BroadSAFE MicroHSM Software Reference

Section 1: Introduction

1.1 Scope

This document describes the BroadSAFE Micro Hardware Security Module[HSM] application programming interface and device driver architecture for the Broadcom microHSM module.

MicroHSM is used for generating and managing secure keys and decrypting messages with the keys from within the hardware security module.

In particular, this document describes the microHSM API specifically targeted to the features of the Broadcom Micro Hardware Security Module core.

The library is intended for use by the microHSM key delivery protocol[KDP] application developers and for implementing secure key management for cryptographic key based applications.

1.2 Overview

This microHSM application programming Interface library and driver source code is provided by Broadcom Corporation in C-language source form. The driver/library may be used both as a reference for programming the microHSM core or for directly accessing the device from user-level applications as a library. The library is designed to facilitate rapid development of prototype for key based applications such as nCipher key delivery protocol and other secure key based applications.

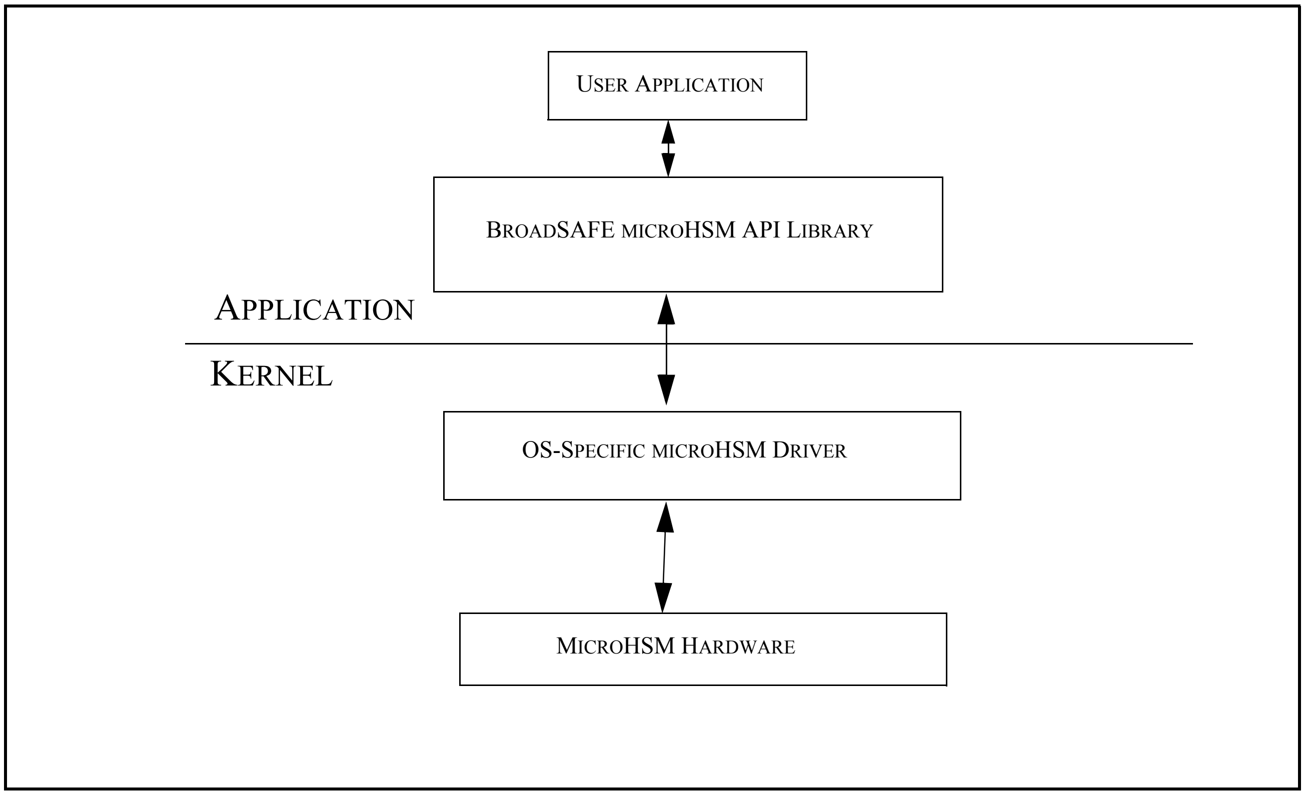


Figure 1: Driver/Library/User Application Relationship

1.3 Using The MicroHSM API Library

1.3.1 HEADER FILES

Include the header file bsafe_uhsm_lib.h in all applications which require uses the microHSM library.

bsafe_uhsm_lib.h is the master header file containing the API definitions, the device accessing functions, and their return codes.

Additional header bsafe_common.h included in bsafe_lib.h is required.

bsafe_common.h also includes the following headers files.

bsafe_vers.h which contains the version of the driver.

bsafe_returncode.h which contains the chip return codes.

bsafe_status.h which contains the ioctl return codes.

bsafe_devname.h which contains the specific device name to be opened to access the driver.

1.3.2 PROGRAMMING MODEL

The library interface for all calls is synchronous and causes the calling application to block until it returns. Most calls map to underlying **ioctl** calls.

In general, applications perform the following operations.

Open the device using **bsafe_open**.

Call the desired function or functions.

Close the device using **bsafe_close**.

1.3.3 SPECIAL DATA TYPES

BSAFE_BIGNUM (BroadSAFE multiple precision number)

The multiple precision number is an array of 32-bit integer values, where the low-ordered 32 bit integer value (array index 0) contains the least-significant multiple precision integer and the high-ordered 32 bit integer value (array index N) contains the most-significant multiple precision integer. Where each 32bit integer is represented as the low-ordered byte(array index 0) is the most-significant byte and the high-ordered byte(array index 4) is the least-significant byte.

In other words the BSAFE_BIGNUM is an array of 32 bit integers(Words) ordered from Least-Significant-Word to Most-Significant-Word, where each Word is ordered as Most-Significant byte to Least-Significant byte.

For example:

Consider an array of 8 bytes from a[0] to a[7]. where a[0] contains the Most Significant Byte and a[7] contains the Least Significant Byte.

Normal format/network-byte order[binarray] of a sample multi precision number array is as below

a[8] ={0x8,0x7,0x6,0x5,0x4,0x3,0x2,0x1} i.e. a[0]= 0x8(MSB).... a[7]=0x1(LSB)

where the bytes are ordered from Most-Significant Byte(0x8) to Least-Significant Byte(0x1) and integer[words] array is ordered as Most-significant word[MSW](0x08070605) to Least-significant word[LSW](0x04030201).

BSAFE_BIGNUM format would be as follows

BSAFE_BIGNUM[8]={0x4,0x3,0x2,0x1,0x8,0x7,0x6,0x5} i.e. a[0]=0x4 ...a[7]=0x5

where the integer[words] array are ordered from Least-significant word[LSW](0x04030201) to Most-significant word[MSW](0x08070605).

Note: Use bsafe_test_set_bsafe_bignum_endianess for using various host application bignums to be passed in to the library for BSAFE_BIGNUM.

Note: Byte streams

All byte streams are expected to be in Normal format(Network byte order) that is Most-significant byte[MSB] to Least-significant byte[LSB].

1.4 Reference Documents

Micro Hardware Security Module Specification

bsafe_uHSM source code.

Section 2: MicroHSM API Functions

This section describes the API calls provided by the BroadSAFE microHSM Library.

Return Values

Return Values	The return value indicates the return status of the operation. The API's returns, the retuncode of the device if it is not success(UHSM_SUCCESS) or the return status of the driver. Returncode of the device is defined in bsafe_returncode.h Returnstatus of the driver is defined in bsafe_status.h

2.1 Device Open/Close Functions

These functions provided by the library to open and close the micro hardware security module driver.

2.1.1 BSAFE_OPEN

This function opens the specified microHSM device for operation. The device name as defined in bsafe_devname.h is passed as a parameter. This return a valid file descriptor(fd) for use with future function calls.

Function	**bsafe_open**
Prototype	**int bsafe_open(char * devname);**
Parameters	*devicename* The device name as defined in bafe_devname.h.
Return Values	The return value indicates the success or failure of the open request. It can be any of the following values On success: The new file descriptor [fd] is returned. This would be a non negative integer. On Failure -1

2.1.2 BSAFE_CLOSE

This function closes the file descriptor opened using bsafe_open.

Function	**bsafe_close**
Prototype	**int bafe_close(int fd)**
Parameters	*fd* The file descriptor to be closed.

Return Values

The return value indicates the success or failure of the close request. It can be any of the following values

On Success

0 is returned.

On failure

-1

2.2 Device Initialization Functions

The initialization functions are used to initialize the Non volatile memory(NVM) with device specif keys and manufacturer specific keys. These functions are used during device initialization or by manufacturer for storing keys.

The non volatile memory is also referred as one-time programmable(OTP) memory.

2.2.1 BSAFE_INIT_DEVICEKEY

This function initializes the hardware module with the required device specific keys and the product specific configuration values. This function call is called once before shipping the product with the product config, uhsm config and other one time programmable values to be fixed in the non volatile memory.

.

Function	**bsafe_init_devicekey**
Prototype	**int bsafe_init_devicekey (int** fd, **unsigned int** uhsm_config, **unsigned char** *prod_config, **unsigned char** *auth_digest, **unsigned short** num_row_retry, **unsigned short** num_col_retry, **unsigned char** *statistics)

Parameters

fd

File descriptor got using the bsafe_open library call.

uhsm_config

A bit array for various configuration options of the micro-HSM. The various bits which can be set are as defined in the enumerated array bsafe_init_devicekey_uhsmConfig_bits_t (bsafe_common.h).

The various bit representation are

DC_EXPORT - Export mode (where all 3DES_HMAC command of bsafe_kdp_decrypt_des3 is converted to a simple DES operation).

DC_DIS_USERSTATUS - The command user status(bsafe_user_status) is disabled.

DC_DIS_RAWRNG - Disable raw random number output to be used directly as output data (bsafe_user_random).

DC_MODSIZE_1024 - The DH session key size is 1024 for bsafe_kdp_start.

DC_RUNSELFTEST - enables self test(bsafe_selftest) at power up or hardware reset.

DC_ENABLE_USERKEY - bsafe_ld_userkey is allowed

DC_DIS_USERAUTH - bsafe_kpd_userauth_sha1pp is disabled

DC_USERAUTH_SHA1 - A SHA1 hash operation is performed on the GPIO UserID input.

DC_AUTH_VALID - bsafe_ld_cfg command is enabled.

SC_EN_LOWFREQ - Enable low frequency monitor

SC_EN_RSTMON - Enable reset monitor

SC_EN_JMPMON - Enable jump monitor

SC_EN_EXCPTMON - Enable exception monitor

prod_config

The 4 byte[32 bit value] is programmed into the non-volatile memory.

auth_digest

The 20 byte SHA1 digest of the manufacturer configuration public key. Or these bytes can be used to store any local information in the non-volatile memory.

num_row_retry

The number of retries to be performed on programming the OTP on a row programming failure.

num_col_retry

The number of retries to be performed on programming the OTP on a column programming failure.

statistics

Pointer to return the 16 byte statistics from the non volatile memory programming logic.

Return Values	As described in the return value section. Special return codes OTP_PREV_PROG_ERR - Non volatile memory previously programmed.

2.2.2 BSAFE_FIPS_CLEAR

This function destructively disables the device. This clears the key information stored in the micro-HSM.

Warning Note: This command is destructive and spoils the functionality of the chip. The device cannot be used after this function.

Function	**bsafe_fips_clear**
Prototype	**int bsafe_fips_clear (int** fd**,** **unsigned int** clr_vector **unsigned short** num_retry)
Parameters	*fd* File descriptor got using the bsafe_open library call. *clr_vector* The value BSAFE_FIPS_CLR_VECTOR as defined in bsafe_common.h has to be passed in this argument. *num_retry* The number of retries to be performed while programming the OTP.
Return Values	As described in the return value section.

2.2.3 BSAFE_SELFTEST

This function can only be called as the first command after a hardware reset and before any other commands are issued to the micro-HSM. If the DC_RUNSELFTEST is set in uhsm_config in the bsafe_init_devicekey command, the tests are not run but it returns the result of the automatic self-test which was run during reset.

If any of the self-tests fail, no further command processing can occur except bsafe_selftest. The result of executing the command leaves the micro-HSM equivalent to that of a hardware reset.

Note:

The loading of micro-HSM driver may by default execute a few commands so the application's first command does not mean it is the first command for the chip.

Function	**bsafe_selftest**

Prototype

```
int bsafe_selftest(int fd,
                   unsigned int runtest,
                   unsigned int * testresult)
```

Parameters

fd

File descriptor got using the bsafe_open library call.

runtest

The bit array for various tests to be done. The various bits which can be set are as defined in the enumerated array bsafe_selftest_bitvector_bits_t (bsafe_common.h).

RAM_BIST_ROM_BIST_TEST - on-chip RAM/ROM BIST

NVM_CHECKSUM_VERIFY_TEST - non-volatile memory checksum verify

HMAC_SHA1_ENGINE_TEST - hmac-SHA1 engine known value test

DES3_ENGINE_TEST - 3DES engine known value test

FIPS186_2_PSEUDO_RANDOM_TEST - pseudo random number with known seed defined in FIPS186-2 appendix 3

DSA_SIGN_VERIFY_TEST - DSA sign/verify pairwise test on known data

DH_TEST - DH known value test

RSA_ENC_DEC_TEST - RSA encrypt/decrypt on known value

DSA_SIGN_VERFIY_KDI_TEST - DSA sign/verify on known values

FIPS140_2_RANDOM_NUM_TEST - Random number continuos test defined by FIPS140-2.

RANDOM_NUM_STAT_TEST - random number statistical test

testresult

Pointer to return 4 byte selftest result. The lower 16 bit array is a bit mask of all tests passed as per the *runtest* argument.

Return Values

As described in the return value section.

2.2.4 BSAFE_LD_CFG

This function is used to load the configuration using a DLIES message. The manufacturer DSA public key(KM_PUB) is used to verify the signature.

Function	**bsafe_ld_cfg**
Prototype	**int** **bsafe_ld_cfg (int** fd, **unsigned char** *keypolicy, **unsigned short** keytype, **bsafe_dsa_pub_t** *km_pub, **unsigned char***dlies_msg, **unsigned int** dlies_msg_len, **BSAFE_BIGNUM** *authsig_r, **BSAFE_BIGNUM** *authsig_s);

Parameters

fd

File descriptor got using the bsafe_open library call.

keypolicy

Pointer to a 6 byte key policy of the km_pub key.

keytype

Key type of the km_pub key.

km_pub

Pointer to the manufacturer DSA public key in bsafe_dsa_pub_t format.

The values are in network byte order.

struct bsafe_dsa_pub_t is defined in bsafe_common.h.

The values p, g, q. and y are in network byte order MSB to LSB

typedef struct bsafe_dsa_pub {

unsigned char * p;/* 128 bytes */

unsigned char * g;/* 128 bytes */

unsigned char * q;/* 20 bytes */

unsigned char * y;/* 128 bytes */

} bsafe_dsa_pub_t;

dlies_msg

Pointer to the encrypted DLIES message.

dlies_msg_len

Length of dlies_msg.

authsig_r

Pointer to 20 byte authentication signature r of the message.

authsig_s

Pointer to 20 byte authentication signature s of the message.

Return Values

As described in the return value section.

2.3 MicroHSM Functions

These microHSM functions provide the capability for device key access, key processing, DLIES message processing and packet/message decryption.

Note:

The id's(sessionid, appkeyid) referred by all microHSM functions relate directly to key locations in the device.

2.3.1 BSAFE_UHSM_GETPUBKEYS

This function is used to get the device public keys. This function returns the device identity pubic key (KDI_PUB) value and the device confidentiality public key (KDC_PUB) value.

The KDI Public value is the DSA public value of 128 byte p, 128 byte g, 20 byte q and 128 byte y.

The KDC public value is the 256 byte DH public value.

Function	**bsafe_uhsm_getpubkey**
Prototype	**int** **bsafe_uhsm_getpubkey(int fd,** **unsigned char** *kdc_pub_out, **bsafe_dsa_pub_t** *kdi_pub_out);
Parameters	*fd* File descriptor got using the bsafe_open library call. *kdc_pub_out* Pointer to return the 256 byte KDC_PUB value. The value is in network byte order. *kdi_pub_out* Pointer to return the 404 byte KDI_PUB value. The values are in network byte order. struct bsafe_dsa_pub_t is defined in bsafe_common.h. The values p, g, q. and y are in network byte order MSB to LSB typedef struct bsafe_dsa_pub { unsigned char * p;/* 128 bytes */ unsigned char * g;/* 128 bytes */ unsigned char * q;/* 20 bytes */ unsigned char * y;/* 128 bytes */ } bsafe_dsa_pub_t;
Return Values	As described in the return value section.

2.3.2 BSAFE_UHSM_START

This function is used to start a new session key. The signed values and the session public key is returned. This public value of the DH pair is the 256 or 128 byte session public key (KSC_PUB) depending on the setting of the DC_MODSIZE_1024 in bsafe_init_devicekey.

Warning Note: This removes the previous sesion key in the device.

Function	**bsafe_uhsm_start**
Prototype	**int** **bsafe_uhsm_start(int** fd, **unsigned char** *challenge, **unsigned short** *sessionid, **unsigned char** *ksc_pub, **BSAFE_BIGNUM** *sig_begin_r, **BSAFE_BIGNUM** *sig_begin_s) ;
Parameters	*fd* File descriptor got using the bsafe_open library call. *challenge* Pointer to a 20 byte challenge byte stream. The value is expected to be in network byte order. *sessionid* Pointer to return the 2 byte session id. ksc_pub Pointer to return the 256/128 byte ksc_pub value. The value is in network byte order. *sig_begin_r* Pointer to return the 20 byte DSA signature r. The value is in network byte order. *sig_begin_s* Pointer to return the 20 byte DSA signature s. The value is in network byte order.
Return Values	As described in the return value section.

2.3.3 BSAFE_UHSM_MESSAGE

This function is used to decrypt nested DLIES message(as defined in IEEE P1363 and sec 8.2). The associated keys decrypted from the message is stored at the returned locations.

Function	**bsafe_uhsm_message**
Prototype	**int** **bsafe_uhsm_message(int** fd, **unsigned short** *sessionids, **unsigned short** numsessionids, **unsigned char** *message, **unsigned int** message_len, **unsigned short** *appkeyid_r, **unsigned short** *sessionid_r) ;
Parameters	*fd* File descriptor got using the bsafe_open library call. *sessionids* Pointer to an array of sessionids to be used with the message decrypt. The number of elements in the array is equal to numsessionids. *numsessionids* The number of session ids in the sessionids array. *message* Pointer to the message byte stream to be decrypted. *message_len* Length of the message to be decrypted. *appkeyid_r* Pointer to return the appkeyid retrieved from the message. Pass in NULL if not used for this decrypt. *sessionid_r* Pointer to return the sessionid retrieved from the message. Pass in NULL if not used for this decrypt.

Return Values	As described in the return value section. Special error codes AKEY_BAD_LOC_ERR - not enough space in the device to hold this decrypted key. BSAFE_NO_FREE_AKEYLOC - all Akey locations are used. BSAFE_NO_FREE_KEKLOC - all kek locations are used. **Note:** use bsafe_clr_key to clear keys to make space for this key.

2.3.4 BSAFE_UHSM_USERAUTH_SHA1PP

This function is used to perform user authentication. The userID required for this SHA1-Passphrase[SHA1pp] is passed in using the gpio pins.

Note: The bsafe_uhsm_init_devicekey's uhsm_config bits DC_DIS_USERAUTH and DC_USERAUTH_SHA1 bits affects this command.

Function	**bsafe_uhsm_userauth_sha1pp**
Prototype	**int** **bsafe_uhsm_userauth_sha1pp (int** fd, **unsigned char** *challenge, **BSAFE_BIGNUM** *sig_userauth_r, **BSAFE_BIGNUM** *sig_userauth_s) ;
Parameters	*fd* File descriptor got using the bsafe_open library call. *challenge* Pointer to a 20 byte challenge byte stream. The value is expected to be in network byte order. *sig_userauth_r* Pointer to return the 20 byte DSA signature r . The value is in network byte order. *sig_userauth_s* Pointer to return the 20 byte DSA signature s . The value is in network byte order.
Return Values	As described in the return value section.

2.3.5 BSAFE_UHSM_DECRYPT_RSAPRIVATE

This function is used to decrypt a packet/message using RSA private key.

Function	**bsafe_uhsm_decrypt_rsaprivate**
Prototype	**int** **bsafe_uhsm_decrypt_rsaprivate (int** fd, **unsigned short** appkeyid, **unsigned char** *ip, **unsigned int** p_len, **unsigned char** *op);
Parameters	*fd* File descriptor got using the bsafe_open library call. *appkeyid* The application key id to be used to decrypt this packet. *ip* Pointer of the input packet to be decrypted. The message is expected to be in network byte order. *p_len* The length of the packet. *op* Pointer to return the output packet after decryption. The message is in network byte order. The length of the decrypted message is equal to the packet length(p_len).
Return Values	As described in the return value section.

2.3.6 BSAFE_UHSM_DECRYPT_DES3

This function is used to decrypt a packet/message using 3DES.

Function	**bsafe_uhsm_decrypt_rsaprivate**
Prototype	**int** **bsafe_uhsm_decrypt_rsaprivate (int** fd, **unsigned short** appkeyid, **unsigned char** *iv, **unsigned char** *ip, **unsigned int** p_len, **unsigned char** *op);
Parameters	*fd* File descriptor got using the bsafe_open library call. *appkeyid* The application key id to be used to decrypt this packet. *iv* Pointer of the 8 byte initialization vectior(iv). The message is expected to be in network byte order. *ip* Pointer of the input packet to be decrypted. The message is expected to be in network byte order. *p_len* The length of the packet. *op* Pointer to return the output packet after decryption. The message is in network byte order. The length of the decrypted message is equal to the packet length(p_len).
Return Values	As described in the return value section.

2.4 Hash Functions

This function is used to hash a message.

2.4.1 BSAFE_USER_SHA1

This function is used to generate 20 bytes of hash for a message. The function can be called with various modes to get the digest for a set of fragmented data using the various modes.

Function	**ubsec_user_sha1**
Prototype	**int** **bsafe_user_sha1 (int** fd, **unsigned short** prevlen, **unsigned short** mode, **unsigned char** *indigest, **unsigned char** data, **unsigned char** *outdigest);
Parameters	*fd* File descriptor got using the bsafe_open library call. *prevlen* Total previous length of the data included in the indigest calculation. *mode* Selected processing mode. The different modes are defined in the enumerated array bsafe_user_sha1_mode_t (basfe_common.h). The various modes are **BSAFE_MODE_SHA1_INIT** - Init (indigest is ignored. the starting point uses SHA1 constants) **BSAFE_MODE_SHA1_UPDATE** - indigest is used **BSAFE_MODE_SHA1_FINISH** - indigest is used *indigest* Pointer to the 20 bytes starting state of the hash engine. *data* Pointer to the data to be hashed *data_len* Length of the data to be hashed. *outdigest* Pointer to return the resulting 20 bytes of digest of the hash operation.
Return Values	As described in the return value section.

2.5 Key Functions

These functions are used to manipulate/load the user specified keys by the application into the hardware for various operations.

Note:

The return ksize in the key functions is the number of locations/buckets occupied by the key in the device. The size of various key locations and buckets is in bsafe_keyloc_properties_t(bsafe_common.h).

2.5.1 BSAFE_KEK_STATUS

This function is used to query the status of the kek location.

Function	**bsafe_kek_status**
Prototype	**int** **bsafe_kek_status (int** fd, **unsigned short** kekloc, **unsigned char** *keypolicy, **unsigned short** *keytype, **unsigned int** *ksize);
Parameters	*fd* File descriptor got using the bsafe_open library call. *kekloc* The kek location whose status to query. *keypolicy* Pointer to return the 6 byte key policy information. The policy information is in network byte order. *keytype* Pointer to return the 2 byte key type information. *ksize* Pointer to return the number of key locations used by this key.
Return Values	As described in the return value section.

2.5.2 BSAFE_AKEY_STATUS

This function is used to query the status of the application key location.

Function	**bsafe_akey_status**
Prototype	**int** **bsafe_akey_status (int** fd, **unsigned short** akeyloc, **unsigned char** *keypolicy, **unsigned short** *keytype, **unsigned int** *ksize);
Parameters	*fd* File descriptor got using the bsafe_open library call. *akeyloc* The akey location whose status to query. *keypolicy* Pointer to return the 6 byte key policy information. The policy information is in network byte order. *keytype* Pointer to return the 2 byte key type information. *ksize* Pointer to return the number of key locations used by this key.
Return Values	As described in the return value section.

2.5.3 BSAFE_LD_USERKEY

This function is used to load the application key to a specified application key location.

Function	**bsafe_ld_userkey**
Prototype	**int** **bsafe_ld_userkey (int** fd, **unsigned short** akeylocation, **unsigned char** *keydata, **unsigned int** keydata_len, **unsigned int** *ksize);

Parameters	*fd* File descriptor got using the bsafe_open library call. *akeylocation* The first key location where the application key is to be loaded. *keydata* Pointer of the application key to be loaded. The key data format is as specified in the data sheet (TBD: add it to this document) *keydata_len* Length of keydata. *ksize* Pointer to return the number of key locations used by this key.
Return Values	As described in the return value section.

2.5.4 BSAFE_CLR_KEY

This function is used to clear the key in the from the device.

Function	**bsafe_clr_key**
Prototype	**int** **bsafe_clr_key (int** fd, **unsigned short** location, **unsigned short** cache, **unsigned int** *ksize);
Parameters	*fd* File descriptor got using the bsafe_open library call. *location* The first location of the key. *cache* The type of key. The types are defined in the enumerated array bsafe_clr_key_cache_t (bsafe_common.h). **BSAFE_KEK_CACHE** - for clearing a kek key **BSAFE_AKEY_CACHE** - for clearing an application key *ksize* Pointer to return the number of key locations used by this key.
Return Values	As described in the return value section.

2.6 Random Number Generation Functions

This function provides the capability to generate random number for various cryptographic applications.

2.6.1 BSAFE_USER_RANDOM

This function generates a 20 bytes of true random number

Function	**ubsec_user_random**
Prototype	**int** **bsafe_user_random (int** fd, **unsigned int** rngtype, **unsigned char** *random);
Parameters	*fd* File descriptor got using the bsafe_open library call. *rngtype* The type of random to be created. The types are defined in the enumerated array bsafe_user_rngtype_t (bsafe_common.h). **BSAFE_USER_RANDOM_RNGTYPE_RAW** - raw random number **BSAFE_USER_RANDOM_RNGTYPE_X** - random number processed according to FIPS186-2 ‘x’ generation **BSAFE_USER_RANDOM_RNGTYPE_K** - random number processed according to FIPS168-2 ‘k’ generation *random* Pointer to return the 20 byte random.
Return Values	As described in the return value section.

2.7 SAL Status Functions

This function gets the Security Assurance Logic(SAL) status of the device. (TBD from data sheet the interpretation of it)

2.7.1 BSAFE_USER_STATUS

This function is used to get the SAL status of the device from the SAL event register.

Function	**ubsec_user_status**
Prototype	**int** **bsafe_user_status (int** fd, **unsigned int** *SALeventstatus);
Parameters	*fd* File descriptor got using the bsafe_open library call. *SALeventstatus* Pointer to return the SAL register status of the device
Return Values	As described in the return value section.

2.8 NOP Functions

This function is used to pass in a no-operation command.

2.8.1 BSAFE_USER_NOP

This function is used to test a pass through of a data stream through the device.

Function	**ubsec_user_nop**
Prototype	**int** **bsafe_user_nop (int** fd, **unsigned char** *datain, **unsigned int** data_len, **unsigned char** *dataout);

Parameters	*fd* File descriptor got using the bsafe_open library call. *datain* Pointer to the data to pass through the device *data_len* Length of the data *dataout* Pointer to return the data passed to the device
Return Values	As described in the return value section.

2.9 MISC FUNCTIONS

This function is used to set various properties of the driver for various modes of operation.

2.9.1 BSAFE_TEST_SET_BSAFE_BIGNUM_ENDIANESS

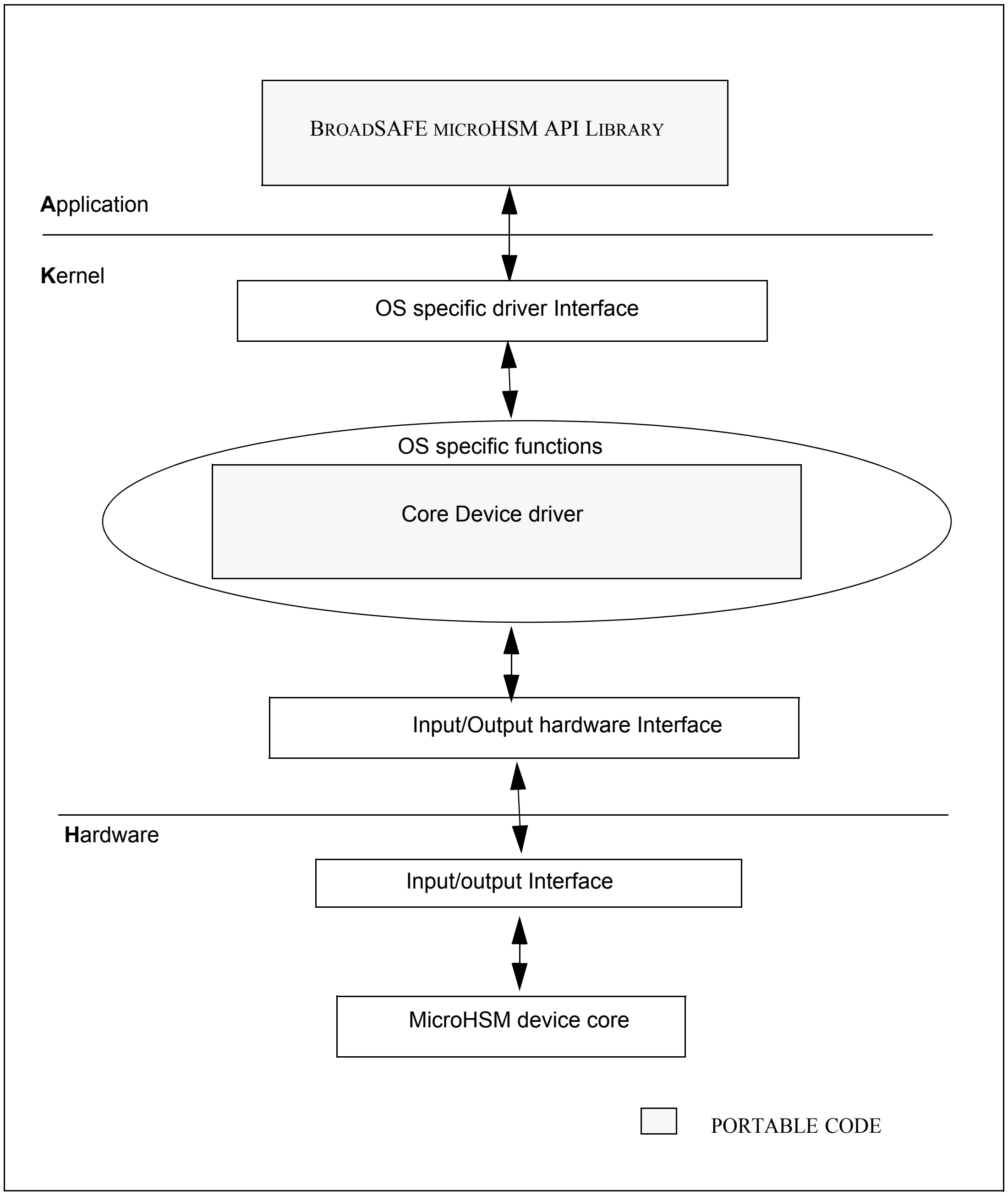
This function is used to test and set the BSAFE_BIGNUM endianess based on the given bignum array. This affects the entire driver operation.

Function	**bsafe_test_set_bsafe_bignum_endianess**
Prototype	**int** **bsafe_test_set_bsafe_bignum_endianess (int** fd, **BSAFE_BIGNUM** *one);
Parameters	*fd* File descriptor got using the bsafe_open library call. *one* Pointer to the host application bignum which contains the multi-precision number (0x0807060504030201 where 0x08 is the MSB and 0x01 is the LSB)
Return Values	As described in the return value section. On Unknown bignum format: -1

Section 3: MicroHSM Device Driver

This section specifies the microHSM device driver architecture and various portable modules for different IO interfaces and different operating systems driver architectures.

The microHSM core device driver is modularized for operating system portability and portability for different input/output interface implementations.



3.1 Core Device Driver

This is the core functions which directly map to the capabilities and commands of the microHSM device.

The driver code is written in C-language.

3.1.1 CORE DRIVER PORTING

These macro's are used by the core driver for various operating system specific functions. The code can be ported to different platforms with the porting of the following macros. These functions/macros are defined in the OS independent Layer(OSL) header bsafe_osl.h.

OSL macros	*function*
OS_DMA_ALLOC	Allocate DMA memory
OS_DMA_FREE	Free DMA memory
OS_ALLOC	Allocate memory
OS_FREE	Free memory
OS_COPY_FROM_USER	Copy from user space to kernel space
OS_COPY_TO_USER	Copy from kernel space to user space
OS_MEMCPY	memory copy
OS_MEMSET	memory set
MAX_TIMEOUT	Maximum delay time out for a blocking ioctl call for processing of a command
DELAY	Delay function for microseconds
DELAY_US	Minimum delay seconds between device status polling.
PRINTF	Display of error/log messages

The base data types used for the device core definitions.

OSL macros	*function*
BYTE	8 bit (byte) data type
UINT16	16 bit data type
UINT32	32 bit data type
INT	signed 32 bit data type

3.2 OS Specific Driver

This wrapper driver probes and finds the microHSM core and initializes the device and provides input/output driver interface to the core driver.

The wrapper is responsible for scheduling between multiple devices. This wrapper is responsible to serialize commands to the chip(Refer Datasheet).

3.2.1 OS driver to core interface

The calls implemented for specific operating system has to call the appropriate core driver specific calls. The arguments and definitions are as defined in bsafe_core_if.h

Table 1:

OS(xx) driver function	*function description*	Core driver function call to be called by the OS(xx) driver function
bsafe_(xx)_initialize	Probe and find the device and initialize the function. This is called once on load	bsafe_initialize
bsafe_(xx)_open	Driver open function call	
bsafe_(xx)_ioctl	Driver ioctl function call	bsafe_ioctl
bsafe_(xx)_close	Driver close function call	
bsafe_(xx)_clean	Uninitialize or unload of driver	bsafe_clean
bsafe_(xx)_isr_callback	Interrupt service routine callback which is hooked to an interrupt	bsafe_isr_callback

3.3 Input/Output Hardware Interfaces

The input/output hardware interface is implemented based on the specific hardware interface for the microHSM core.

The hardware interface can be implemented based on the data input-output interface or by using the standard data input/output interface but implementing the uint32 input/output interface.

3.3.1 Data input/output interface

These macro's are used by the core driver for read and write of a datastream/packet through the interface to the core. The interface implementation is responsible for the required handshake required by the core and the input/output interface. The definitions are defined in bsafe_core_if.h.

DATA I/O interface	*function*
WRITE_TO_DEVICE	Write a block of data through the interface
READ_FROM_DEVICE	Read a block of data from the interface

3.3.2 uint32 input/output interface

These core input/output macro's are used by the core driver for read and write of a 32 bit data through the interface to the core. These macros are used by the WRITE_TO_DEVICE/READ_FROM_DEVICE standard implementation in bsafe_readwrite.c. The definitions are define in bsafe_core_if.h

UINT32 I/O interface	*function*
WRITE_UINT32_TO_DEVICE	Write 32 bits of data through the interface
READ_UINT32_FROM_DEVICE	Read 32 bits of data from the interface

Section 4: Appendix

4.1 Driver Return Status (bsafe_status.h)

The return status of the driver is defined in bsafe_status.h

```
#ifndef __BSAFE_STATUS_H_
#define __BSAFE_STATUS_H_

#ifndef BSAFE_STATUS_BEGIN
#define BSAFE_STATUS_BEGIN typedef enum bsafe_status {
#define BSAFE_STATUS_END } bsafe_status_t;
#define BSAFE_ERRORCODE(name,value) name=value,
#endif

BSAFE_STATUS_BEGIN
BSAFE_ERRORCODE( BSAFE_MEM_ALLOC_FAILED,-101)
BSAFE_ERRORCODE( BSAFE_NO_FREE_AKEYLOC,-102)
BSAFE_ERRORCODE( BSAFE_NO_FREE_KEKLOC,-103)
BSAFE_ERRORCODE( BSAFE_INVALID_KEYLOC_PARAM,-104)
BSAFE_ERRORCODE( BSAFE_UNKNOWN_IOCTL,-105)
BSAFE_ERRORCODE( BSAFE_OUTPUT_CMD_SIZE_MISMATCH,-106)
BSAFE_ERRORCODE( BSAFE_TIMEDOUT,-107)
BSAFE_ERRORCODE( BSAFE_WRITE_FAILED,-201)
BSAFE_ERRORCODE( BSAFE_READ_FAILED,-202)
BSAFE_ERRORCODE( BSAFE_SUCCESS,0)
BSAFE_STATUS_END

#endif /* __BSAFE_STATUS_H_ */
```

4.2 Device Return Code (bsafe_returncode.h)

The returncode's of the device is defined in bsafe_returncode.h

```
#ifndef __BSAFE_RETURNCODE_H_
#define __BSAFE_RETURNCODE_H_

#ifndef BSAFE_RETURNCODE_BEGIN
#define BSAFE_RETURNCODE_BEGIN typedef enum bsafe_returncode {
#define BSAFE_RETURNCODE_END } bsafe_returncode_t;
```

```
#define BSAFE_RETURNCODE( name,value) name=value,
#endif


BSAFE_RETURNCODE_BEGIN


///////////////////////////////////////////////////////
// Define Declarations)
///////////////////////////////////////////////////////

BSAFE_RETURNCODE( UHSM_SUCCESS,0x0000)
BSAFE_RETURNCODE( UHSM_FAIL,0x0001)
BSAFE_RETURNCODE( TPM_SUCCESS,0x0000)
BSAFE_RETURNCODE( TPM_FAIL,0x0001)

//
// these are related to command parsing)
//
BSAFE_RETURNCODE( CMD_MAX_LENGTH_ERR,0x8002)
BSAFE_RETURNCODE( CMD_LENGTH_ERR,0x8003)
BSAFE_RETURNCODE( CMD_UNKNOWN_ERR,0x8004)
BSAFE_RETURNCODE( CMD_BAD_PARAM_ERR,0x8005)
BSAFE_RETURNCODE( UNINIT_GLB_TIMER_ERR          ,0x8006)
BSAFE_RETURNCODE( BAD_CMD_ERR          ,0x8007)

//
// these are related to OTP )
//
BSAFE_RETURNCODE( OTP_NOT_CLR_ERR,0x8080)
BSAFE_RETURNCODE( OTP_PREV_PROG_ERR,0x8081)
BSAFE_RETURNCODE( OTP_PROG_FAIL_ERR,0x8082)
BSAFE_RETURNCODE( OTP_NOT_PROG_ERR,0x8083)

//
// these are related to KEK or AKEY )
//
BSAFE_RETURNCODE( KEY_CACHE_ERR,0x0100)

BSAFE_RETURNCODE( KEK_BAD_TYPE_ERR,0x0101)
BSAFE_RETURNCODE( KEK_BAD_SIZE_ERR,0x0102)
BSAFE_RETURNCODE( KEK_BAD_STAT_ERR,0x0103)
BSAFE_RETURNCODE( KEK_BAD_LOC_ERR,0x0104)
BSAFE_RETURNCODE( KEK_BKT_FULL_ERR,0x8105)
BSAFE_RETURNCODE( KEK_POLICY_ERR,0x8106)
BSAFE_RETURNCODE( KEK_TIME_EXP_ERR,0x8107)

BSAFE_RETURNCODE( AKEY_BAD_TYPE_ERR,0x0201)
BSAFE_RETURNCODE( AKEY_BAD_SIZE_ERR,0x0202)
BSAFE_RETURNCODE( AKEY_BAD_STAT_ERR,0x0203)
BSAFE_RETURNCODE( AKEY_BAD_LOC_ERR,0x0204)
BSAFE_RETURNCODE( AKEY_BKT_FULL_ERR,0x8205)
BSAFE_RETURNCODE( AKEY_POLICY_ERR,0x8206)
BSAFE_RETURNCODE( AKEY_TIME_EXP_ERR,0x8207)

//
// these are related to OTP configuration)
//
BSAFE_RETURNCODE( AUTHORIZATION_VLD_ERR,0x0300)
BSAFE_RETURNCODE( DIS_USER_AUTH_ERR,0x0301)
BSAFE_RETURNCODE( ENABLE_USERKEY_ERR,0x0302)
BSAFE_RETURNCODE( DIS_RAWRNG_ERR,0x0303)
BSAFE_RETURNCODE( DIS_USERSTATUS_ERR,0x0304)

//
// these are more specific to individual commands)
//
BSAFE_RETURNCODE( SELFTEST_HW_FAIL_ERR,0x0400 )
BSAFE_RETURNCODE( SELFTEST_DONE_ERR,0x0401 )

BSAFE_RETURNCODE( LDCFG_INVLD_KMP_ERR,0x0420)
BSAFE_RETURNCODE( LDCFG_SIG_ERR,0x0421)
BSAFE_RETURNCODE( LDCFG_MSGID_ERR,0x0422)

BSAFE_RETURNCODE( USERAUTH_TIMEOUT_ERR          ,0x0440)

BSAFE_RETURNCODE( DLIES_NO_KS_ERR,0x0500)
BSAFE_RETURNCODE( DLIES_CACHE_ERR,0x0501)
BSAFE_RETURNCODE( DLIES_NEST_LVL_ERR,0x0502)
BSAFE_RETURNCODE( DLIES_LOC_ERR,0x0503)
BSAFE_RETURNCODE( DLIES_MSGID_ERR,0x0504)
BSAFE_RETURNCODE( DLIES_DECRYPT_ERR,0x0505)
BSAFE_RETURNCODE( DLIES_MAC_ERR,0x0506)

//
// these are more related to hardware modules)
//
BSAFE_RETURNCODE( DES_BLK_SIZE_ERR,0x0800)
BSAFE_RETURNCODE( PKE_UNKNOWN_OP_ERR,0x0820)
BSAFE_RETURNCODE( HSH_MODE_ERR,0x0840)
BSAFE_RETURNCODE( HSH_BLK_SIZE_ERR,0x0841)
BSAFE_RETURNCODE( RNG_TYPE_ERR   ,0x0860)
BSAFE_RETURNCODE( RNG_TEST_FAIL_ERR,0x0861)


BSAFE_RETURNCODE_END

#endif /* __BSAFE_RETURN_H_ */
```

Broadcom Corporation

16251 Laguna Canyon Road
P.O. Box 57013
Irvine, California 92618
Phone: 949-450-8700
Fax: 949-450-8710