

### LINUX

libkcapi Manual December 2020

#### Name

`kcapi_cipher_stream_update_last` — send last data for processing (stream)

#### Synopsis

```
int32_t kcapi_cipher_stream_update_last (struct kcapi_handle *
handle, struct iovec * iov, uint32_t iovlen);
```

#### Arguments

*handle*

[in] cipher handle

*iov*

[in] scatter/gather list with data to be processed by the cipher operation.

*iovlen*

[in] number of scatter/gather list elements.

#### Description

Using this function call, more plaintext for encryption or ciphertext for decryption can be submitted to the kernel.

This call is identical to the `kcapi_cipher_stream_update` call with the exception that it marks the last data buffer before the cipher operation is triggered. This is call is important for stream ciphers like CTR or CTS mode when providing the last block. It is permissible to provide a zero buffer if all data including the last block is already provided by `kcapi_cipher_stream_update`.

#### WARNING

If this call is not made for stream ciphers with input data that is not a multiple of the block size of the block cipher, the kernel will not return the last block that contains less data than the block size of the block cipher. For example, sending 257 bytes of data to be encrypted with `ctr(aes)`, the kernel will return only 256 bytes without this call.

*return* number of bytes sent to the kernel upon success; a negative errno-style error code if an error occurred

## kcapi\_cipher\_stream\_op

**LINUX**

libkcapi Manual December 2020

### Name

kcapi\_cipher\_stream\_op — obtain processed data (stream)

### Synopsis

```
int32_t kcapi_cipher_stream_op (struct kcapi_handle * handle, struct
iovec * iov, uint32_t iovlen);
```

### Arguments

*handle*

[in] cipher handle

*iov*

[out] scatter/gather list pointing to buffers to be filled with the resulting data from a cipher operation.

*iovlen*

[in] number of scatter/gather list elements.

### Description

This call can be called interleaved with `kcapi_cipher_stream_update` to fetch the processed data.

This function may cause the caller to sleep if the kernel buffer holding the data is empty. The process will be woken up once more data is sent by calling `kcapi_cipher_stream_update`.

Note, when supplying buffers that are not multiple of block size, the buffers will only be filled up to the maximum number of full block sizes that fit into the buffer.

The kernel supports multithreaded applications where one or more threads send data via the `kcapi_cipher_stream_update` function and another thread collects the processed data via `kcapi_cipher_stream_op`. The kernel, however, will return data via `kcapi_cipher_stream_op` as soon as it has some data available. For example, one thread sends 1000 bytes to be encrypted and another thread already waits for the ciphertext. The kernel may send only, say, 500 bytes back to the waiting process during one `kcapi_cipher_stream_op` call. In a subsequent calls to `kcapi_cipher_stream_op` more ciphertext is returned. This implies that when the receiving thread shall collect all data there is, `kcapi_cipher_stream_op` must be called in a loop until all data is received.

*return* number of bytes obtained from the kernel upon success; a negative `errno`-style error code if an error occurred

## AEAD Cipher API - Generic

These generic API for symmetric ciphers calls are to be used for both, the one-shot and the stream encryption/decryption operations.

The following API calls allow using the Authenticated Encryption with Associated Data.

### kcapi\_aead\_init

#### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_aead\_init — initialization of cipher handle

#### Synopsis

```
int kcapi_aead_init (struct kcapi_handle ** handle, const char *
ciphername, uint32_t flags);
```

#### Arguments

*handle*

[out] cipher handle filled during the call

*ciphername*

[in] kernel crypto API cipher name as specified in /proc/crypto

*flags*

[in] flags specifying the type of cipher handle

#### Description

This function initializes an AEAD cipher handle and establishes the connection to the kernel.

On success, a pointer to kcapi\_handle object is returned in \*handle. Function kcapi\_aead\_destroy should be called afterwards to free resources.

*return* 0 upon success; -ENOENT - algorithm not available; -EOPNOTSUPP - AF\_ALG family not available; -EINVAL - accept syscall failed -ENOMEM - cipher handle cannot be allocated

### kcapi\_aead\_destroy

#### LINUX

## Name

`kcapi_aead_destroy` — close the AEAD handle and release resources

## Synopsis

```
void kcapi_aead_destroy (struct kcapi_handle * handle);
```

## Arguments

*handle*

[in] cipher handle to release

## kcapi\_aead\_setkey

### LINUX

## Name

`kcapi_aead_setkey` — set the key for the AEAD handle

## Synopsis

```
int kcapi_aead_setkey (struct kcapi_handle * handle, const uint8_t *  
key, uint32_t keylen);
```

## Arguments

*handle*

[in] cipher handle

*key*

[in] key buffer

*keylen*

[in] length of key buffer

## Description

With this function, the caller sets the key for subsequent encryption or decryption operations.

After the caller provided the key, the caller may securely destroy the key as it is now maintained by the kernel.



*return* 0 upon success; a negative errno-style error code if an error occurred

## kcapi\_aead\_setassoclen

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_aead\_setassoclen — set authentication data size

#### Synopsis

```
void kcapi_aead_setassoclen (struct kcapi_handle * handle, uint32_t
assoclen);
```

#### Arguments

*handle*

[in] cipher handle

*assoclen*

[in] length of associated data length

#### Description

The associated data is retained in the cipher handle. During initialization of a cipher handle, it is sent to the kernel. The kernel cipher implementations may verify the appropriateness of the authentication data size and may return an error during initialization if the authentication size is not considered appropriate.

## kcapi\_aead\_settaglen

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_aead\_settaglen — set authentication tag size

#### Synopsis

```
int kcapi_aead_settaglen (struct kcapi_handle * handle, uint32_t
taglen);
```

## Arguments

*handle*

[in] cipher handle

*taglen*

[in] length of authentication tag

## Description

Set the authentication tag size needed for encryption operation. The tag is created during encryption operation with the size provided with this call.

*return* 0 upon success; a negative errno-style error code if an error occurred

## kcapi\_aead\_ivsize

### LINUX

libkcapi Manual December 2020

## Name

kcapi\_aead\_ivsize — return size of IV required for cipher

## Synopsis

```
uint32_t kcapi_aead_ivsize (struct kcapi_handle * handle);
```

## Arguments

*handle*

[in] cipher handle

## Description

*return* > 0 specifying the IV size; 0 on error

## kcapi\_aead\_blocksize

### LINUX

libkcapi Manual December 2020

## Name

kcapi\_aead\_blocksize — return size of one block of the cipher

## Synopsis

```
uint32_t kcap_i_aead_blocksize (struct kcap_i_handle * handle);
```

## Arguments

*handle*

[in] cipher handle

## Description

*return* > 0 specifying the block size; 0 on error

## kcap\_i\_aead\_authsize

### LINUX

libkcap\_i Manual December 2020

## Name

kcap\_i\_aead\_authsize — return the maximum size of the tag

## Synopsis

```
uint32_t kcap_i_aead_authsize (struct kcap_i_handle * handle);
```

## Arguments

*handle*

[in] cipher handle

## Description

The returned maximum is the largest size of the authentication tag that can be produced by the AEAD cipher. Smaller tag sizes may be chosen depending on the AEAD cipher type.

*return* > 0 specifying the block size; 0 on error

## kcapi\_aead\_inbuflen\_enc

**LINUX**

libkcapi Manual December 2020

### Name

`kcapi_aead_inbuflen_enc` — return minimum encryption input buffer length

### Synopsis

```
uint32_t kcapi_aead_inbuflen_enc (struct kcapi_handle * handle,  
uint32_t inlen, uint32_t assoclen, uint32_t taglen);
```

### Arguments

*handle*

[in] cipher handle

*inlen*

[in] size of plaintext

*assoclen*

[in] size of associated data (AD)

*taglen*

[in] size of authentication tag

### Description

*return* minimum size of input data length in bytes

## kcapi\_aead\_inbuflen\_dec

**LINUX**

libkcapi Manual December 2020

### Name

`kcapi_aead_inbuflen_dec` — return minimum decryption input buffer length

### Synopsis

```
uint32_t kcapi_aead_inbuflen_dec (struct kcapi_handle * handle,  
uint32_t inlen, uint32_t assoclen, uint32_t taglen);
```

## Arguments

*handle*

[in] cipher handle

*inlen*

[in] size of ciphertext

*assoclen*

[in] size of associated data (AD)

*taglen*

[in] size of authentication tag

## Description

*return* minimum size of output data length in bytes

## kcapi\_aead\_outbuflen\_enc

### LINUX

libkcapi Manual December 2020

### Name

kcapi\_aead\_outbuflen\_enc — return minimum encryption output buffer length

### Synopsis

```
uint32_t kcapi_aead_outbuflen_enc (struct kcapi_handle * handle,
uint32_t inlen, uint32_t assoclen, uint32_t taglen);
```

## Arguments

*handle*

[in] cipher handle

*inlen*

[in] size of plaintext

*assoclen*

[in] size of associated data (AD)

*taglen*

[in] size of authentication tag

## Description

*return* minimum size of output data length in bytes

## kcapi\_aead\_outbuflen\_dec

### LINUX

libkcapi ManualDecember 2020

## Name

kcapi\_aead\_outbuflen\_dec — return minimum decryption output buffer length

## Synopsis

```
uint32_t kcapi_aead_outbuflen_dec (struct kcapi_handle * handle,  
uint32_t inlen, uint32_t assoclen, uint32_t taglen);
```

## Arguments

*handle*

[in] cipher handle

*inlen*

[in] size of ciphertext

*assoclen*

[in] size of associated data (AD)

*taglen*

[in] size of authentication tag

## Description

*return* minimum size of output data length in bytes

## kcapi\_aead\_ccm\_nonce\_to\_iv

### LINUX

libkcapi ManualDecember 2020

## Name

kcapi\_aead\_ccm\_nonce\_to\_iv — convert CCM nonce into IV

## Synopsis

```
int kcap_i_aead_ccm_nonce_to_iv (const uint8_t * nonce, uint32_t
noncelen, uint8_t ** iv, uint32_t * ivlen);
```

## Arguments

*nonce*

[in] buffer with nonce

*noncelen*

[in] length of nonce

*iv*

[out] newly allocated buffer with IV

*ivlen*

[out] length of IV

## Description

This service function converts a CCM nonce value into an IV usable by the kernel crypto API.

Caller must free iv.

*return* 0 upon success; < 0 upon failure

## kcap\_i\_aead\_getdata\_input

### LINUX

libkcap\_i Manual December 2020

## Name

kcap\_i\_aead\_getdata\_input — get the pointers into input buffer

## Synopsis

```
void kcap_i_aead_getdata_input (struct kcap_i_handle * handle, uint8_t
* encdata, uint32_t encdatalen, int enc, uint8_t ** aad, uint32_t
* aadlen, uint8_t ** data, uint32_t * datalen, uint8_t ** tag,
uint32_t * taglen);
```

## Arguments

*handle*

[in] cipher handle

*encdata*

[in] data buffer returned by the encryption operation

*encdatalen*

[in] size of the encryption data buffer

*enc*

[in] does output buffer hold encryption or decryption result?

*aad*

[out] AD buffer pointer; when set to NULL, no data pointer is returned

*aadlen*

[out] length of AD; when aad was set to NULL, no information is returned

*data*

[out] pointer to output buffer from AEAD encryption operation when set to NULL, no data pointer is returned

*datalen*

[out] length of data buffer; when data was set to NULL, no information is returned

*tag*

[out] tag buffer pointer; when set to NULL, no data pointer is returned

*taglen*

[out] length of tag; when tag was set to NULL, no information is returned

## Description

This function is a service function to the consumer to locate the right ciphertext buffer offset holding the authentication tag. In addition, it provides the consumer with the length of the tag and the length of the ciphertext.

## kcapi\_aead\_getdata\_output

**LINUX**

libkcapi Manual December 2020

### Name

kcapi\_aead\_getdata\_output — get the pointers into output buffer

### Synopsis

```
void kcapi_aead_getdata_output (struct kcapi_handle * handle,  
uint8_t * encdata, uint32_t encdatalen, int enc, uint8_t ** aad,  
uint32_t * aadlen, uint8_t ** data, uint32_t * datalen, uint8_t **  
tag, uint32_t * taglen);
```



## Arguments

*handle*

[in] cipher handle

*encdata*

[in] data buffer returned by the encryption operation

*encdatalen*

[in] size of the encryption data buffer

*enc*

[in] does output buffer hold encryption or decryption result?

*aad*

[out] AD buffer pointer; when set to NULL, no data pointer is returned; returned pointer may also be NULL

*aadlen*

[out] length of AD; when aad was set to NULL, no information is returned

*data*

[out] pointer to output buffer from AEAD encryption operation when set to NULL, no data pointer is returned

*datalen*

[out] length of data buffer; when data was set to NULL, no information is returned

*tag*

[out] tag buffer pointer; when set to NULL, no data pointer is returned; returned pointer may also be NULL

*taglen*

[out] length of tag; when tag was set to NULL, no information is returned

## Description

This function is a service function to the consumer to locate the right ciphertext buffer offset holding the authentication tag. In addition, it provides the consumer with the length of the tag and the length of the ciphertext.

## Synchronous AEAD Cipher API - One Shot

### kcapi\_aead\_encrypt

**LINUX**

libkcapi Manual December 2020

#### Name

kcapi\_aead\_encrypt — synchronously encrypt AEAD data (one shot)

## Synopsis

```
int32_t kcapi_aead_encrypt (struct kcapi_handle * handle, const
uint8_t * in, uint32_t inlen, const uint8_t * iv, uint8_t * out,
uint32_t outlen, int access);
```

## Arguments

*handle*

[in] cipher handle

*in*

[in] plaintext data buffer

*inlen*

[in] length of plaintext buffer

*iv*

[in] IV to be used for cipher operation

*out*

[out] data buffer holding cipher text and authentication tag

*outlen*

[in] length of out buffer

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

The AEAD cipher operation requires the furnishing of the associated authentication data. In case such data is not required, it can be set to NULL and length value must be set to zero.

It is perfectly legal to use the same buffer as the plaintext and ciphertext pointers. That would mean that after the encryption operation, the plaintext is overwritten with the ciphertext.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

The IV buffer must be exactly `kcapi_cipher_ivsize` bytes in size.

After invoking this function the caller should use `kcapi_aead_getdata_output` to obtain the resulting ciphertext and authentication tag references.

## IMPORTANT NOTE

The kernel will only process `sysconf(_SC_PAGESIZE) * ALG_MAX_PAGES` at one time. Longer input data cannot be handled by the kernel.

*return* number of bytes encrypted upon success; a negative errno-style error code if an error occurred

## kcapi\_aead\_decrypt

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_aead\_decrypt — synchronously decrypt AEAD data (one shot)

#### Synopsis

```
int32_t kcapi_aead_decrypt (struct kcapi_handle * handle, const
uint8_t * in, uint32_t inlen, const uint8_t * iv, uint8_t * out,
uint32_t outlen, int access);
```

#### Arguments

*handle*

[in] cipher handle

*in*

[in] ciphertext data buffer

*inlen*

[in] length of in buffer

*iv*

[in] IV to be used for cipher operation

*out*

[out] plaintext data buffer

*outlen*

[in] length of out buffer

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE use vmsplice access; KCAPI\_ACCESS\_SENDMSG sendmsg access)

#### Description

The AEAD cipher operation requires the furnishing of the associated authentication data. In case such data is not required, it can be set to NULL and length value must be set to zero.

It is perfectly legal to use the same buffer as the plaintext and ciphertext pointers. That would mean that after the encryption operation, the ciphertext is overwritten with the plaintext.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the `vmsplice` call may not send all data to the kernel.

The IV buffer must be exactly `kcapi_cipher_ivsize` bytes in size.

To catch authentication errors (i.e. integrity violations) during the decryption operation, the return value of this call should be checked. If this function returns `-EBADMSG`, an authentication error was detected.

## IMPORTANT NOTE

The kernel will only process `sysconf(_SC_PAGESIZE) * ALG_MAX_PAGES` at one time. Longer input data cannot be handled by the kernel.

*return* number of bytes decrypted upon success; a negative `errno`-style error code if an error occurred

## Aynchronous AEAD Cipher API - One Shot

### `kcapi_aead_encrypt_aio`

#### LINUX

libkcapi Manual December 2020

#### Name

`kcapi_aead_encrypt_aio` — asynchronously encrypt AEAD data (one shot)

#### Synopsis

```
int32_t kcapi_aead_encrypt_aio (struct kcapi_handle * handle, struct
iovec * iniov, struct iovec * outiov, uint32_t iovlen, const
uint8_t * iv, int access);
```

#### Arguments

*handle*

[in] cipher handle

*iniov*

[in] array of scatter-gather list with input buffers

*outiov*

[out] array of scatter-gather list with output buffers

*iovlen*

[in] number of IOVECs in array

*iv*

[in] IV to be used for cipher operation

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

The AEAD cipher operation requires the furnishing of the associated authentication data. In case such data is not required, it can be set to NULL and length value must be set to zero.

Each IOVEC is processed with its individual AEAD cipher operation. The memory holding the input data will receive the processed data.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

The IV buffer must be exactly `kcapi_cipher_ivsize` bytes in size.

After invoking this function the caller should use `kcapi_aead_getdata_output` to obtain the resulting ciphertext and authentication tag references.

## IMPORTANT NOTE

The kernel will only process `sysconf(_SC_PAGESIZE) * ALG_MAX_PAGES` at one time. Longer input data cannot be handled by the kernel.

*return* number of bytes encrypted upon success; a negative errno-style error code if an error occurred

## kcapi\_aead\_decrypt\_aio

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_aead_decrypt_aio` — asynchronously decrypt AEAD data (one shot)

### Synopsis

```
int32_t kcapi_aead_decrypt_aio (struct kcapi_handle * handle, struct
iovec * iniov, struct iovec * outiov, uint32_t iovlen, const
uint8_t * iv, int access);
```

### Arguments

*handle*

[in] cipher handle

*iniov*

[in] array of scatter-gather list with input buffers

*outiov*

[out] array of scatter-gather list with output buffers

*iovlen*

[in] number of IOVECs in array

*iv*

[in] IV to be used for cipher operation

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

The AEAD cipher operation requires the furnishing of the associated authentication data. In case such data is not required, it can be set to NULL and length value must be set to zero.

Each IOVEC is processed with its individual AEAD cipher operation. The memory holding the input data will receive the processed data.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

The IV buffer must be exactly `kcapi_cipher_ivsize` bytes in size.

To catch authentication errors (i.e. integrity violations) during the decryption operation, the return value of this call should be checked. If this function returns `-EBADMSG`, an authentication error was detected.

## IMPORTANT NOTE

The kernel will only process `sysconf(_SC_PAGESIZE) * ALG_MAX_PAGES` at one time. Longer input data cannot be handled by the kernel.

*return* number of bytes encrypted upon success; a negative errno-style error code if an error occurred

## Synchronous AEAD Cipher API - Stream

### `kcapi_aead_stream_init_enc`

**LINUX**

libkcapi Manual December 2020

#### Name

`kcapi_aead_stream_init_enc` — start an encryption operation (stream)

## Synopsis

```
int32_t kcap_i_aead_stream_init_enc (struct kcap_i_handle * handle,
const uint8_t * iv, struct iovec * iov, uint32_t iovlen);
```

## Arguments

*handle*

[in] cipher handle

*iv*

[in] IV to be used for cipher operation

*iov*

[in] scatter/gather list with data to be encrypted. This is the pointer to the first iov entry if an array of iov entries is supplied. See `sendmsg(2)` for details on how iov is to be used. This pointer may be NULL if no data to be encrypted is available at the point of the call.

*iovlen*

[in] number of scatter/gather list elements. If iov is NULL, this value must be zero.

## Description

A stream encryption operation is started with this call. Multiple successive `kcap_i_aead_stream_update` function calls can be invoked to send more plaintext data to be encrypted. The kernel buffers the input until `kcap_i_aead_stream_op` picks up the encrypted data. Once plaintext is encrypted during the `kcap_i_aead_stream_op` it is removed from the kernel buffer.

Note, unlike the corresponding symmetric cipher API, the function calls of `kcap_i_aead_stream_update` and `kcap_i_aead_stream_op` cannot be mixed! This due to the nature of AEAD where the cipher operation ensures the integrity of the entire data (decryption) or calculates a message digest over the entire data (encryption).

When using the stream API, the caller must ensure that data is sent in the correct order (regardless whether data is sent in multiple chunks using `kcap_i_aead_stream_init_enc` or `kcap_i_cipher_stream_update`): (i) the complete associated data must be provided, followed by (ii) the plaintext.

The IV buffer must be exactly `kcap_i_cipher_ivsize` bytes in size.

*return* number of bytes sent to the kernel upon success; a negative `errno`-style error code if an error occurred

## `kcap_i_aead_stream_init_dec`

**LINUX**

**Name**

`kcapi_aead_stream_init_dec` — start a decryption operation (stream)

**Synopsis**

```
int32_t kcapi_aead_stream_init_dec (struct kcapi_handle * handle,
const uint8_t * iv, struct iovec * iov, uint32_t iovlen);
```

**Arguments**

*handle*

[in] cipher handle

*iv*

[in] IV to be used for cipher operation

*iov*

[in] scatter/gather list with data to be encrypted. This is the pointer to the first iov entry if an array of iov entries is supplied. See `sendmsg(2)` for details on how iov is to be used. This pointer may be NULL if no data to be encrypted is available at the point of the call.

*iovlen*

[in] number of scatter/gather list elements. If iov is NULL, this value must be zero.

**Description**

A stream decryption operation is started with this call. Multiple successive `kcapi_aead_stream_update` function calls can be invoked to send more ciphertext data to be encrypted. The kernel buffers the input until `kcapi_aead_stream_op` picks up the decrypted data. Once ciphertext is decrypted during the `kcapi_aead_stream_op` it is removed from the kernel buffer.

Note, unlike the corresponding symmetric cipher API, the function calls of `kcapi_aead_stream_update` and `kcapi_aead_stream_op` cannot be mixed! This due to the nature of AEAD where the cipher operation ensures the integrity of the entire data (decryption) or calculates a message digest over the entire data (encryption).

When using the stream API, the caller must ensure that data is sent in the correct order (regardless whether data is sent in multiple chunks using `kcapi_aead_stream_init_enc` or `kcapi_cipher_stream_update`): (i) the complete associated data must be provided, followed by (ii) the plaintext. For decryption, also (iii) the tag value must be sent.

The IV buffer must be exactly `kcapi_cipher_ivsize` bytes in size.

*return* number of bytes sent to the kernel upon success; a negative errno-style error code if an error occurred



## kcapi\_aead\_stream\_update

### LINUX

libkcapi Manual December 2020

#### Name

`kcapi_aead_stream_update` — send more data for processing (stream)

#### Synopsis

```
int32_t kcapi_aead_stream_update (struct kcapi_handle * handle,
struct iovec * iov, uint32_t iovlen);
```

#### Arguments

*handle*

[in] cipher handle

*iov*

[in] scatter/gather list with data to be processed by the cipher operation.

*iovlen*

[in] number of scatter/gather list elements.

#### Description

Using this function call, more plaintext for encryption or ciphertext for decryption can be submitted to the kernel.

Note, see the order of input data as outlined in `kcapi_aead_stream_init_dec`.

This function may cause the caller to sleep if the kernel buffer holding the data is getting full. The process will be woken up once more buffer space becomes available by calling `kcapi_aead_stream_op`.

#### Note

The last block of input data MUST be provided with `kcapi_aead_stream_update_last` as the kernel must be informed about the completion of the input data.

With the separate API calls of `kcapi_aead_stream_update` and `kcapi_aead_stream_op` a multi-threaded application can be implemented where one thread sends data to be processed and one thread picks up data processed by the cipher operation.

#### IMPORTANT NOTE

The kernel will only process `sysconf(_SC_PAGESIZE) * ALG_MAX_PAGES` at one time. Longer input data cannot be handled by the kernel.

## WARNING

The memory referenced by *iov* is not accessed by the kernel during this call. The memory is first accessed when `kcapi_cipher_stream_op` is called. Thus, you **MUST** make sure that the referenced memory is still present at the time `kcapi_cipher_stream_op` is called.

*return* number of bytes sent to the kernel upon success; a negative errno-style error code if an error occurred

## kcapi\_aead\_stream\_update\_last

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_aead_stream_update_last` — send last data for processing (stream)

### Synopsis

```
int32_t kcapi_aead_stream_update_last (struct kcapi_handle * handle,
struct iovec * iov, uint32_t iovlen);
```

### Arguments

*handle*

[in] cipher handle

*iov*

[in] scatter/gather list with data to be processed by the cipher operation.

*iovlen*

[in] number of scatter/gather list elements.

### Description

Using this function call, more plaintext for encryption or ciphertext for decryption can be submitted to the kernel.

This call is identical to the `kcapi_aead_stream_update` call with the exception that it marks the last data buffer before the cipher operation is triggered. Typically, the tag value is provided with this call.

## WARNING

The memory referenced by *iov* is not accessed by the kernel during this call. The memory is first accessed when `kcapi_cipher_stream_op` is called. Thus, you **MUST** make sure that the referenced memory is still present at the time `kcapi_cipher_stream_op` is called.

*return* number of bytes sent to the kernel upon success; a negative errno-style error code if an error occurred

## kcapi\_aead\_stream\_op

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_aead\_stream\_op — obtain processed data (stream)

#### Synopsis

```
int32_t kcapi_aead_stream_op (struct kcapi_handle * handle, struct
iovec * iov, uint32_t iovlen);
```

#### Arguments

*handle*

[in] cipher handle

*iov*

[out] scatter/gather list pointing to buffers to be filled with the resulting data from a cipher operation.

*iovlen*

[in] number of outiovec scatter/gather list elements.

#### Description

This function may cause the caller to sleep if the kernel buffer holding the data is empty. The process will be woken up once more data is sent by calling `kcapi_cipher_stream_update`.

Note, when supplying buffers that are not multiple of block size, the buffers will only be filled up to the maximum number of full block sizes that fit into the buffer.

*return* number of bytes obtained from the kernel upon success; a negative errno-style error code if an error occurred

## Message Digest Cipher API - Generic

### kcapi\_md\_init

#### LINUX

## Name

`kcapi_md_init` — initialize cipher handle

## Synopsis

```
int kcapi_md_init (struct kcapi_handle ** handle, const char *  
ciphername, uint32_t flags);
```

## Arguments

*handle*

[out] cipher handle filled during the call

*ciphername*

[in] kernel crypto API cipher name as specified in `/proc/crypto`

*flags*

[in] flags specifying the type of cipher handle

## Description

This function provides the initialization of a (keyed) message digest handle and establishes the connection to the kernel.

On success, a pointer to `kcapi_handle` object is returned in `*handle`. Function `kcapi_md_destroy` should be called afterwards to free resources.

*return* 0 upon success; -ENOENT - algorithm not available; -EOPNOTSUPP - AF\_ALG family not available; -EINVAL - accept syscall failed; -ENOMEM - cipher handle cannot be allocated

## `kcapi_md_destroy`

### LINUX

## Name

`kcapi_md_destroy` — close the message digest handle and release resources

## Synopsis

```
void kcapi_md_destroy (struct kcapi_handle * handle);
```

## Arguments

*handle*

[in] cipher handle to release

## kcapi\_md\_setkey

**LINUX**

libkcapi Manual December 2020

### Name

kcapi\_md\_setkey — set the key for the message digest handle

### Synopsis

```
int kcapi_md_setkey (struct kcapi_handle * handle, const uint8_t *
key, uint32_t keylen);
```

## Arguments

*handle*

[in] cipher handle

*key*

[in] key buffer

*keylen*

[in] length of key buffer

### Description

With this function, the caller sets the key for subsequent hashing operations. This call is applicable for keyed message digests.

After the caller provided the key, the caller may securely destroy the key as it is now maintained by the kernel.

*return* 0 upon success; a negative errno-style error code if an error occurred

## kcapi\_md\_digestsize

**LINUX**

## Name

`kcapi_md_digestsize` — return the size of the message digest

## Synopsis

```
uint32_t kcapi_md_digestsize (struct kcapi_handle * handle);
```

## Arguments

*handle*

[in] cipher handle

## Description

The returned message digest size can be used before the `kcapi_md_final` function invocation to determine the right memory size to be allocated for this call.

*return* > 0 specifying the block size; 0 on error

# Message Digest Cipher API - One Shot

## kcapi\_md\_digest

### LINUX

## Name

`kcapi_md_digest` — calculate message digest on buffer (one-shot)

## Synopsis

```
int32_t kcapi_md_digest (struct kcapi_handle * handle, const
uint8_t * in, uint32_t inlen, uint8_t * out, uint32_t outlen);
```

## Arguments

*handle*

[in] cipher handle

*in*

[in] buffer with input data

*inlen*

[in] length of input buffer

*out*

[out] buffer for message digest

*outlen*

[in] length of out

**Description**

With this one-shot function, a message digest of the given buffer is generated. The output buffer must be allocated by the caller and have at least the length of the message digest size for the chosen message digest.

The message digest handle must have been initialized, potentially by also setting the key using the generic message digest API functions.

The input buffer can be at most INT\_MAX in size.

*return* size of message digest upon success; -EIO - data cannot be obtained; -ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest

**Message Digest Cipher API - Convenience****kcapi\_md\_sha1****LINUX**

libkcapi Manual December 2020

**Name**

kcapi\_md\_sha1 — SHA-1 message digest on one buffer

**Synopsis**

```
int32_t kcapi_md_sha1 (const uint8_t * in, uint32_t inlen, uint8_t
* out, uint32_t outlen);
```

**Arguments***in*

[in] buffer with input data

*inlen*

[in] length of input buffer

*out*

[out] buffer for message digest

*outlen*

[in] length of out

## Description

With this one-shot convenience function, a message digest of the given buffer is generated. The output buffer must be allocated by the caller and have at least the length of the message digest size for the chosen message digest.

*return* size of message digest upon success; -EIO - data cannot be obtained; -ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest

## kcapi\_md\_sha224

### LINUX

libkcapi Manual December 2020

### Name

kcapi\_md\_sha224 — SHA-224 message digest on one buffer

### Synopsis

```
int32_t kcapi_md_sha224 (const uint8_t * in, uint32_t inlen, uint8_t
* out, uint32_t outlen);
```

### Arguments

*in*

[in] buffer with input data

*inlen*

[in] length of input buffer

*out*

[out] buffer for message digest

*outlen*

[in] length of out

## Description

With this one-shot convenience function, a message digest of the given buffer is generated. The output buffer must be allocated by the caller and have at least the length of the message digest size for the chosen message digest.

*return* size of message digest upon success; -EIO - data cannot be obtained; -ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest



## kcapi\_md\_sha256

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_md\_sha256 — SHA-256 message digest on one buffer

#### Synopsis

```
int32_t kcapi_md_sha256 (const uint8_t * in, uint32_t inlen, uint8_t
* out, uint32_t outlen);
```

#### Arguments

*in*

[in] buffer with input data

*inlen*

[in] length of input buffer

*out*

[out] buffer for message digest

*outlen*

[in] length of out

#### Description

With this one-shot convenience function, a message digest of the given buffer is generated. The output buffer must be allocated by the caller and have at least the length of the message digest size for the chosen message digest.

*return* size of message digest upon success; -EIO - data cannot be obtained; -ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest

## kcapi\_md\_sha384

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_md\_sha384 — SHA-384 message digest on one buffer

## Synopsis

```
int32_t kcap_i_md_sha384 (const uint8_t * in, uint32_t inlen, uint8_t
* out, uint32_t outlen);
```

## Arguments

*in*

[in] buffer with input data

*inlen*

[in] length of input buffer

*out*

[out] buffer for message digest

*outlen*

[in] length of out

## Description

With this one-shot convenience function, a message digest of the given buffer is generated. The output buffer must be allocated by the caller and have at least the length of the message digest size for the chosen message digest.

*return* size of message digest upon success; -EIO - data cannot be obtained; -ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest

## kcap\_i\_md\_sha512

### LINUX

libkcap\_i ManualDecember 2020

## Name

kcap\_i\_md\_sha512 — SHA-512 message digest on one buffer

## Synopsis

```
int32_t kcap_i_md_sha512 (const uint8_t * in, uint32_t inlen, uint8_t
* out, uint32_t outlen);
```

## Arguments

*in*

[in] buffer with input data

*inlen*

[in] length of input buffer

*out*

[out] buffer for message digest

*outlen*

[in] length of out

## Description

With this one-shot convenience function, a message digest of the given buffer is generated. The output buffer must be allocated by the caller and have at least the length of the message digest size for the chosen message digest.

*return* size of message digest upon success; -EIO - data cannot be obtained; -ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest

## kcapi\_md\_hmac\_sha1

### LINUX

libkcapi Manual December 2020

### Name

kcapi\_md\_hmac\_sha1 — HMAC SHA-1 keyed message digest on one buffer

### Synopsis

```
int32_t kcapi_md_hmac_sha1 (const uint8_t * key, uint32_t keylen,
const uint8_t * in, uint32_t inlen, uint8_t * out, uint32_t
outlen);
```

### Arguments

*key*

[in] buffer with HMAC key

*keylen*

[in] length of HMAC key buffer

*in*

[in] buffer with input data

*inlen*

[in] length of input buffer

*out*

[out] buffer for message digest

*outlen*  
[in] length of out

## Description

With this one-shot convenience function, a keyed message digest of the given buffer is generated. The output buffer must be allocated by the caller and have at least the length of the message digest size for the chosen keyed message digest.

*return* size of message digest upon success; -EIO - data cannot be obtained; -ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest

## kcapi\_md\_hmac\_sha224

### LINUX

libkcapi Manual December 2020

### Name

kcapi\_md\_hmac\_sha224 — HMAC SHA-224 keyed message digest on one buffer

### Synopsis

```
int32_t kcapi_md_hmac_sha224 (const uint8_t * key, uint32_t keylen,  
const uint8_t * in, uint32_t inlen, uint8_t * out, uint32_t  
outlen);
```

### Arguments

*key*  
[in] buffer with HMAC key

*keylen*  
[in] length of HMAC key buffer

*in*  
[in] buffer with input data

*inlen*  
[in] length of input buffer

*out*  
[out] buffer for message digest

*outlen*  
[in] length of out

## Description

With this one-shot convenience function, a keyed message digest of the given buffer is generated. The output buffer must be allocated by the caller and have at least the length of the message digest size for the chosen keyed message digest.

*return* size of message digest upon success; -EIO - data cannot be obtained; -ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest

## kcapi\_md\_hmac\_sha256

### LINUX

libkcapi Manual December 2020

### Name

kcapi\_md\_hmac\_sha256 — HMAC SHA-256 keyed message digest on one buffer

### Synopsis

```
int32_t kcapi_md_hmac_sha256 (const uint8_t * key, uint32_t keylen,
const uint8_t * in, uint32_t inlen, uint8_t * out, uint32_t
outlen);
```

### Arguments

*key*

[in] buffer with HMAC key

*keylen*

[in] length of HMAC key buffer

*in*

[in] buffer with input data

*inlen*

[in] length of input buffer

*out*

[out] buffer for message digest

*outlen*

[in] length of out

### Description

With this one-shot convenience function, a keyed message digest of the given buffer is generated. The output buffer must be allocated by the caller and have at least the length of the message digest size for the chosen keyed message digest.

*return* size of message digest upon success; -EIO - data cannot be obtained; - ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest

## kcapi\_md\_hmac\_sha384

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_md\_hmac\_sha384 — HMAC SHA-384 keyed message digest on one buffer

#### Synopsis

```
int32_t kcapi_md_hmac_sha384 (const uint8_t * key, uint32_t keylen,
const uint8_t * in, uint32_t inlen, uint8_t * out, uint32_t
outlen);
```

#### Arguments

*key*

[in] buffer with HMAC key

*keylen*

[in] length of HMAC key buffer

*in*

[in] buffer with input data

*inlen*

[in] length of input buffer

*out*

[out] buffer for message digest

*outlen*

[in] length of out

#### Description

With this one-shot convenience function, a keyed message digest of the given buffer is generated. The output buffer must be allocated by the caller and have at least the length of the message digest size for the chosen keyed message digest.

*return* size of message digest upon success; -EIO - data cannot be obtained; - ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest

## kcapi\_md\_hmac\_sha512

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_md\_hmac\_sha512 — HMAC SHA-512 keyed message digest on one buffer

#### Synopsis

```
int32_t kcapi_md_hmac_sha512 (const uint8_t * key, uint32_t keylen,
const uint8_t * in, uint32_t inlen, uint8_t * out, uint32_t
outlen);
```

#### Arguments

*key*

[in] buffer with HMAC key

*keylen*

[in] length of HMAC key buffer

*in*

[in] buffer with input data

*inlen*

[in] length of input buffer

*out*

[out] buffer for message digest

*outlen*

[in] length of out

#### Description

With this one-shot convenience function, a keyed message digest of the given buffer is generated. The output buffer must be allocated by the caller and have at least the length of the message digest size for the chosen keyed message digest.

*return* size of message digest upon success; -EIO - data cannot be obtained; -ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest

## Message Digest Cipher API - Stream

### kcapi\_md\_update

**LINUX**

libkcapi Manual December 2020

#### Name

kcapi\_md\_update — message digest update function (stream)

#### Synopsis

```
int32_t kcapi_md_update (struct kcapi_handle * handle, const
uint8_t * buffer, uint32_t len);
```

#### Arguments

*handle*

[in] cipher handle

*buffer*

[in] holding the data to add to the message digest

*len*

[in] buffer length

#### Description

The input buffer can be at most INT\_MAX in size.

*return* 0 upon success; a negative errno-style error code if an error occurred

### kcapi\_md\_final

**LINUX**

libkcapi Manual December 2020

#### Name

kcapi\_md\_final — message digest finalization function (stream)

#### Synopsis

```
int32_t kcapi_md_final (struct kcapi_handle * handle, uint8_t *
buffer, uint32_t len);
```



## Arguments

*handle*

[in] cipher handle

*buffer*

[out] filled with the message digest

*len*

[in] buffer length

## Description

*return* size of message digest upon success; -EIO - data cannot be obtained; -ENOMEM - buffer is too small for the complete message digest, the buffer is filled with the truncated message digest

## Random Number API

### kcapi\_rng\_init

#### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_rng\_init — initialize cipher handle

#### Synopsis

```
int kcapi_rng_init (struct kcapi_handle ** handle, const char *
ciphername, uint32_t flags);
```

## Arguments

*handle*

[out] cipher handle filled during the call

*ciphername*

[in] kernel crypto API cipher name as specified in /proc/crypto

*flags*

[in] flags specifying the type of cipher handle (unused for RNG)

## Description

This function provides the initialization of a random number generator handle and establishes the connection to the kernel.

On success, a pointer to `kcapi_handle` object is returned in `*handle`. Function `kcapi_rng_destroy` should be called afterwards to free resources.

*return* 0 upon success; -ENOENT - algorithm not available; -EOPNOTSUPP - AF\_ALG family not available; -EINVAL - accept syscall failed -ENOMEM - cipher handle cannot be allocated

## kcapi\_rng\_destroy

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_rng_destroy` — close the RNG handle and release resources

### Synopsis

```
void kcapi_rng_destroy (struct kcapi_handle * handle);
```

### Arguments

*handle*

[in] cipher handle to release

## kcapi\_rng\_seed

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_rng_seed` — seed the RNG

### Synopsis

```
int kcapi_rng_seed (struct kcapi_handle * handle, uint8_t * seed,  
uint32_t seedlen);
```

## Arguments

*handle*

[in] cipher handle

*seed*

[in] seed data

*seedlen*

[in] size of seed

## Description

Note, this call must be called to initialize the selected RNG. When the SP800-90A DRBG is used, this call causes the DRBG to seed itself from the internal noise sources.

Note, in case of using the SP800-90A DRBG, the seed buffer may be NULL. If it is not NULL, the DRBG uses the given data either as personalization string in case of the initial seeding or additional data for reseeding.

*return* 0 upon success; a negative errno-style error code if an error occurred

## kcapi\_rng\_generate

### LINUX

libkcapi Manual December 2020

### Name

kcapi\_rng\_generate — generate a random number

### Synopsis

```
int32_t kcapi_rng_generate (struct kcapi_handle * handle, uint8_t *
buffer, uint32_t len);
```

## Arguments

*handle*

[in] cipher handle

*buffer*

[out] filled with the random number

*len*

[in] buffer length

## Description

*return* size of random number generated upon success; -EIO - data cannot be obtained

## kcapi\_rng\_seedsize

### LINUX

libkcapi ManualDecember 2020

## Name

kcapi\_rng\_seedsize — return required seed size of DRNG

## Synopsis

```
uint32_t kcapi_rng_seedsize (struct kcapi_handle * handle);
```

## Arguments

*handle*

[in] cipher handle

## Description

*return* > 0 specifying the block size; 0 on error

## Random Number API - Convenience

## kcapi\_rng\_get\_bytes

### LINUX

libkcapi ManualDecember 2020

## Name

kcapi\_rng\_get\_bytes — Convenience function to generate random bytes

## Synopsis

```
int32_t kcapi_rng_get_bytes (uint8_t * buffer, uint32_t outlen);
```

## Arguments

*buffer*

[out] filled with the random number

*outlen*

[in] buffer length

## Description

This convenience function generates random bytes of the size of *outlen* and stores them into the provided buffer.

*return* size of random number generated upon success; -EIO - data cannot be obtained

## Asymmetric Cipher API - Generic

API function calls used to invoke asymmetric ciphers.

### kcapi\_akcipher\_init

**LINUX**

libkcapi Manual December 2020

## Name

`kcapi_akcipher_init` — initialize cipher handle

## Synopsis

```
int kcapi_akcipher_init (struct kcapi_handle ** handle, const char *
ciphername, uint32_t flags);
```

## Arguments

*handle*

[out] cipher handle filled during the call

*ciphername*

[in] kernel crypto API cipher name as specified in `/proc/crypto`

*flags*

[in] flags specifying the type of cipher handle

## Description

This function provides the initialization of an asymmetric cipher handle and establishes the connection to the kernel.

On success, a pointer to `kcapi_handle` object is returned in `*handle`. Function `kcapi_akcipher_destroy` should be called afterwards to free resources.

*return* 0 upon success; -ENOENT - algorithm not available; -EOPNOTSUPP - AF\_ALG family not available; -EINVAL - accept syscall failed -ENOMEM - cipher handle cannot be allocated

## kcapi\_akcipher\_destroy

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_akcipher_destroy` — close the cipher handle and release resources

### Synopsis

```
void kcapi_akcipher_destroy (struct kcapi_handle * handle);
```

### Arguments

*handle*

[in] cipher handle to release

## kcapi\_akcipher\_setkey

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_akcipher_setkey` — set the private key for the cipher handle

### Synopsis

```
int kcapi_akcipher_setkey (struct kcapi_handle * handle, const  
uint8_t * key, uint32_t keylen);
```

## Arguments

*handle*

[in] cipher handle

*key*

[in] key buffer in DER format

*keylen*

[in] length of key buffer

## Description

With this function, the caller sets the key for subsequent cipher operations.

The key must be in DER format as follows

```
SEQUENCE { version INTEGER, n INTEGER ({ rsa_get_n }), e INTEGER ({
rsa_get_e }), d INTEGER ({ rsa_get_d }), prime1 INTEGER, prime2 INTEGER,
exponent1 INTEGER, exponent2 INTEGER, coefficient INTEGER }
```

After the caller provided the key, the caller may securely destroy the key as it is now maintained by the kernel.

*return* upon success the value of the maximum size for the asymmetric operation is returned (e.g. the modulus size); a negative errno-style error code if an error occurred

## kcapi\_akcipher\_setpubkey

### LINUX

libkcapi Manual December 2020

### Name

kcapi\_akcipher\_setpubkey — set the public key for the cipher handle

### Synopsis

```
int kcapi_akcipher_setpubkey (struct kcapi_handle * handle, const
uint8_t * key, uint32_t keylen);
```

## Arguments

*handle*

[in] cipher handle

*key*

[in] key buffer in DER format

*keylen*

[in] length of key buffer

## Description

With this function, the caller sets the key for subsequent cipher operations.

The key must be in DER format as follows

SEQUENCE { n INTEGER ({ rsa\_get\_n }), e INTEGER ({ rsa\_get\_e }) }

After the caller provided the key, the caller may securely destroy the key as it is now maintained by the kernel.

*return* upon success the value of the maximum size for the asymmetric operation is returned (e.g. the modulus size); a negative errno-style error code if an error occurred

## Synchronous asymmetric Cipher API - One Shot

### kcapi\_akcipher\_encrypt

**LINUX**

libkcapi Manual December 2020

#### Name

kcapi\_akcipher\_encrypt — encrypt data

#### Synopsis

```
int32_t kcapi_akcipher_encrypt (struct kcapi_handle * handle, const
uint8_t * in, uint32_t inlen, uint8_t * out, uint32_t outlen, int
access);
```

#### Arguments

*handle*

[in] cipher handle

*in*

[in] plaintext data buffer

*inlen*

[in] length of in buffer

*out*

[out] ciphertext data buffer

*outlen*

[in] length of out buffer



*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

It is perfectly legal to use the same buffer as the plaintext and ciphertext pointers. That would mean that after the encryption operation, the plaintext is overwritten with the ciphertext.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

If the output size is insufficiently large, `-EINVAL` is returned. The output buffer must be at least as large as the modulus of the used key.

*return* number of bytes returned by the encryption operation upon success; a negative `errno`-style error code if an error occurred

## kcapi\_akcipher\_decrypt

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_akcipher_decrypt` — decrypt data

### Synopsis

```
int32_t kcapi_akcipher_decrypt (struct kcapi_handle * handle, const
uint8_t * in, uint32_t inlen, uint8_t * out, uint32_t outlen, int
access);
```

### Arguments

*handle*

[in] cipher handle

*in*

[in] ciphertext data buffer

*inlen*

[in] length of in buffer

*out*

[out] plaintext data buffer

*outlen*

[in] length of out buffer

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

It is perfectly legal to use the same buffer as the plaintext and ciphertext pointers. That would mean that after the decryption operation, the ciphertext is overwritten with the plaintext.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

If the output size is insufficiently large, `-EINVAL` is returned. The output buffer must be at least as large as the modulus of the used key.

*return* number of bytes returned by the decryption operation upon success; a negative `errno`-style error code if an error occurred

## kcapi\_akcipher\_sign

**LINUX**

libkcapi Manual December 2020

### Name

`kcapi_akcipher_sign` — signature generation

### Synopsis

```
int32_t kcapi_akcipher_sign (struct kcapi_handle * handle, const
uint8_t * in, uint32_t inlen, uint8_t * out, uint32_t outlen, int
access);
```

### Arguments

*handle*

[in] cipher handle

*in*

[in] message data buffer

*inlen*

[in] length of in buffer

*out*

[out] signature data buffer

*outlen*

[in] length of out buffer

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

It is perfectly legal to use the same buffer as the message and signature pointers. That would mean that after the signature generation operation, the message is overwritten with the signature.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

If the output size is insufficiently large, `-EINVAL` is returned. The output buffer must be at least as large as the modulus of the used key.

*return* number of bytes returned by the signature generation operation upon success; a negative `errno`-style error code if an error occurred

## kcapi\_akcipher\_verify

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_akcipher_verify` — signature verification

### Synopsis

```
int32_t kcapi_akcipher_verify (struct kcapi_handle * handle, const
uint8_t * in, uint32_t inlen, uint8_t * out, uint32_t outlen, int
access);
```

### Arguments

*handle*

[in] cipher handle

*in*

[in] message data buffer

*inlen*

[in] length of in buffer

*out*

[out] signature data buffer

*outlen*

[in] length of out buffer

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

It is perfectly legal to use the same buffer as the message and signature pointers. That would mean that after the signature generation operation, the message is overwritten with the signature.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

If the output size is insufficiently large, `-EINVAL` is returned. The output buffer must be at least as large as the modulus of the used key.

To catch signature verification errors, the return value of this call should be checked. If this function returns `-EBADMSG`, the verification of the signature failed.

*return* number of bytes returned by the signature verification operation upon success; a negative `errno`-style error code if an error occurred

## Asynchronous asymmetric Cipher API - One Shot

### kcapi\_akcipher\_encrypt\_aio

**LINUX**

libkcapi Manual December 2020

#### Name

`kcapi_akcipher_encrypt_aio` — encrypt data (asynchronous one shot)

#### Synopsis

```
int32_t kcapi_akcipher_encrypt_aio (struct kcapi_handle * handle,
struct iovec * iniov, struct iovec * outiov, uint32_t iovlen, int
access);
```

#### Arguments

*handle*

[in] cipher handle

*iniov*

[in] head of scatter-gather list array holding the plaintext

*outiov*

[out] head of scatter-gather list of the destination buffers filled with cipher-text

*iovlen*

[in] number of scatter-gather list entries

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

The individual scatter-gather list entries are processed with separate invocations of the the given cipher.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

*return* number of bytes encrypted upon success; a negative errno-style error code if an error occurred

## kcapi\_akcipher\_decrypt\_aio

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_akcipher_decrypt_aio` — decrypt data (asynchronous one shot)

### Synopsis

```
int32_t kcapi_akcipher_decrypt_aio (struct kcapi_handle * handle,
struct iovec * iniov, struct iovec * outiov, uint32_t iovlen, int
access);
```

### Arguments

*handle*

[in] cipher handle

*iniov*

[in] head of scatter-gather list array holding the plaintext

*outiov*

[out] head of scatter-gather list of the destination buffers filled with cipher-text

*iovlen*

[in] number of scatter-gather list entries

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

The individual scatter-gather list entries are processed with separate invocations of the the given cipher.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

*return* number of bytes decrypted upon success; a negative errno-style error code if an error occurred

## kcapi\_akcipher\_sign\_aio

**LINUX**

libkcapi Manual December 2020

### Name

kcapi\_akcipher\_sign\_aio — sign data (asynchronous one shot)

### Synopsis

```
int32_t kcapi_akcipher_sign_aio (struct kcapi_handle * handle,  
struct iovec * iniov, struct iovec * outiov, uint32_t iovlen, int  
access);
```

### Arguments

*handle*

[in] cipher handle

*iniov*

[in] head of scatter-gather list array holding the plaintext

*outiov*

[out] head of scatter-gather list of the destination buffers filled with cipher-text

*iovlen*

[in] number of scatter-gather list entries

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

The individual scatter-gather list entries are processed with separate invocations of the the given cipher.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the vmsplice call may not send all data to the kernel.

*return* number of bytes signed upon success; a negative errno-style error code if an error occurred

## kcapi\_akcipher\_verify\_aio

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_akcipher_verify_aio` — verify data (asynchronous one shot)

### Synopsis

```
int32_t kcapi_akcipher_verify_aio (struct kcapi_handle * handle,
struct iovec * iniov, struct iovec * outiov, uint32_t iovlen, int
access);
```

### Arguments

*handle*

[in] cipher handle

*iniov*

[in] head of scatter-gather list array holding the plaintext

*outiov*

[out] head of scatter-gather list of the destination buffers filled with cipher-text

*iovlen*

[in] number of scatter-gather list entries

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

The individual scatter-gather list entries are processed with separate invocations of the the given cipher.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the `vmsplice` call may not send all data to the kernel.

*return* number of bytes verify upon success; a negative `errno`-style error code if an error occurred

## Asymmetric Cipher API - Stream

### `kcapi_akcipher_stream_init_enc`

#### LINUX

libkcapi Manual December 2020

#### Name

`kcapi_akcipher_stream_init_enc` — start an encryption operation  
(stream)

#### Synopsis

```
int32_t kcapi_akcipher_stream_init_enc (struct kcapi_handle * handle,  
struct iovec * iov, uint32_t iovlen);
```

#### Arguments

*handle*

[in] cipher handle

*iov*

[in] scatter/gather list with data to be encrypted. This is the pointer to the first `iovec` entry if an array of `iovec` entries is supplied. See `sendmsg(2)` for details on how `iov` is to be used. This pointer may be `NULL` if no data to be encrypted is available at the point of the call.

*iovlen*

[in] number of scatter/gather list elements. If `iov` is `NULL`, this value must be zero.

#### Description

A stream encryption operation is started with this call. Multiple successive `kcapi_akcipher_stream_update` function calls can be invoked to send more plaintext data to be encrypted. The last invocation to supply data must be done with `kcapi_akcipher_stream_update_last`. The kernel buffers the input until `kcapi_akcipher_stream_op` picks up the encrypted data. Once plaintext is



encrypted during the `kcapi_cipher_stream_op` it is removed from the kernel buffer.

The function calls of `kcapi_akcipher_stream_update` and `kcapi_akcipher_stream_op` can be mixed, even by multiple threads of an application.

*return* number of bytes sent to the kernel upon success; a negative `errno`-style error code if an error occurred

## kcapi\_akcipher\_stream\_init\_dec

### LINUX

libkcapi Manual December 2020

### Name

`kcapi_akcipher_stream_init_dec` — start an decryption operation (stream)

### Synopsis

```
int32_t kcapi_akcipher_stream_init_dec (struct kcapi_handle * handle,
struct iovec * iov, uint32_t iovlen);
```

### Arguments

*handle*

[in] cipher handle

*iov*

[in] scatter/gather list with data to be decrypted. This is the pointer to the first iov entry if an array of iov entries is supplied. See `sendmsg(2)` for details on how iov is to be used. This pointer may be NULL if no data to be decrypted is available at the point of the call.

*iovlen*

[in] number of scatter/gather list elements. If iov is NULL, this value must be zero.

### Description

A stream decryption operation is started with this call. Multiple successive `kcapi_akcipher_stream_update` function calls can be invoked to send more plaintext data to be decrypted. The last invocation to supply data must be done with `kcapi_akcipher_stream_update_last`. The kernel buffers the input until `kcapi_akcipher_stream_op` picks up the encrypted data. Once plaintext is decrypted during the `kcapi_cipher_stream_op` it is removed from the kernel buffer.

The function calls of `kcapi_akcipher_stream_update` and `kcapi_akcipher_stream_op` can be mixed, even by multiple threads of an application.

*return* number of bytes sent to the kernel upon success; a negative errno-style error code if an error occurred

## kcapi\_akcipher\_stream\_init\_sgn

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_akcipher\_stream\_init\_sgn — start an signing operation (stream)

#### Synopsis

```
int32_t kcapi_akcipher_stream_init_sgn (struct kcapi_handle * handle,
struct iovec * iov, uint32_t iovlen);
```

#### Arguments

*handle*

[in] cipher handle

*iov*

[in] scatter/gather list with data to be signed. This is the pointer to the first iov entry if an array of iov entries is supplied. See sendmsg(2) for details on how iov is to be used. This pointer may be NULL if no data to be signed is available at the point of the call.

*iovlen*

[in] number of scatter/gather list elements. If iov is NULL, this value must be zero.

#### Description

A stream signing operation is started with this call. Multiple successive kcapi\_akcipher\_stream\_update function calls can be invoked to send more plaintext data to be signed. The last invocation to supply data must be done with kcapi\_akcipher\_stream\_update\_last. The kernel buffers the input until kcapi\_akcipher\_stream\_op picks up the signed data. Once plaintext is signed during the kcapi\_cipher\_stream\_op it is removed from the kernel buffer.

The function calls of kcapi\_akcipher\_stream\_update and kcapi\_akcipher\_stream\_op can be mixed, even by multiple threads of an application.

*return* number of bytes sent to the kernel upon success; a negative errno-style error code if an error occurred

## kcapi\_akcipher\_stream\_init\_vfy

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_akcipher\_stream\_init\_vfy — start an signature verification operation (stream)

#### Synopsis

```
int32_t kcapi_akcipher_stream_init_vfy (struct kcapi_handle * handle,
struct iovec * iov, uint32_t iovlen);
```

#### Arguments

*handle*

[in] cipher handle

*iov*

[in] scatter/gather list with data to be verified. This is the pointer to the first iov entry if an array of iov entries is supplied. See sendmsg(2) for details on how iov is to be used. This pointer may be NULL if no data to be verified is available at the point of the call.

*iovlen*

[in] number of scatter/gather list elements. If iov is NULL, this value must be zero.

#### Description

A stream signature verification operation is started with this call. Multiple successive kcapi\_akcipher\_stream\_update function calls can be invoked to send more plaintext data to be verified. The last invocation to supply data must be done with kcapi\_akcipher\_stream\_update\_last. The kernel buffers the input until kcapi\_akcipher\_stream\_op picks up the verified data. Once plaintext is verified during the kcapi\_cipher\_stream\_op it is removed from the kernel buffer.

The function calls of kcapi\_akcipher\_stream\_update and kcapi\_akcipher\_stream\_op can be mixed, even by multiple threads of an application.

*return* number of bytes sent to the kernel upon success; a negative errno-style error code if an error occurred

## kcapi\_akcipher\_stream\_update

### LINUX

## Name

`kcapi_akcipher_stream_update` — send more data for processing (stream)

## Synopsis

```
int32_t kcapi_akcipher_stream_update (struct kcapi_handle * handle,
struct iovec * iov, uint32_t iovlen);
```

## Arguments

*handle*

[in] cipher handle

*iov*

[in] scatter/gather list with data to be processed by the cipher operation.

*iovlen*

[in] number of scatter/gather list elements.

## Description

Using this function call, more data can be submitted to the kernel.

This function may cause the caller to sleep if the kernel buffer holding the data is getting full. The process will be woken up once more buffer space becomes available by calling `kcapi_akcipher_stream_op`.

## Note

with the separate API calls of `kcapi_akcipher_stream_update` and `kcapi_akcipher_stream_op` a multi-threaded application can be implemented where one thread sends data to be processed and one thread picks up data processed by the cipher operation.

## WARNING

The memory referenced by *iov* is not accessed by the kernel during this call. The memory is first accessed when `kcapi_cipher_stream_op` is called. Thus, you MUST make sure that the referenced memory is still present at the time `kcapi_cipher_stream_op` is called.

*return* number of bytes sent to the kernel upon success; a negative errno-style error code if an error occurred

## `kcapi_akcipher_stream_op`

## LINUX

libkcapidecember 2020

**Name**`kcapi_akcipher_stream_op` — obtain processed data (stream)**Synopsis**

```
int32_t kcapi_akcipher_stream_op (struct kcapi_handle * handle,
struct iovec * iov, uint32_t iovlen);
```

**Arguments***handle*

[in] cipher handle

*iov*

[in/out] scatter/gather list pointing to buffers to be filled with the resulting data from a cipher operation.

*iovlen*

[in] number of scatter/gather list elements.

**Description**

This call can be called interleaved with `kcapi_akcipher_stream_update` to fetch the processed data.

This function may cause the caller to sleep if the kernel buffer holding the data is empty. The process will be woken up once more data is sent by calling `kcapi_cipher_stream_update`.

Note, when supplying buffers that are not multiple of block size, the buffers will only be filled up to the maximum number of full block sizes that fit into the buffer.

*return* number of bytes obtained from the kernel upon success; a negative errno-style error code if an error occurred

**Key Protocol Primitives API - Generic****kcapi\_kpp\_init****LINUX**

libkcapidecember 2020

**Name**`kcapi_kpp_init` — initialize cipher handle

## Synopsis

```
int kcapi_kpp_init (struct kcapi_handle ** handle, const char *  
ciphername, uint32_t flags);
```

## Arguments

*handle*

[out] cipher handle filled during the call

*ciphername*

[in] kernel crypto API cipher name as specified in /proc/crypto

*flags*

[in] flags specifying the type of cipher handle

## Description

This function provides the initialization of a KPP cipher handle and establishes the connection to the kernel.

On success, a pointer to `kcapi_handle` object is returned in `*handle`. Function `kcapi_kpp_destroy` should be called afterwards to free resources.

*return* 0 upon success; -ENOENT - algorithm not available; -EOPNOTSUPP - AF\_ALG family not available; -EINVAL - accept syscall failed -ENOMEM - cipher handle cannot be allocated

## kcapi\_kpp\_destroy

### LINUX

libkcapi Manual December 2020

## Name

`kcapi_kpp_destroy` — close the cipher handle and release resources

## Synopsis

```
void kcapi_kpp_destroy (struct kcapi_handle * handle);
```

## Arguments

*handle*

[in] cipher handle to release

## kcapi\_kpp\_dh\_setparam\_pkcs3

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_kpp\_dh\_setparam\_pkcs3 — set the PG parameters using PKCS3 format

#### Synopsis

```
int kcapi_kpp_dh_setparam_pkcs3 (struct kcapi_handle * handle, const
uint8_t * pkcs3, uint32_t pkcs3len);
```

#### Arguments

*handle*

[in] cipher handle

*pkcs3*

[in] parameter buffer in DER format

*pkcs3len*

[in] length of key buffer

#### Description

With this function, the caller sets the PG parameters for subsequent cipher operations.

The parameter set must be in DER format as follows

SEQUENCE { prime INTEGER ({ dh\_get\_p }), base INTEGER ({ dh\_get\_g }) }

The following command generates such parameter set where the output

#### file content is has the correct DER structure

```
openssl dhparam -outform DER -out dhparam.der 2048
```

Note, this function defines that the subsequent key generation and shared secret operation performs an FFC Diffie-Hellman operation.

After the caller provided the key, the caller may destroy the parameter as it is now maintained by the kernel.

*return* upon success the value of the maximum size for the KPP operation is returned (e.g. the prime size); a negative errno-style error code if an error occurred

## kcapi\_kpp\_ecdh\_setcurve

**LINUX**

libkcapi Manual December 2020

### Name

kcapi\_kpp\_ecdh\_setcurve — set the ECC curve to be used for ECDH

### Synopsis

```
int kcapi_kpp_ecdh_setcurve (struct kcapi_handle * handle, unsigned
long curve_id);
```

### Arguments

*handle*

[in] cipher handle

*curve\_id*

[in] ID of the ECC curve

### Description

With this function, the caller sets the ECC curve for subsequent cipher operations. The curve ID is one of the ECC\_CURVE\_\* identifiers.

Note, this function defines that the subsequent key generation and shared secret operation performs an ECC Diffie-Hellman operation.

*return* 0 upon success; a negative errno-style error code if an error occurred

## kcapi\_kpp\_setkey

**LINUX**

libkcapi Manual December 2020

### Name

kcapi\_kpp\_setkey — set the private key of the DH / ECDH operation

### Synopsis

```
int kcapi_kpp_setkey (struct kcapi_handle * handle, const uint8_t *
key, uint32_t keylen);
```



## Arguments

*handle*

[in] cipher handle

*key*

[in] key buffer

*keylen*

[in] length of key buffer

## Description

With this function, the caller sets the key for subsequent DH / ECDH public key generation or shared secret generation operations.

If the key / keylen is zero, the kernel tries to generate the private key itself and retains it internally. This is useful if the DH / ECDH operation shall be performed on ephemeral keys where the caller is only interested in eventually obtain the shared secret.

After the caller provided the key, the caller may securely destroy the key as it is now maintained by the kernel.

Note, the key can only be set after the DH parameters or the ECC curve has been set.

*return* in case of success a positive integer is returned that denominates the maximum output size of the cryptographic operation -- this value must be used as the size of the output buffer for one cryptographic operation); a negative errno-style error code if an error occurred -- the error -EOPNOTSUPP is returned in case a kernel-triggered private key generation is requested, but the underlying cipher implementation does not support this operation.

## Synchronous Key Protocol Primitives API - One Shot

### kcapi\_kpp\_keygen

#### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_kpp\_keygen — generate a public key

#### Synopsis

```
int32_t kcapi_kpp_keygen (struct kcapi_handle * handle, uint8_t *
pubkey, uint32_t pubkeylen, int access);
```

## Arguments

*handle*

[in] cipher handle

*pubkey*

[out] generated public key

*pubkeylen*

[in] length of key buffer

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

*return* number of bytes returned by the key generation operation upon success; a negative errno-style error code if an error occurred

## kcapi\_kpp\_ssgen

### LINUX

libkcapi Manual December 2020

### Name

kcapi\_kpp\_ssgen — generate a shared secret

### Synopsis

```
int32_t kcapi_kpp_ssgen (struct kcapi_handle * handle, const
uint8_t * pubkey, uint32_t pubkeylen, uint8_t * ss, uint32_t sslen,
int access);
```

## Arguments

*handle*

[in] cipher handle

*pubkey*

[in] public key of peer that shall be used to generate the shared secret with

*pubkeylen*

[in] length of the public key buffer

*ss*

[out] generated shared secret

*sslen*

[in] length of key buffer

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

**Description**

*return* number of bytes returned by the shared secret generation operation upon success; a negative errno-style error code if an error occurred

**Asynchronous Key Protocol Primitives API - One Shot****kcapi\_kpp\_keygen\_aio****LINUX**

libkcapi Manual December 2020

**Name**

kcapi\_kpp\_keygen\_aio — generate a public key (asynchronous one shot)

**Synopsis**

```
int32_t kcapi_kpp_keygen_aio (struct kcapi_handle * handle, struct
iovec * outiov, uint32_t iovlen, int access);
```

**Arguments***handle*

[in] cipher handle

*outiov*

[out] head of scatter-gather list of the destination buffers filled with the generated public key

*iovlen*

[in] number of scatter-gather list entries

*access*

[in] kernel access type (KCAPI\_ACCESS\_HEURISTIC - use internal heuristic for fastest kernel access; KCAPI\_ACCESS\_VMSPLICE - use vmsplice access; KCAPI\_ACCESS\_SENDMSG - sendmsg access)

## Description

The individual scatter-gather list entries are processed with separate invocations of the the given cipher.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the `vmsplice` call may not send all data to the kernel.

*return* number of bytes verify upon success; a negative `errno`-style error code if an error occurred

## kcapi\_kpp\_ssgen\_aio

### LINUX

libkcapi Manual December 2020

## Name

`kcapi_kpp_ssgen_aio` — generate a shared secret (asynchronous one shot)

## Synopsis

```
int32_t kcapi_kpp_ssgen_aio (struct kcapi_handle * handle, struct
iovec * iniov, struct iovec * outiov, uint32_t iovlen, int access);
```

## Arguments

*handle*

[in] cipher handle

*iniov*

[in] head of scatter-gather list of the source buffers with the public keys of the peer

*outiov*

[out] head of scatter-gather list of the destination buffers filled with the generated shared secret

*iovlen*

[in] number of scatter-gather list entries

*access*

[in] kernel access type (`KCAPI_ACCESS_HEURISTIC` - use internal heuristic for fastest kernel access; `KCAPI_ACCESS_VMSPLICE` - use `vmsplice` access; `KCAPI_ACCESS_SENDMSG` - `sendmsg` access)

## Description

The individual scatter-gather list entries are processed with separate invocations of the the given cipher.

The memory should be aligned at the page boundary using `posix_memalign(sysconf(_SC_PAGESIZE))`. If it is not aligned at the page boundary, the `vmsplce` call may not send all data to the kernel.

*return* number of bytes verify upon success; a negative `errno`-style error code if an error occurred

## Key Derivation Functions

API function calls used to invoke a KDF. The KDF functions are based on a message digest or keyed message digest function. The caller must have the handle allocated with `kcapi_md_init`. If the caller wishes to use a keyed message digest, the caller must invoke `kcapi_md_setkey` before those functions.

### kcapi\_kdf\_dpi

#### LINUX

libkcapi Manual December 2020

#### Name

`kcapi_kdf_dpi` — Double Pipeline Mode Key Derivation Function

#### Synopsis

```
int32_t kcapi_kdf_dpi (struct kcapi_handle * handle, const uint8_t
* src, uint32_t slen, uint8_t * dst, uint32_t dlen);
```

#### Arguments

*handle*

[in] cipher handle allocated by caller. This cipher handle must be allocated with `kcapi_md_init`. If the caller is interested in a KDF using a keyed message digest, the caller should also call `kcapi_md_setkey` before invoking this function.

*src*

[in] Input data that should be transformed into a key (see below).

*slen*

[in] Length of the `src` input data.

*dst*

[out] Buffer to store the generated key in,

*dlen*

[in] Length of the `dst` buffer. This value defines the number of bytes generated by the KDF.

## Description

This function is an implementation of the KDF in double pipeline iteration mode according with counter to SP800-108 section 5.3.

The caller must provide Label || 0x00 || Context in *src*. This *src* pointer may also be NULL if the caller wishes not to provide anything.

*return* 0 upon success; a negative *errno*-style error code if an error occurred

## kcapi\_kdf\_fb

### LINUX

libkcapi Manual December 2020

## Name

*kcapi\_kdf\_fb* — Feedback Mode Key Derivation Function

## Synopsis

```
int32_t kcapi_kdf_fb (struct kcapi_handle * handle, const uint8_t *
src, uint32_t slen, uint8_t * dst, uint32_t dlen);
```

## Arguments

*handle*

[in] cipher handle allocated by caller. This cipher handle must be allocated with *kcapi\_md\_init*. If the caller is interested in a KDF using a keyed message digest, the caller should also call *kcapi\_md\_setkey* before invoking this function.

*src*

[in] Input data that should be transformed into a key (see below).

*slen*

[in] Length of the *src* input data.

*dst*

[out] Buffer to store the generated key in,

*dlen*

[in] Length of the *dst* buffer. This value defines the number of bytes generated by the KDF.

## Description

This function is an implementation of the KDF in feedback mode with a non-NULL IV and with counter according to SP800-108 section 5.2. The IV is supplied with *src* and must be equal to the digestsize of the used cipher.

In addition, the caller must provide Label || 0x00 || Context in *src*. This *src* pointer must not be NULL as the IV is required. The ultimate format of the *src*

pointer is IV || Label || 0x00 || Context where the length of the IV is equal to the block size (i.e. the digest size of the underlying hash) of the PRF.

*return* 0 upon success; a negative errno-style error code if an error occurred

## kcapi\_kdf\_ctr

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_kdf\_ctr — Counter Mode Key Derivation Function

#### Synopsis

```
int32_t kcapi_kdf_ctr (struct kcapi_handle * handle, const uint8_t
* src, uint32_t slen, uint8_t * dst, uint32_t dlen);
```

#### Arguments

*handle*

[in] cipher handle allocated by caller. This cipher handle must be allocated with `kcapi_md_init`. If the caller is interested in a KDF using a keyed message digest, the caller should also call `kcapi_md_setkey` before invoking this function.

*src*

[in] Input data that should be transformed into a key (see below).

*slen*

[in] Length of the src input data.

*dst*

[out] Buffer to store the generated key in,

*dlen*

[in] Length of the dst buffer. This value defines the number of bytes generated by the KDF.

#### Description

This function is an implementation of the KDF in counter mode according to SP800-108 section 5.1 as well as SP800-56A section 5.8.1 (Single-step KDF).

SP800-108: The caller must provide Label || 0x00 || Context in `src`. This `src` pointer may also be NULL if the caller wishes not to provide anything.

SP800-56A: If a keyed MAC is used, the key shall NOT be the shared secret from the DH operation, but an independently generated key. The `src` pointer is defined as Z || other info where Z is the shared secret from DH and other info is an arbitrary string (see SP800-56A section 5.8.1.2).

*return* 0 upon success; a negative errno-style error code if an error occurred

## kcapi\_pbkdf

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_pbkdf — Password-based Key Derivation Function

#### Synopsis

```
int32_t kcapi_pbkdf (const char * hashname, const uint8_t * pw,  
uint32_t pwlen, const uint8_t * salt, uint32_t saltlen, uint32_t  
count, uint8_t * key, uint32_t keylen);
```

#### Arguments

*hashname*

[in] kernel crypto API name of a keyed hash (e.g. hmac(sha1))

*pw*

[in] Password a key shall be derived from

*pwlen*

[in] Length of password string

*salt*

[in] Salt as defined in SP800-132

*saltlen*

[in] Length of salt buffer

*count*

[in] Numbers of iterations to be performed for the PBKDF

*key*

[out] Buffer to store the generated key in

*keylen*

[in] Size of the key to be generated (i.e. length of the key buffer)

#### Description

This function is an implementation of the PBKDF as defined in SP800-132.

*return* 0 upon success; a negative errno-style error code if an error occurred



## kcapi\_pbkdf\_iteration\_count

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_pbkdf\_iteration\_count — Calculate numbers of iterations for a PBKDF

#### Synopsis

```
uint32_t kcapi_pbkdf_iteration_count (const char * hashname, uint64_t
timeshresh);
```

#### Arguments

*hashname*

[in] kernel crypto API name of a keyed hash (e.g. hmac(sha1))

*timeshresh*

[in] Time duration in nanoseconds that the PBKDF operation shall at least require. If that value is 0, a default of (1<<27) nanoseconds is used.

#### Description

The function measures the time the PBKDF operation takes for different round counts for the given keyed message digest type.

The result should be taken as the iteration count for a PBKDF operation.

If an error occurs with the PBKDF calculation, a value of 1<<18 is returned.

*return* number of iterations a PBKDF should take on this computer.

## kcapi\_hkdf

### LINUX

libkcapi Manual December 2020

#### Name

kcapi\_hkdf — Extract-and-Expand HKDF (RFC5869)

#### Synopsis

```
int32_t kcapi_hkdf (const char * hashname, const uint8_t * ikm,
uint32_t ikmlen, const uint8_t * salt, uint32_t saltlen, const
uint8_t * info, uint32_t infoflen, uint8_t * dst, uint32_t dlen);
```

## Arguments

*hashname*

[in] kernel crypto API name of a keyed hash (e.g. hmac(sha1))

*ikm*

[in] Input Keying Material (IKM) -- must be provided

*ikmlen*

[in] IKM buffer length -- must be non-zero

*salt*

[in] salt buffer -- may be NULL

*saltlen*

[in] salt buffer length -- may be zero

*info*

[in] info buffer -- may be NULL

*info len*

[in] info buffer length -- may be zero

*dst*

[out] Buffer to store the generated key in,

*dlen*

[in] Length of the dst buffer. This value defines the number of bytes generated by the KDF.

## Description

Perform the key-derivation function according to RFC5869. The input data is defined in sections 2.2 und 2.3 of RFC5869.

*return* 0 upon success; a negative errno-style error code if an error occurred