

# libsecure Reference

© 2013 Sony Computer Entertainment Inc.  
All Rights Reserved.  
SCE Confidential

# Table of Contents

<b>Datatypes.....</b>	<b>4</b>
SceLibSecurePaddingOption::options .....	5
SceLibSecureAsymmetricKey .....	6
SceLibSecureBlock .....	7
SceLibSecureBlockCipherModeType .....	8
SceLibSecureCipherKey .....	10
SceLibSecureCipherType .....	11
SceLibSecureDigest .....	12
SceLibSecureErrorType .....	13
SceLibSecureFlagsType .....	15
SceLibSecureHashType .....	18
SceLibSecureHmac .....	19
SceLibSecureMessage .....	20
SceLibSecurePaddingOption .....	21
SceLibSecurePaddingOptionOaep .....	22
SceLibSecurePaddingType .....	23
SceLibSecureSymmetricKey .....	25
<b>Functions.....</b>	<b>26</b>
sceLibSecureAddCipher .....	27
sceLibSecureAddHash .....	29
sceLibSecureCryptographyDecrypt .....	30
sceLibSecureCryptographyDeleteContext .....	32
sceLibSecureCryptographyEncrypt .....	34
sceLibSecureCryptographyGenerateKey .....	36
sceLibSecureCryptographyGetBlockSize .....	38
sceLibSecureCryptographyGetContextSize .....	40
sceLibSecureCryptographyGetKeySize .....	42
sceLibSecureCryptographyMessagePadding .....	44
sceLibSecureCryptographyMessagePaddingSize .....	46
sceLibSecureCryptographyMessageUnpadding .....	48
sceLibSecureCryptographyResetContext .....	49
sceLibSecureCryptographySetContext .....	51
sceLibSecureDestroy .....	53
sceLibSecureHashAddMessage .....	54
sceLibSecureHashDeleteContext .....	56
sceLibSecureHashGetBlockSize .....	58
sceLibSecureHashGetContextSize .....	59
sceLibSecureHashGetDigest .....	60
sceLibSecureHashGetDigestSize .....	62
sceLibSecureHashHmac .....	63
sceLibSecureHashMessage .....	65
sceLibSecureHashResetContext .....	67
sceLibSecureHashSetContext .....	69
sceLibSecureInit .....	71

sceLibSecureRandom.....	72
sceLibSecureRemoveCipher .....	73
sceLibSecureRemoveHash .....	74
<b>Callback Functions.....</b>	<b>75</b>
SceLibSecureCryptographyDecryptBlockCallback.....	76
SceLibSecureCryptographyEncryptBlockCallback .....	77
SceLibSecureCryptographyGenerateKeyCallback.....	78
SceLibSecureCryptographyGetContextSizeCallback.....	79
SceLibSecureCryptographyGetKeySizeCallback .....	80
SceLibSecureCryptographySetContextCallback .....	81
SceLibSecureHashAddDataCallback .....	82
SceLibSecureHashGetContextSizeCallback .....	83
SceLibSecureHashGetDigestCallback .....	84
SceLibSecureHashGetDigestSizeCallback .....	85
SceLibSecureHashSetContextCallback .....	86
SceLibSecureMaskGenerationFunctionCallback .....	87

# Datatypes

000004557368

SCE CONFIDENTIAL

---

## SceLibSecurePaddingOption::options

---

A union that defines additional options for a padding scheme.

### Definition

---

```
#include <libsecure.h>
union options {
    SceLibSecurePaddingOptionOaep oaep;
};
```

### Members

---

*oaep*                      The padding options for the OAEP padding scheme.

### Description

---

Allows additional options to be set for a padding scheme. At the moment only the [SCE\\_LIBSECURE\\_PADDING\\_OAEP](#) scheme requires additional options.

---

# SceLibSecureAsymmetricKey

---

A structure that defines a key used for asymmetric ciphers.

## Definition

---

```
#include <libsecure.h>
typedef struct SceLibSecureAsymmetricKey {
    size_t key_size;
    void *private_key;
    void *public_key;
} SceLibSecureAsymmetricKey;
```

## Members

---

<i>key_size</i>	The key size in bytes (*8 for bits).
<i>private_key</i>	A pointer to the start of the private part of the key.
<i>public_key</i>	A pointer to the start of the public part of the key.

## Description

---

This type is required to represent an asymmetric key used during the encryption and the decryption process.

## Notes

---

The libsecure library does not use this structure directly, but does use its generic form [SceLibSecureCipherKey](#).

## See Also

---

[SceLibSecureCipherKey](#)

SCE CONFIDENTIAL

---

# SceLibSecureBlock

---

The structure that defines a block of memory.

## Definition

---

```
#include <libsecure.h>
typedef struct SceLibSecureBlock {
    void *pointer;
    size_t length;
} SceLibSecureBlock;
```

## Members

---

<i>pointer</i>	A pointer to the start of the block.
<i>length</i>	The size of the block in bytes.

## Description

---

This type is required in a large number of functions or in other structures which need to represent a memory block.

SCE CONFIDENTIAL

# SceLibSecureBlockCipherModeType

An enumeration to represent the block cipher modes supported in libsecure.

## Definition

```
#include <libsecure.h>
typedef enum SceLibSecureBlockCipherModeType {
    SCE_LIBSECURE_BLOCKCIPHERMODE_ECB = 0,
    SCE_LIBSECURE_BLOCKCIPHERMODE_CBC,
    SCE_LIBSECURE_BLOCKCIPHERMODE_PCBC,
    SCE_LIBSECURE_BLOCKCIPHERMODE_CFB,
    SCE_LIBSECURE_BLOCKCIPHERMODE_OFB,
    SCE_LIBSECURE_BLOCKCIPHERMODE_CTR,
    SCE_LIBSECURE_BLOCKCIPHERMODE_CTR_ADDITION
} SceLibSecureBlockCipherModeType;
```

## Enumeration Values

Macro	Value	Description
SCE_LIBSECURE_BLOCKCIPHERMODE_ECB	0	Electronic Code Book cipher mode.
SCE_LIBSECURE_BLOCKCIPHERMODE_CBC	N/A	Cipher Block Chaining cipher mode.
SCE_LIBSECURE_BLOCKCIPHERMODE_PCBC	N/A	Propagating Cipher Block Chaining cipher mode.
SCE_LIBSECURE_BLOCKCIPHERMODE_CFB	N/A	Cipher Feedback cipher mode.
SCE_LIBSECURE_BLOCKCIPHERMODE_OFB	N/A	Output Feedback cipher mode.
SCE_LIBSECURE_BLOCKCIPHERMODE_CTR	N/A	Counter cipher mode (the half lower of the IV is truncated and set to the block number).
SCE_LIBSECURE_BLOCKCIPHERMODE_CTR_ADDITION	N/A	Counter cipher mode (the half lower of the IV is added with the block number).

## Description

A cipher operates on blocks of fixed size and because messages may be of any size (which may not be a multiple of a cipher block size), a block cipher mode is used to ensure a different level of confidentiality for messages of arbitrary size. The same block cipher mode has to be used to encrypt and to decrypt the corresponding message.

## Notes

The ECB mode is used to encrypt and to decrypt messages with no relation on the previous encrypted or decrypted block. It may be used to access data in a stream directly. It should only be used with a message that has a length that is less than the key size. It may be used securely with the RSA cipher using a PKCS#1 or OAEP padding, but should be avoided with other symmetric ciphers. The message length must be a multiple of the cipher block size otherwise it may be necessary to apply a padding scheme to the message.

The CBC, PCBC, CFB and OFB modes are used to encrypt and to decrypt messages with relation to the previous encrypted or decrypted block. Thus, the encryption process, using these modes, can only be sequential but the decryption process requires, in the worst case, access to the previous encrypted or decrypted block and can be used for direct access in a stream. These modes require an initial vector to ensure a different level of confidentiality. The CBC and PCBC mode requires a message length that is a

©SCEI



SCE CONFIDENTIAL

---

multiple of the cipher block size. Otherwise it may be necessary to apply a padding scheme to the message.

The CTR mode is used to encrypt and to decrypt messages with no relation to the previous encrypted or decrypted block. However, it is using an initial vector composed of a value set by the application and appended with the block number, which gives the same level of confidentiality for all blocks. This mode can be used for direct access in a stream during the encryption or the decryption process.

**See Also**

---

[sceLibSecureCryptographyGetContextSize](#), [sceLibSecureCryptographySetContext](#)

000004557368

---

# SceLibSecureCipherKey

---

A generic type that defines a key.

## Definition

---

```
#include <libsecure.h>
typedef void *SceLibSecureCipherKey;
```

## Description

---

This type is used to represent either a symmetric or asymmetric key.

## Notes

---

The table below contains a list of the ciphers included in the library, and the type of key to use for each cipher. Providing the wrong key type to a function will result in undefined behavior:

Cipher	Key type
TEA	<a href="#">SceLibSecureSymmetricKey</a>
XTEA	<a href="#">SceLibSecureSymmetricKey</a>
AES	<a href="#">SceLibSecureSymmetricKey</a>
Blowfish	<a href="#">SceLibSecureSymmetricKey</a>
DES	<a href="#">SceLibSecureSymmetricKey</a>
TDEA	<a href="#">SceLibSecureSymmetricKey</a>
RSA	<a href="#">SceLibSecureAsymmetricKey</a>

## See Also

---

[SceLibSecureSymmetricKey](#), [SceLibSecureAsymmetricKey](#)

SCE CONFIDENTIAL

# SceLibSecureCipherType

An enumeration to represent the ciphers supported in libsecure.

## Definition

```
#include <libsecure.h>
typedef enum SceLibSecureCipherType {
    SCE_LIBSECURE_CIPHER_TEA = 0,
    SCE_LIBSECURE_CIPHER_XTEA,
    SCE_LIBSECURE_CIPHER_BLOWFISH,
    SCE_LIBSECURE_CIPHER_AES,
    SCE_LIBSECURE_CIPHER_DES,
    SCE_LIBSECURE_CIPHER_TDEA,
    SCE_LIBSECURE_CIPHER_RSA = 0x1000,
    SCE_LIBSECURE_CIPHER_USER_BASE = 0x2000
} SceLibSecureCipherType;
```

## Enumeration Values

Macro	Value	Description
SCE_LIBSECURE_CIPHER_TEA	0	Tiny Encryption Algorithm cipher.
SCE_LIBSECURE_CIPHER_XTEA	N/A	eXtended TEA cipher.
SCE_LIBSECURE_CIPHER_BLOWFISH	N/A	Blowfish cipher.
SCE_LIBSECURE_CIPHER_AES	N/A	Advanced Encryption Standard cipher.
SCE_LIBSECURE_CIPHER_DES	N/A	Data Encryption Standard cipher.
SCE_LIBSECURE_CIPHER_TDEA	N/A	Triple Data Encryption Algorithm cipher.
SCE_LIBSECURE_CIPHER_RSA	0x1000	RSA cipher.
SCE_LIBSECURE_CIPHER_USER_BASE	0x2000	User base for additional ciphers.

## Description

A cipher is a reversible mathematic operation used to transpose a plaintext message to a ciphertext (called “encryption process” or “encrypting the message”). Ciphers are also used to transpose a ciphertext back to a plaintext message (called “decryption process” or “decrypting the message”). The encryption and the decryption processes require two inputs which are the message and the key. A cipher can be symmetric, which means using the same key to encrypt and to decrypt a message. It can also be asymmetric and use a set of keys (a public key is used for the encryption, and a private key is used for the decryption).

## Notes

The symmetric ciphers supported in libsecure are TEA, XTEA, Blowfish, AES, DES and Triple DES. The asymmetric cipher supported in libsecure is RSA.

## See Also

[sceLibSecureCryptographyGetContextSize](#), [sceLibSecureCryptographySetContext](#), [sceLibSecureCryptographyGetBlockSize](#), [sceLibSecureCryptographyGenerateKey](#), [sceLibSecureCryptographyGetKeySize](#), [sceLibSecureAddCipher](#), [sceLibSecureRemoveCipher](#)

SCE CONFIDENTIAL

---

# SceLibSecureDigest

---

A structure that defines a digest.

## Definition

---

```
#include <libsecure.h>
typedef SceLibSecureBlock SceLibSecureDigest;
```

## Description

---

This type is the same as a block of memory and it is used to store a hash value.

## See Also

---

[SceLibSecureBlock](#)

SCE CONFIDENTIAL

# SceLibSecureErrorType

An enumeration to represent the error codes returned by the libsecure functions.

## Definition

```
#include <libsecure.h>
typedef enum SceLibSecureErrorType {
    SCE_LIBSECURE_OK = 0,
    SCE_LIBSECURE_ERROR_ALREADY_INITIALIZED,
    SCE_LIBSECURE_ERROR_NOT_INITIALIZED,
    SCE_LIBSECURE_ERROR_OUT_OF_MEMORY,
    SCE_LIBSECURE_ERROR_INVALID_KEY,
    SCE_LIBSECURE_ERROR_INVALID_BLOCK,
    SCE_LIBSECURE_ERROR_INVALID_PARAMETER,
    SCE_LIBSECURE_ERROR_INVALID_CONTEXT,
    SCE_LIBSECURE_ERROR_INVALID_DIGEST,
    SCE_LIBSECURE_ERROR_INVALID_FUNCTION,
    SCE_LIBSECURE_ERROR_INVALID_NUMBER_OF_ROUNDS,
    SCE_LIBSECURE_ERROR_INVALID_ID,
    SCE_LIBSECURE_ERROR_INVALID_MESSAGE,
    SCE_LIBSECURE_ERROR_INVALID_IV,
    SCE_LIBSECURE_ERROR_INVALID_PADDING,
    SCE_LIBSECURE_ERROR_ALREADY_EXISTS,
    SCE_LIBSECURE_ERROR_RANDOM_PROVIDER_UNAVAILABLE,
    SCE_LIBSECURE_ERROR_MESSAGE_TOO_SMALL,
    SCE_LIBSECURE_ERROR_MESSAGE_TOO_BIG,
    SCE_LIBSECURE_ERROR_INTERNAL_PROVIDER_ERROR
} SceLibSecureErrorType;
```

## Enumeration Values

Macro	Value	Description
SCE_LIBSECURE_OK	0	The last function call completed successfully.
SCE_LIBSECURE_ERROR_ALREADY_INITIALIZED	N/A	The library has already been initialized (this happens only when calling <a href="#">sceLibSecureInit()</a> ).
SCE_LIBSECURE_ERROR_NOT_INITIALIZED	N/A	The library has not been initialized. It is required to call <a href="#">sceLibSecureInit()</a> before any other functions.
SCE_LIBSECURE_ERROR_OUT_OF_MEMORY	N/A	The function did not complete because there is not enough memory. A bigger memory block is needed when calling <a href="#">sceLibSecureInit()</a> .
SCE_LIBSECURE_ERROR_INVALID_KEY	N/A	The key provided to the function is invalid either because the size is incorrect or the pointer to the key is NULL.
SCE_LIBSECURE_ERROR_INVALID_BLOCK	N/A	The memory block provided to the function is invalid either because the size is incorrect or the pointer is NULL.
SCE_LIBSECURE_ERROR_INVALID_PARAMETER	N/A	A parameter passed to the function is invalid. This may happen when the function is expecting a non NULL pointer.
SCE_LIBSECURE_ERROR_INVALID_CONTEXT	N/A	The context passed to the function is either not a context or not set correctly.
SCE_LIBSECURE_ERROR_INVALID_DIGEST	N/A	The digest passed to the function is either not a digest or not set correctly.
SCE_LIBSECURE_ERROR_INVALID_FUNCTION	N/A	A callback function that needed to be called has not been specified.

©SCEI

Macro	Value	Description
SCE_LIBSECURE_ERROR_INVALID_NUMBER_OF_ROUNDS	N/A	The number of rounds passed is invalid and cannot be used.
SCE_LIBSECURE_ERROR_INVALID_ID	N/A	An unrecognized cipher or hash id has been passed to the function.
SCE_LIBSECURE_ERROR_INVALID_MESSAGE	N/A	The message provided to the function is invalid either because the size is incorrect or the pointer to the message is NULL.
SCE_LIBSECURE_ERROR_INVALID_IV	N/A	The initial vector provided to the function is invalid either because the size is incorrect or the pointer to the initial vector is NULL.
SCE_LIBSECURE_ERROR_INVALID_PADDING	N/A	An unrecognized padding has been passed to the function.
SCE_LIBSECURE_ERROR_ALREADY_EXISTS	N/A	The cipher or hash id passed has already been allocated and cannot be used again to add a new cipher or hash.
SCE_LIBSECURE_ERROR_RANDOM_PROVIDER_UNAVAILABLE	N/A	The random generator number cannot be used. This error occurs when libsecure needs to use a random number and the flag <a href="#">SCE_LIBSECURE_FLAGS_RANDOM_GENERATOR</a> during the library initialization.
SCE_LIBSECURE_ERROR_MESSAGE_TOO_SMALL	N/A	The message passed is too small and cannot be processed.
SCE_LIBSECURE_ERROR_MESSAGE_TOO_BIG	N/A	The message passed is too big and cannot be processed.
SCE_LIBSECURE_ERROR_INTERNAL_PROVIDER_ERROR	N/A	A cipher or hash returned an unexpected error code.

### Description

All the libsecure functions return error code values to indicate whether the function has completed successfully or if an error has occurred. The callback function used for the implementation of additional ciphers or hashes must return one of the error codes that is the most appropriate to the situation.

SCE CONFIDENTIAL

# SceLibSecureFlagsType

An enumeration to represent the flags used during the libsecure initialization.

## Definition

```
#include <libsecure.h>
typedef enum SceLibSecureFlagsType {
    SCE_LIBSECURE_FLAGS_NONE = 0,
    SCE_LIBSECURE_FLAGS_CIPHER_TEA = 1 << 0,
    SCE_LIBSECURE_FLAGS_CIPHER_XTEA = 1 << 1,
    SCE_LIBSECURE_FLAGS_CIPHER_BLOWFISH = 1 << 2,
    SCE_LIBSECURE_FLAGS_CIPHER_AES = 1 << 3,
    SCE_LIBSECURE_FLAGS_CIPHER_RSA = 1 << 4,
    SCE_LIBSECURE_FLAGS_CIPHER_DES = 1 << 5,
    SCE_LIBSECURE_FLAGS_CIPHER_TDEA = 1 << 6,
    SCE_LIBSECURE_FLAGS_ALL_CIPHERS = SCE_LIBSECURE_FLAGS_CIPHER_TEA |
    SCE_LIBSECURE_FLAGS_CIPHER_XTEA | SCE_LIBSECURE_FLAGS_CIPHER_BLOWFISH |
    SCE_LIBSECURE_FLAGS_CIPHER_AES | SCE_LIBSECURE_FLAGS_CIPHER_RSA |
    SCE_LIBSECURE_FLAGS_CIPHER_TDEA | SCE_LIBSECURE_FLAGS_CIPHER_DES,
    SCE_LIBSECURE_FLAGS_HASH_SHA1 = 0x100 << 0,
    SCE_LIBSECURE_FLAGS_HASH_MD5 = 0x100 << 1,
    SCE_LIBSECURE_FLAGS_HASH_SHA256 = 0x100 << 2,
    SCE_LIBSECURE_FLAGS_HASH_SHA384 = 0x100 << 3,
    SCE_LIBSECURE_FLAGS_HASH_SHA512 = 0x100 << 4,
    SCE_LIBSECURE_FLAGS_ALL_HASH = SCE_LIBSECURE_FLAGS_HASH_SHA1 |
    SCE_LIBSECURE_FLAGS_HASH_MD5 | SCE_LIBSECURE_FLAGS_HASH_SHA256 |
    SCE_LIBSECURE_FLAGS_HASH_SHA384 | SCE_LIBSECURE_FLAGS_HASH_SHA512,
    SCE_LIBSECURE_FLAGS_RANDOM_GENERATOR = 0x10000,
    SCE_LIBSECURE_FLAGS_RELAXED = 0x20000,
    SCE_LIBSECURE_FLAGS_EVP_PADDING_COMPATIBILITY = 0x40000,
    SCE_LIBSECURE_FLAGS_ALL = SCE_LIBSECURE_FLAGS_ALL_CIPHERS |
    SCE_LIBSECURE_FLAGS_ALL_HASH | SCE_LIBSECURE_FLAGS_RANDOM_GENERATOR
} SceLibSecureFlagsType;
```

## Enumeration Values

Macro	Value	Description
SCE_LIBSECURE_FLAGS_NONE	0	No default components of libsecure will be used
SCE_LIBSECURE_FLAGS_CIPHER_TEA	1 << 0	The cipher TEA will be used.
SCE_LIBSECURE_FLAGS_CIPHER_XTEA	1 << 1	The cipher XTEA will be used.
SCE_LIBSECURE_FLAGS_CIPHER_BLOWFISH	1 << 2	The cipher Blowfish will be used.
SCE_LIBSECURE_FLAGS_CIPHER_AES	1 << 3	The cipher AES will be used.
SCE_LIBSECURE_FLAGS_CIPHER_RSA	1 << 4	The cipher RSA will be used.
SCE_LIBSECURE_FLAGS_CIPHER_DES	1 << 5	The cipher DES will be used.
SCE_LIBSECURE_FLAGS_CIPHER_TDEA	1 << 6	The cipher TDEA will be used.

SCE CONFIDENTIAL

Macro	Value	Description
SCE_LIBSECURE_FLAGS_ALL_CIPHERS	SCE_LIBSECURE_FLAGS_CIPHER_TEA   SCE_LIBSECURE_FLAGS_CIPHER_XTEA   SCE_LIBSECURE_FLAGS_CIPHER_BLOWFISH   SCE_LIBSECURE_FLAGS_CIPHER_AES   SCE_LIBSECURE_FLAGS_CIPHER_RSA   SCE_LIBSECURE_FLAGS_CIPHER_TDEA   SCE_LIBSECURE_FLAGS_CIPHER_DES	All ciphers included in libsecure will be used.
SCE_LIBSECURE_FLAGS_HASH_SHA1	0x100 << 0	The hash SHA-1 will be used.
SCE_LIBSECURE_FLAGS_HASH_MD5	0x100 << 1	The hash MD5 will be used.
SCE_LIBSECURE_FLAGS_HASH_SHA256	0x100 << 2	The hash SHA-256 will be used.
SCE_LIBSECURE_FLAGS_HASH_SHA384	0x100 << 3	The hash SHA-384 will be used.
SCE_LIBSECURE_FLAGS_HASH_SHA512	0x100 << 4	The hash SHA-512 will be used.
SCE_LIBSECURE_FLAGS_ALL_HASH	SCE_LIBSECURE_FLAGS_HASH_SHA1   SCE_LIBSECURE_FLAGS_HASH_MD5   SCE_LIBSECURE_FLAGS_HASH_SHA256   SCE_LIBSECURE_FLAGS_HASH_SHA384   SCE_LIBSECURE_FLAGS_HASH_SHA512	All hash included in libsecure will be used.
SCE_LIBSECURE_FLAGS_RANDOM_GENERATOR	0x10000	Random messages can be generated.
SCE_LIBSECURE_FLAGS_RELAXED	0x20000	Standard enforcements will be relaxed. This allows any block cipher mode with asymmetric ciphers to be used. Also, any size of message will be accepted when applying the PKCS#1 and the OAEP padding.



SCE CONFIDENTIAL

Macro	Value	Description
<code>SCE_LIBSECURE_FLAGS_EVP_PADDING_COMPATIBILITY</code>	<code>0x40000</code>	The same behavior for message padding as EVP when used in OpenSSL. A block filled according to the padding size scheme is added if the message size is a multiple of the block size.
<code>SCE_LIBSECURE_FLAGS_ALL</code>	<code>SCE_LIBSECURE_FLAGS_ALL_CIPHERS   SCE_LIBSECURE_FLAGS_ALL_HASH   SCE_LIBSECURE_FLAGS_RANDOM_GENERATOR</code>	All components included in libsecure will be used.

### Description

The libsecure initialization requires flags to specify which components will be used.

### Notes

A [SCE\\_LIBSECURE\\_ERROR\\_RANDOM\\_PROVIDER\\_UNAVAILABLE](#) error is returned if either [sceLibSecureCryptographyDeleteContext\(\)](#), [sceLibSecureCryptographyGenerateKey\(\)](#), [sceLibSecureRandom\(\)](#), [sceLibSecureHashDeleteContext\(\)](#) or [sceLibSecureCryptographyMessagePadding\(\)](#) is called, and the [SCE\\_LIBSECURE\\_FLAGS\\_RANDOM\\_GENERATOR](#) flag was not provided during initialization. When the [SCE\\_LIBSECURE\\_FLAGS\\_RELAXED](#) flag is provided, the use of any block cipher mode with asymmetric ciphers is allowed. Also, when applying PKCS#1 and OAEP padding, any size of the message is acceptable.

### See Also

[sceLibSecureInit](#)

SCE CONFIDENTIAL

# SceLibSecureHashType

An enumeration to represent the hashes supported in libsecure.

## Definition

```
#include <libsecure.h>
typedef enum SceLibSecureHashType {
    SCE_LIBSECURE_HASH_SHA1 = 0,
    SCE_LIBSECURE_HASH_MD5,
    SCE_LIBSECURE_HASH_SHA256,
    SCE_LIBSECURE_HASH_SHA384,
    SCE_LIBSECURE_HASH_SHA512,
    SCE_LIBSECURE_HASH_USER_BASE = 0x2000
} SceLibSecureHashType;
```

## Enumeration Values

Macro	Value	Description
SCE_LIBSECURE_HASH_SHA1	0	SHA-1.
SCE_LIBSECURE_HASH_MD5	N/A	MD5.
SCE_LIBSECURE_HASH_SHA256	N/A	SHA-256.
SCE_LIBSECURE_HASH_SHA384	N/A	SHA-384.
SCE_LIBSECURE_HASH_SHA512	N/A	SHA-512.
SCE_LIBSECURE_HASH_USER_BASE	0x2000	User base for an additional hash.

## Description

A hash is a non-reversible mathematical operation to transpose a message to a value considered as the message's digital fingerprint. It is used for message integrity.

## See Also

[sceLibSecureHashGetDigestSize](#), [sceLibSecureHashGetContextSize](#),  
[sceLibSecureHashSetContext](#), [sceLibSecureHashGetBlockSize](#),  
[sceLibSecureAddHash](#), [sceLibSecureRemoveHash](#)

SCE CONFIDENTIAL

---

# SceLibSecureHmac

---

A structure that defines a message authentication (HMAC) value.

## Definition

---

```
#include <libsecure.h>
typedef SceLibSecureBlock SceLibSecureHmac;
```

## Description

---

This type is the same as a block of memory. Its size does match the digest size of the underlying hash used during the message authentication.

## See Also

---

[SceLibSecureBlock](#)

SCE CONFIDENTIAL

---

# SceLibSecureMessage

---

A structure that defines a message.

## Definition

---

```
#include <libsecure.h>
typedef SceLibSecureBlock SceLibSecureMessage;
```

## Description

---

This type is the same as a block of memory.

## See Also

---

[SceLibSecureBlock](#)

SCE CONFIDENTIAL

# SceLibSecurePaddingOption

A structure that defines padding options.

## Definition

```
#include <libsecure.h>
typedef struct SceLibSecurePaddingOption {
    SceLibSecurePaddingType padding;
    union SceLibSecurePaddingOption::options options;
} SceLibSecurePaddingOption;
```

## Members

<i>padding</i>	The padding scheme to apply to the message.
<i>options</i>	The padding options are set according to the padding scheme specified.

## Description

Some padding schemes are more complex than others. Therefore it may be necessary to pass options to the padding scheme to allow these padding schemes to be used in libsecure.

## Notes

At the moment, only the OAEP padding scheme requires options.

## See Also

[SceLibSecurePaddingOptionOaep](#), [sceLibSecureCryptographyGetBlockSize](#),  
[sceLibSecureCryptographyMessagePaddingSize](#),  
[sceLibSecureCryptographyMessagePadding](#),  
[sceLibSecureCryptographyMessageUnpadding](#),  
[sceLibSecureCryptographyEncrypt](#), [sceLibSecureCryptographyDecrypt](#)

SCE CONFIDENTIAL

# SceLibSecurePaddingOptionOaep

A structure that defines padding options for the OAEP padding scheme.

## Definition

```
#include <libsecure.h>
typedef struct SceLibSecurePaddingOptionOaep {
    SceLibSecureHashType h;
    SceLibSecureBlock *label;
    SceLibSecureMaskGenerationFunctionCallback maskgen_fnc;
    void *mgf_options;
} SceLibSecurePaddingOptionOaep;
```

## Members

<i>h</i>	The hash to use for the OAEP padding (required).
<i>label</i>	A block of memory used during the hash of the padding (optional).
<i>maskgen_fnc</i>	The mask generation function (optional).
<i>mgf_options</i>	Some additional options passed to the mask generation function (optional).

## Description

Padding a message with the OAEP scheme requires a hash method and possibly a label and a mask generation function.

## See Also

[SceLibSecurePaddingOption](#), [SceLibSecureMaskGenerationFunctionCallback](#)

# SceLibSecurePaddingType

An enumeration to represent the padding schemes supported in libsecure.

## Definition

```
#include <libsecure.h>
typedef enum SceLibSecurePaddingType {
    SCE_LIBSECURE_PADDING_NONE = 0,
    SCE_LIBSECURE_PADDING_NONE_NORMAL,
    SCE_LIBSECURE_PADDING_NONE_STEALING,
    SCE_LIBSECURE_PADDING_NONE_BLKTERM,
    SCE_LIBSECURE_PADDING_NIL,
    SCE_LIBSECURE_PADDING_SPACE,
    SCE_LIBSECURE_PADDING_RANDOM,
    SCE_LIBSECURE_PADDING_SIZE,
    SCE_LIBSECURE_PADDING_NILSIZE,
    SCE_LIBSECURE_PADDING_BIT,
    SCE_LIBSECURE_PADDING_PKCS1,
    SCE_LIBSECURE_PADDING_OAEP
} SceLibSecurePaddingType;
```

## Enumeration Values

Macro	Value	Description
SCE_LIBSECURE_PADDING_NONE	0	No padding is enforced. The incomplete block will not be encrypted.
SCE_LIBSECURE_PADDING_NONE_NORMAL	N/A	No padding is enforced. The incomplete block will be encrypted only for CFB, OFB and CTR modes.
SCE_LIBSECURE_PADDING_NONE_STEALING	N/A	No padding is enforced. Uses the ciphertext stealing method.
SCE_LIBSECURE_PADDING_NONE_BLKTERM	N/A	No padding is enforced. Uses the residual block termination method. This mode can only be used with the CBC cipher mode.
SCE_LIBSECURE_PADDING_NIL	N/A	Add nil bytes up to the end of the block to encrypt.
SCE_LIBSECURE_PADDING_SPACE	N/A	Add spaces up to the end of the block to encrypt.
SCE_LIBSECURE_PADDING_RANDOM	N/A	Add random values up to the end of the block to encrypt.
SCE_LIBSECURE_PADDING_SIZE	N/A	The padding consists of n values n, n the remaining number of bytes to complete the block and it can be used for PKCS#7 padding with symmetric ciphers.
SCE_LIBSECURE_PADDING_NILSIZE	N/A	The padding consists of nil bytes except the last one being n.
SCE_LIBSECURE_PADDING_BIT	N/A	Add a single bit 1 followed by 0 bits.
SCE_LIBSECURE_PADDING_PKCS1	N/A	RSA PKCS#1 v1.5 padding (encrypt/decrypt data blocks of the key size in bytes minus 11 bytes).
SCE_LIBSECURE_PADDING_OAEP	N/A	RSA OAEP padding (encrypt/decrypt data blocks including hash of the message).

SCE CONFIDENTIAL

## Description

A padding scheme is used when the message length is not a multiple of the cipher block size and depends on the block cipher mode in use. Note that a padding scheme cannot be used with CFB, OFB and CTR modes. It is possible to not apply a padding scheme and to specify the encryption and the decryption behavior when encountering an incomplete block at the end of the message, or when using the CFB, OFB and CTR modes.

## Notes

Please find below the list of padding that can be used for each cipher mode:

Padding/ Cipher Mode	ECB	CBC	PCBC	CFB	OFB	CTR
SCE_LIBSECURE_PADDING_NONE	X	X	X	X	X	X
SCE_LIBSECURE_PADDING_NONE_NORMAL	X	X	X	X	X	X
SCE_LIBSECURE_PADDING_NONE_STEALING	X	X	X	X	X	X
SCE_LIBSECURE_PADDING_NONE_BLKTERM		X	X			
SCE_LIBSECURE_PADDING_NIL	X	X	X			
SCE_LIBSECURE_PADDING_SPACE	X	X	X			
SCE_LIBSECURE_PADDING_RANDOM	X	X	X			
SCE_LIBSECURE_PADDING_SIZE	X	X	X			
SCE_LIBSECURE_PADDING_NILSIZE	X	X	X			
SCE_LIBSECURE_PADDING_BIT	X	X	X			
SCE_LIBSECURE_PADDING_PKCS1	X					
SCE_LIBSECURE_PADDING_OAEP	X					

## See Also

[SceLibSecurePaddingOption](#), [SceLibSecureBlockCipherModeType](#)



SCE CONFIDENTIAL

---

# SceLibSecureSymmetricKey

---

A structure that defines a key used for symmetric ciphers.

## Definition

---

```
#include <libsecure.h>
typedef struct SceLibSecureSymmetricKey {
    size_t key_size;
    void *key;
} SceLibSecureSymmetricKey;
```

## Members

---

<i>key_size</i>	The key size in bytes (*8 for bits).
<i>key</i>	A pointer to the start of the key.

## Description

---

This type is required to represent a symmetric key used during the encryption and the decryption process, or during a message authentication.

## Notes

---

The libsecure API does not use this structure directly, but does use its generic form [SceLibSecureCipherKey](#).

## See Also

---

[SceLibSecureCipherKey](#)

# Functions

000004557368

SCE CONFIDENTIAL

# sceLibSecureAddCipher

Extends the library by adding a cipher to it.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureAddCipher(
    SceLibSecureCipherType *cipher,
    size_t block_size,
    SceLibSecureCryptographyEncryptBlockCallback enc_block_fn,
    SceLibSecureCryptographyDecryptBlockCallback dec_block_fn,
    SceLibSecureCryptographyGetContextSizeCallback get_context_size_fn,
    SceLibSecureCryptographySetContextCallback set_context_fn,
    SceLibSecureCryptographyGenerateKeyCallback gen_key_fn,
    SceLibSecureCryptographyGetKeySizeCallback get_key_size_fn,
    int bignum_arithmetic
);
```

## Arguments

[out] <i>cipher</i>	Receives the identifier of the added cipher.
[in] <i>block_size</i>	The cipher block size in bytes. If a block size of 0 is passed, the block size is based on the key length passed when setting the context.
[in] <i>enc_block_fn</i>	The <a href="#">SceLibSecureCryptographyEncryptBlockCallback</a> function to encrypt a block (required).
[in] <i>dec_block_fn</i>	The <a href="#">SceLibSecureCryptographyDecryptBlockCallback</a> function to decrypt a block (required).
[in] <i>get_context_size_fn</i>	The <a href="#">SceLibSecureCryptographyGetContextSizeCallback</a> function to retrieve the context size (optional).
[in] <i>set_context_fn</i>	The <a href="#">SceLibSecureCryptographySetContextCallback</a> function to set a context (optional).
[in] <i>gen_key_fn</i>	The <a href="#">SceLibSecureCryptographyGenerateKeyCallback</a> function to generate a key (optional). If none is specified, a random key will be generated when the function <a href="#">sceLibSecureCryptographyGenerateKey()</a> is called.
[in] <i>get_key_size_fn</i>	The <a href="#">SceLibSecureCryptographyGetKeySizeCallback</a> function to retrieve the key size (optional). If none is specified, the key size returned by the function <a href="#">sceLibSecureCryptographyGetKeySize()</a> will be the cipher block size.
[in] <i>bignum_arithmetic</i>	Indicates whether the cipher uses big number arithmetic (such as RSA) to pass the correct message format when encrypting or decrypting a message with this cipher. A zero indicates that the cipher does not use big number arithmetic. A non-zero value indicates that it does.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Extends the library by adding a cipher to it. This function requires a set of callback functions and some information about the cipher. This function returns a cipher identifier for further use in the library.

SCE CONFIDENTIAL

---

**Example**

---

```
SceLibSecureErrorType error;  
SceLibSecureCipherType id;  
...  
error = sceLibSecureAddCipher(&id, 8, tea_enc_block, tea_dec_block,  
    NULL, NULL, NULL, NULL, 0);  
if(error != SCE\_LIBSECURE\_OK) { // Error management }
```

**See Also**

---

[sceLibSecureRemoveCipher](#), [SceLibSecureCipherType](#),  
[SceLibSecureCryptographyEncryptBlockCallback](#),  
[SceLibSecureCryptographyDecryptBlockCallback](#),  
[SceLibSecureCryptographyGetContextSizeCallback](#),  
[SceLibSecureCryptographySetContextCallback](#),  
[SceLibSecureCryptographyGenerateKeyCallback](#),  
[SceLibSecureCryptographyGetKeySizeCallback](#)

SCE CONFIDENTIAL

# sceLibSecureAddHash

Extends the library by adding a hash to it.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureAddHash(
    SceLibSecureHashType *hash,
    size_t block_size,
    SceLibSecureHashGetDigestSizeCallback get_digest_size_fn,
    SceLibSecureHashAddDataCallback add_data_fn,
    SceLibSecureHashGetDigestCallback get_digest_fn,
    SceLibSecureHashGetContextSizeCallback get_context_size_fn,
    SceLibSecureHashSetContextCallback set_context_fn
);
```

## Arguments

[out] <i>hash</i>	Receives the identifier of the added hash.
[in] <i>block_size</i>	The hash block size in bytes. It is required to specify a non-zero value.
[in] <i>get_digest_size_fn</i>	The <a href="#">SceLibSecureHashGetDigestSizeCallback</a> function to retrieve the digest size (required).
[in] <i>add_data_fn</i>	The <a href="#">SceLibSecureHashAddDataCallback</a> function to add more data to a hash (required).
[in] <i>get_digest_fn</i>	The <a href="#">SceLibSecureHashGetDigestCallback</a> function to retrieve the digest value (required).
[in] <i>get_context_size_fn</i>	The <a href="#">SceLibSecureHashGetContextSizeCallback</a> function to retrieve the context size (required).
[in] <i>set_context_fn</i>	The <a href="#">SceLibSecureHashSetContextCallback</a> function to set the context (required).

## Return Values

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

Extends the library by adding a hash to it. This function requires a set of callback functions and some information about the hash. This function returns a hash identifier for further use in the library.

## Example

```
SceLibSecureErrorType error;
SceLibSecureHashType id;
...
error = sceLibSecureAddHash(&id, 64, md5_get_digest_size, md5_add_data,
    md5_get_digest_size, md5_get_context_size, md5_set_context);
if(error != SCE_LIBSECURE_OK) { // Error management }
```

## See Also

[sceLibSecureRemoveHash](#), [SceLibSecureHashType](#),  
[SceLibSecureHashGetDigestSizeCallback](#),  
[SceLibSecureHashAddDataCallback](#), [SceLibSecureHashGetDigestCallback](#),  
[SceLibSecureHashGetContextSizeCallback](#), [SceLibSecureHashSetContextCallback](#)

SCE CONFIDENTIAL

# sceLibSecureCryptographyDecrypt

Decrypts a ciphertext message.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographyDecrypt (
    SceLibSecureBlock *context,
    SceLibSecureMessage *data,
    const SceLibSecurePaddingOption *pad
);
```

## Arguments

[in,out] <i>context</i>	The context to use.
[in,out] <i>data</i>	A ciphertext message. This becomes a plaintext message after calling the function.
[in] <i>pad</i>	The padding options.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Decrypts a ciphertext message. After calling this function, the message *data* passed contains the plaintext message.

## Example

```
char array_key[16] = { ... };
SceLibSecureSymmetricKey key = {sizeof(array_key), array_key};
char array_message[16] = { ... };
SceLibSecureMessage message = {array_message, sizeof(array_message)};
SceLibSecurePaddingOption padding;
size_t context_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
...
// Retrieve the context size required
error = sceLibSecureCryptographyGetContextSize(SCE_LIBSECURE_CIPHER_XTEA,
SCE_LIBSECURE_BLOCKCIPHERMODE_ECB, (SceLibSecureCipherKey*)&key, 16,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureCryptographySetContext(&context,
SCE_LIBSECURE_CIPHER_XTEA, SCE_LIBSECURE_BLOCKCIPHERMODE_ECB,
(SceLibSecureCipherKey*)&key, 16, NULL, 0);
    if(error != SCE_LIBSECURE_OK) { // Error management }
    ...
    // Decrypt the message
    padding.padding = SCE_LIBSECURE_PADDING_RANDOM;
```

SCE CONFIDENTIAL

---

```
error = sceLibSecureCryptographyDecrypt(&context, &message, &padding);  
if(error != SCE\_LIBSECURE\_OK) { // Error management }  
}  
else { // Not enough memory }
```

---

**Notes**

The process shown below has to be followed when encrypting or decrypting a message:

- (1) Retrieve or generate a key.
- (2) Retrieve the context size required for the chosen cipher.
- (3) Set the context according to the chosen cipher.
- (4) Retrieve the cipher block size.
- (5) Pad the message for encryption, if necessary.
- (6) Encrypt or decrypt the message (\*).
- (7) Unpad the message after decryption, if necessary.

The message *data* passed to this function should be a multiple of the cipher's block size, otherwise the error [SCE\\_LIBSECURE\\_ERROR\\_MESSAGE\\_TOO\\_SMALL](#) is returned.

---

**See Also**

[sceLibSecureCryptographyEncrypt](#), [sceLibSecureCryptographyMessageUnpadding](#),  
[SceLibSecureMessage](#), [SceLibSecurePaddingOption](#)

SCE CONFIDENTIAL

# sceLibSecureCryptographyDeleteContext

Deletes (destroys) a cipher context.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographyDeleteContext(
    SceLibSecureBlock *context
);
```

## Arguments

[in,out] *context* The context to delete.

## Return Values

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

Deletes (destroys) a cipher context. For security reasons, it is advised to use this function with a context which has been used or won't be used again. It clears the context with random values.

## Example

```
char array_key[16] = { ... };
SceLibSecureSymmetricKey key = {sizeof(array_key), array_key};
size_t context_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
...
// Retrieve the context size required
error = sceLibSecureCryptographyGetContextSize(SCE_LIBSECURE_CIPHER_XTEA,
SCE_LIBSECURE_BLOCKCIPHERMODE_ECB, (SceLibSecureCipherKey*)&key, 16,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureCryptographySetContext(&context,
SCE_LIBSECURE_CIPHER_XTEA, SCE_LIBSECURE_BLOCKCIPHERMODE_ECB,
(SceLibSecureCipherKey*)&key, 16, NULL, 0);
    if(error != SCE_LIBSECURE_OK) { // Error management }
    ...
    // Delete the context
    error = sceLibSecureCryptographyDeleteContext(&context);
    if(error != SCE_LIBSECURE_OK) { // Error management }
}
else { // Not enough memory }
```



SCE CONFIDENTIAL

---

**Notes**

---

If the context has been deleted it cannot be used again, and needs to be initialized again before being used.

**See Also**

---

[sceLibSecureCryptographyGetContextSize](#), [sceLibSecureCryptographySetContext](#),  
[sceLibSecureCryptographyResetContext](#), [SceLibSecureCipherType](#),  
[SceLibSecureBlockCipherModeType](#), [SceLibSecureCipherKey](#)

000004557368

SCE CONFIDENTIAL

# sceLibSecureCryptographyEncrypt

Encrypts a plaintext message.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographyEncrypt (
    SceLibSecureBlock *context,
    SceLibSecureMessage *data,
    const SceLibSecurePaddingOption *pad
);
```

## Arguments

[in,out] <i>context</i>	The context to use.
[in,out] <i>data</i>	A plaintext message. This becomes a ciphertext message after calling the function.
[in] <i>pad</i>	The padding options.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Encrypts a plaintext message. After calling this function, the message *data* passed contains the ciphertext message.

## Example

```
char array_key[16] = { ... };
SceLibSecureSymmetricKey key = {sizeof(array_key), array_key};
char array_message[10] = { ... };
SceLibSecureMessage message = {array_message, sizeof(array_message)};
SceLibSecurePaddingOption padding;
size_t context_size, padding_size;
SceLibSecureBlock context, message_padded;
SceLibSecureErrorType error;
...
// Retrieve the context size required
error = sceLibSecureCryptographyGetContextSize(SCE_LIBSECURE_CIPHER_XTEA,
SCE_LIBSECURE_BLOCKCIPHERMODE_ECB, (SceLibSecureCipherKey*)&key, 16,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureCryptographySetContext(&context,
SCE_LIBSECURE_CIPHER_XTEA, SCE_LIBSECURE_BLOCKCIPHERMODE_ECB,
(SceLibSecureCipherKey*)&key, 16, NULL, 0);
    if(error != SCE_LIBSECURE_OK) { // Error management }
    ...
    // Retrieve the padding size to apply to the message
    padding.padding = SCE_LIBSECURE_PADDING_RANDOM;
```

©SCEI

## SCE CONFIDENTIAL

---

```

    error = sceLibSecureCryptographyMessagePaddingSize(&context, &message,
&padding, &padding_size);
    if(error != SCE\_LIBSECURE\_OK) { // Error management }

    // Allocate a memory block to contain the message and its padding
    message_padded.length = message.length + padding_size;
    message_padded.pointer = malloc(message_padded.length);

    // Copy the message over
    memcpy(message_padded.pointer, message.pointer, message.length);

    // Apply the padding scheme to the message
    error = sceLibSecureCryptographyMessagePadding(&context, &message_padded,
&padding, message.length);
    if(error != SCE\_LIBSECURE\_OK) { // Error management }

    // Encrypt the message
    error = sceLibSecureCryptographyEncrypt(&context, &message_padded,
&padding);
    if(error != SCE\_LIBSECURE\_OK) { // Error management }
}
else { // Not enough memory }

```

### Notes

---

The process shown below has to be followed when encrypting or decrypting a message:

- (1) Retrieve or generate a key.
- (2) Retrieve the context size required for the chosen cipher.
- (3) Set the context according to the chosen cipher.
- (4) Retrieve the cipher block size.
- (5) Pad the message for encryption, if necessary.
- (6) Encrypt or decrypt the message (\*).
- (7) Unpad the message after decryption, if necessary.

The message *data* passed to this function must include a padding scheme and, if the message is not a multiple of the cipher's block size, the error [SCE\\_LIBSECURE\\_ERROR\\_MESSAGE\\_TOO\\_SMALL](#) is returned.

### See Also

---

[sceLibSecureCryptographyDecrypt](#), [sceLibSecureCryptographyMessagePaddingSize](#),  
[sceLibSecureCryptographyMessagePadding](#), [SceLibSecureMessage](#),  
[SceLibSecurePaddingOption](#)

SCE CONFIDENTIAL

# sceLibSecureCryptographyGenerateKey

Generates a key for a cipher.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographyGenerateKey(
    SceLibSecureCipherType cipher,
    SceLibSecureCipherKey *key
);
```

## Arguments

[in] <i>cipher</i>	The cipher identifier.
[out] <i>key</i>	The generated key.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Generates a key for a cipher. A [SceLibSecureSymmetricKey](#) or [SceLibSecureAsymmetricKey](#) key is required, and the memory allocated must match the size retrieved with the [sceLibSecureCryptographyGetKeySize\(\)](#) function.

## Example

```
SceLibSecureSymmetricKey key;
SceLibSecureErrorType error;
...
// Retrieve the memory size required to store the key
error = sceLibSecureCryptographyGetKeySize(SCE_LIBSECURE_CIPHER_XTEA, 16,
&key.length);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory required for the key
key.pointer = malloc(key.length);
if(key.pointer == NULL) { // Not enough memory }

// Generate the key
error = sceLibSecureCryptographyGenerateKey(SCE_LIBSECURE_CIPHER_XTEA,
(SceLibSecureCipherKey*)&key);
if(error != SCE_LIBSECURE_OK) { // Error management }
```

## Notes

The process shown below has to be followed when encrypting or decrypting a message:

- (1) Retrieve or generate a key (\*).
- (2) Retrieve the context size required for the chosen cipher.
- (3) Set the context according to the chosen cipher.
- (4) Retrieve the cipher block size.
- (5) Pad the message for encryption, if necessary.
- (6) Encrypt or decrypt the message.

SCE CONFIDENTIAL

---

(7) Unpad the message after decryption, if necessary.

For symmetric ciphers (TEA, XTEA, Blowfish, AES, DES and Triple DES), the key is randomly generated using the [sceLibSecureRandom\(\)](#) function internally. To use this function, the library must be initialized with either the [SCE\\_LIBSECURE\\_FLAGS\\_ALL](#) or a value ORed with [SCE\\_LIBSECURE\\_FLAGS\\_RANDOM\\_GENERATOR](#).

#### **See Also**

---

[sceLibSecureCryptographyGetKeySize](#), [SceLibSecureCipherType](#),  
[SceLibSecureSymmetricKey](#), [SceLibSecureAsymmetricKey](#), [SceLibSecureCipherKey](#)

000004557368

# sceLibSecureCryptographyGetBlockSize

Retrieves the amount of data needed to complete a cipher block and uses this information to prepare a message before the encryption or decryption process.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographyGetBlockSize(
    SceLibSecureCipherType cipher,
    const SceLibSecureCipherKey *key,
    const SceLibSecurePaddingOption *pad,
    size_t *blk_size
);
```

## Arguments

[in] <i>cipher</i>	The cipher identifier.
[in] <i>key</i>	The key to use (optional).
[in] <i>pad</i>	The padding scheme to use.
[out] <i>blk_size</i>	Receives the block size required.

## Return Values

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

Retrieves the amount of data needed to complete a cipher block and uses this information to prepare a message before the encryption or decryption process. Depending on the padding scheme, the cipher block size may vary.

## Example

```
size_t block_size;
SceLibSecureErrorType error;
...
error = sceLibSecureCryptographyGetBlockSize(SCE_LIBSECURE_CIPHER_XTEA, NULL,
SCE_LIBSECURE_BLOCKCIPHERMODE_ECB, &block_size);
if(error != SCE_LIBSECURE_OK) { // Error management }
```

## Notes

The process shown below has to be followed when encrypting or decrypting a message:

- (1) Retrieve or generate a key.
- (2) Retrieve the context size required for the chosen cipher.
- (3) Set the context according to the chosen cipher.
- (4) Retrieve the cipher block size (\*).
- (5) Pad the message for encryption, if necessary.
- (6) Encrypt or decrypt the message.
- (7) Unpad the message after decryption if necessary.

The key is optional only for fixed block ciphers or for those ciphers for which the block size does not depend on the key size (all symmetric ciphers falls in this category).

SCE CONFIDENTIAL

---

Please find below the list of ciphers included in the library and their block size:

Cipher	Padding	Block Size (bytes)
TEA	All	8
XTEA	All	8
AES	All	16
Blowfish	All	8
DES	All	8
TDEA	All	8
RSA	PKCS#1	key size - 11
RSA	OAEP	key size - 2 * (hash size + 1)

**See Also**

---

[sceLibSecureCryptographyMessagePaddingSize](#),  
[sceLibSecureCryptographyMessagePadding](#), [SceLibSecureCipherType](#),  
[SceLibSecurePaddingOption](#), [SceLibSecureCipherKey](#)

# sceLibSecureCryptographyGetContextSize

Retrieves the context size required for a specified cipher.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographyGetContextSize(
    SceLibSecureCipherType cipher,
    SceLibSecureBlockCipherModeType blk_mode,
    const SceLibSecureCipherKey *key,
    size_t nb_rounds,
    size_t *mem_size
);
```

## Arguments

[in] <i>cipher</i>	The cipher identifier to use.
[in] <i>blk_mode</i>	The block cipher mode to use.
[in] <i>key</i>	The key to use.
[in] <i>nb_rounds</i>	The number of rounds to use (used only for TEA, XTEA and Blowfish ciphers).
[out] <i>mem_size</i>	Receives the output context size required.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Retrieves the context size required for a specified cipher. This function is used to allocate the required memory block to use as the context. Depending on the cipher, you must specify a valid number of rounds otherwise an error [SCE\\_LIBSECURE\\_ERROR\\_INVALID\\_NUMBER\\_OF\\_ROUNDS](#) is returned.

## Example

```
char array_key[16] = { ... };
SceLibSecureSymmetricKey key = {sizeof(array_key), array_key};
size_t context_size;
SceLibSecureErrorType error;
...
error = sceLibSecureCryptographyGetContextSize(SCE_LIBSECURE_CIPHER_XTEA,
SCE_LIBSECURE_BLOCKCIPHERMODE_ECB, (SceLibSecureCipherKey*)&key, 16,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }
```

## Notes

The process shown below has to be followed when encrypting or decrypting a message:

- (1) Retrieve or generate a key.
- (2) Retrieve the context size required for the chosen cipher (\*).
- (3) Set the context according to the chosen cipher.
- (4) Retrieve the cipher block size.
- (5) Pad the message for encryption if necessary.
- (6) Encrypt or decrypt the message.
- (7) Unpad the message after decryption if necessary.



SCE CONFIDENTIAL

---

**See Also**

---

[sceLibSecureCryptographySetContext](#), [SceLibSecureCipherType](#),  
[SceLibSecureBlockCipherModeType](#), [SceLibSecureCipherKey](#)

000004557368

# sceLibSecureCryptographyGetKeySize

Retrieves the memory size required to store a key for a specific cipher.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographyGetKeySize(
    SceLibSecureCipherType cipher,
    size_t key_size,
    size_t *size_in_bytes
);
```

## Arguments

[in] <i>cipher</i>	The cipher identifier.
[in] <i>key_size</i>	The key size in bytes.
[out] <i>size_in_bytes</i>	Receives the memory size in bytes that is required to store the key.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Retrieves the memory size required to store a key for a specific cipher. This function is mainly used as a utility function to find out the size a key would take for its storage.

## Example

```
char array_key[16] = { ... };
SceLibSecureSymmetricKey key = {sizeof(array_key), array_key};
size_t key_mem_size;
SceLibSecureErrorType error;
...
error = sceLibSecureCryptographyGetKeySize(SCE_LIBSECURE_CIPHER_XTEA,
key.length, &key_mem_size);
if(error != SCE_LIBSECURE_OK) { // Error management }
```

## Notes

A symmetric cipher is using a symmetric key and the memory size required to store the key is identical to the key size. An asymmetric cipher is using an asymmetric key and the memory size required to store the key includes the size of the public key and the private key.

Please find below the list of ciphers included in the library and the key size and memory required to store them:

Cipher	Key size (bytes)	Memory size (bytes)
TEA	16	16
XTEA	16	16
AES	16 to 32	16 to 32
Blowfish	4 to 56	4 to 56
DES	8	8
TDEA	8 to 24	8 to 24
RSA	>=1	key size * 2

SCE CONFIDENTIAL

---

**See Also**

---

[sceLibSecureCryptographyGenerateKey](#), [SceLibSecureCipherType](#),  
[SceLibSecureSymmetricKey](#), [SceLibSecureAsymmetricKey](#)

000004557368

# sceLibSecureCryptographyMessagePadding

Applies a padding scheme to a message.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographyMessagePadding(
    SceLibSecureBlock *context,
    SceLibSecureMessage *data,
    const SceLibSecurePaddingOption *pad,
    size_t real_data_size
);
```

## Arguments

[in] <i>context</i>	The context to use.
[in,out] <i>data</i>	The plaintext message.
[in] <i>pad</i>	The padding options.
[in] <i>real_data_size</i>	The number of valid bytes in the message.

## Return Values

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

Applies a padding scheme to a message. This ensures the message is a multiple of the cipher's block size before the encryption process. This step is optional in case you wish to apply your own custom padding scheme.

## Example

```
char array_key[16] = { ... };
SceLibSecureSymmetricKey key = {sizeof(array_key), array_key};
char array_message[10] = { ... };
SceLibSecureMessage message = {array_message, sizeof(array_message)};
SceLibSecurePaddingOption padding;
size_t context_size, padding_size;
SceLibSecureBlock context, message_padded;
SceLibSecureErrorType error;
...
// Retrieve the context size required
error = sceLibSecureCryptographyGetContextSize(SCE_LIBSECURE_CIPHER_XTEA,
SCE_LIBSECURE_BLOCKCIPHERMODE_ECB, (SceLibSecureCipherKey*)&key, 16,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureCryptographySetContext(&context,
SCE_LIBSECURE_CIPHER_XTEA, SCE_LIBSECURE_BLOCKCIPHERMODE_ECB,
(SceLibSecureCipherKey*)&key, 16, NULL, 0);
    if(error != SCE_LIBSECURE_OK) { // Error management }
```

SCE CONFIDENTIAL

```

...
// Retrieve the padding size to apply to the message
padding.padding = SCE_LIBSECURE_PADDING_RANDOM;
error = sceLibSecureCryptographyMessagePaddingSize(&context, &message,
&padding, &padding_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate a memory block to contain the message and its padding
message_padded.length = message.length + padding_size;
message_padded.pointer = malloc(message_padded.length);

// Copy the message over
memcpy(message_padded.pointer, message.pointer, message.length);

// Apply the padding scheme to the message
error = sceLibSecureCryptographyMessagePadding(&context, &message_padded,
&padding, message.length);
if(error != SCE_LIBSECURE_OK) { // Error management }
}
else { // Not enough memory }

```

## Notes

The process shown below has to be followed when encrypting or decrypting a message:

- (1) Retrieve or generate a key.
- (2) Retrieve the context size required for the chosen cipher.
- (3) Set the context according to the chosen cipher.
- (4) Retrieve the cipher block size.
- (5) Pad the message for encryption, if necessary (\*).
- (6) Encrypt or decrypt the message.
- (7) Unpad the message after decryption, if necessary.

Some ciphers may be incompatible with some padding schemes. We recommend that you use any of the following padding scheme with symmetric ciphers ([SCE\\_LIBSECURE\\_PADDING\\_NIL](#), [SCE\\_LIBSECURE\\_PADDING\\_SPACE](#), [SCE\\_LIBSECURE\\_PADDING\\_RANDOM](#), [SCE\\_LIBSECURE\\_PADDING\\_SIZE](#), [SCE\\_LIBSECURE\\_PADDING\\_NILSIZE](#), [SCE\\_LIBSECURE\\_PADDING\\_BIT](#)) and any of the following for the asymmetric ciphers ([SCE\\_LIBSECURE\\_PADDING\\_PKCS1](#), [SCE\\_LIBSECURE\\_PADDING\\_OAEP](#)).

It is possible to not use a padding scheme ([SCE\\_LIBSECURE\\_PADDING\\_NONE](#), [SCE\\_LIBSECURE\\_PADDING\\_NONE\\_NORMAL](#), [SCE\\_LIBSECURE\\_PADDING\\_NONE\\_STEALING](#), [SCE\\_LIBSECURE\\_PADDING\\_NONE\\_BLKTERM](#)). In this case, it is not necessary to call the padding functions ([sceLibSecureCryptographyMessagePaddingSize\(\)](#) or [sceLibSecureCryptographyMessagePadding\(\)](#)) and the plaintext message and the ciphertext message are the same size.

The size of the message *data* passed to this function must be equal to the original message plus the padding size retrieved with the [sceLibSecureCryptographyMessagePaddingSize\(\)](#) function.

## See Also

[sceLibSecureCryptographyMessagePaddingSize](#),  
[sceLibSecureCryptographyMessageUnpadding](#), [SceLibSecureMessage](#),  
[SceLibSecurePaddingOption](#)

# sceLibSecureCryptographyMessagePaddingSize

Retrieves the padding size to apply to a message.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographyMessagePaddingSize(
    SceLibSecureBlock *context,
    const SceLibSecureMessage *data,
    const SceLibSecurePaddingOption *pad,
    size_t *mem_size
);
```

## Arguments

[in] <i>context</i>	The context to use.
[in] <i>data</i>	The plaintext message.
[in] <i>pad</i>	The padding options.
[out] <i>mem_size</i>	Receives the padding size to add to the message.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Retrieves the padding size to apply to a message. This ensures the message is a multiple of the cipher's block size before the encryption process. This step is optional in case you wish to apply your own custom padding scheme.

## Example

```
char array_key[16] = { ... };
SceLibSecureSymmetricKey key = {sizeof(array_key), array_key};
char array_message[10] = { ... };
SceLibSecureMessage message = {array_message, sizeof(array_message)};
SceLibSecurePaddingOption padding;
size_t context_size, padding_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
...
// Retrieve the context size required
error = sceLibSecureCryptographyGetContextSize(SCE_LIBSECURE_CIPHER_XTEA,
SCE_LIBSECURE_BLOCKCIPHERMODE_ECB, (SceLibSecureCipherKey*)&key, 16,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureCryptographySetContext(&context,
SCE_LIBSECURE_CIPHER_XTEA, SCE_LIBSECURE_BLOCKCIPHERMODE_ECB,
(SceLibSecureCipherKey*)&key, 16, NULL, 0);
    if(error != SCE_LIBSECURE_OK) { // Error management }
```

SCE CONFIDENTIAL

---

```

...
// Retrieve the padding size to apply to the message
padding.padding = SCE\_LIBSECURE\_PADDING\_RANDOM;
error = sceLibSecureCryptographyMessagePaddingSize(&context, &message,
&padding, &padding_size);
if(error != SCE\_LIBSECURE\_OK) { // Error management }
}
else { // Not enough memory }

```

**Notes**

---

The process shown below has to be followed when encrypting or decrypting a message:

- (1) Retrieve or generate a key.
- (2) Retrieve the context size required for the chosen cipher.
- (3) Set the context according to the chosen cipher.
- (4) Retrieve the cipher block size.
- (5) Pad the message for encryption if necessary (\*).
- (6) Encrypt or decrypt the message.
- (7) Unpad the message after decryption if necessary.

Some ciphers may be incompatible with some padding schemes. We recommend that you use one of the the following padding schemes with symmetric ciphers ([SCE\\_LIBSECURE\\_PADDING\\_NIL](#), [SCE\\_LIBSECURE\\_PADDING\\_SPACE](#), [SCE\\_LIBSECURE\\_PADDING\\_RANDOM](#), [SCE\\_LIBSECURE\\_PADDING\\_SIZE](#), [SCE\\_LIBSECURE\\_PADDING\\_NILSIZE](#), [SCE\\_LIBSECURE\\_PADDING\\_BIT](#)) and any of the following for the asymmetric ciphers ([SCE\\_LIBSECURE\\_PADDING\\_PKCS1](#), [SCE\\_LIBSECURE\\_PADDING\\_OAEP](#)).

It is possible to not use a padding scheme ([SCE\\_LIBSECURE\\_PADDING\\_NONE](#), [SCE\\_LIBSECURE\\_PADDING\\_NONE\\_NORMAL](#), [SCE\\_LIBSECURE\\_PADDING\\_NONE\\_STEALING](#), [SCE\\_LIBSECURE\\_PADDING\\_NONE\\_BLKTERM](#)). In this case, it is not necessary to call the padding functions ([sceLibSecureCryptographyMessagePaddingSize\(\)](#) or [sceLibSecureCryptographyMessagePadding\(\)](#)) and the plaintext message and the ciphertext message are the same size.

**See Also**

---

[sceLibSecureCryptographyMessagePadding](#),  
[sceLibSecureCryptographyMessageUnpadding](#), [SceLibSecureMessage](#),  
[SceLibSecurePaddingOption](#)

# sceLibSecureCryptographyMessageUnpadding

Drops a padding scheme from a message after the decryption process.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographyMessageUnpadding(
    SceLibSecureBlock *context,
    SceLibSecureMessage *data,
    const SceLibSecurePaddingOption *pad,
    size_t *real_data_size
);
```

## Arguments

[in] <i>context</i>	The context to use.
[in] <i>data</i>	The plaintext message.
[in] <i>pad</i>	The padding options.
[out] <i>real_data_size</i>	Receives the number of valid bytes in the message.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Drops a padding scheme from a message after the decryption process. This step is optional in case you apply your own custom padding scheme.

## Notes

The process shown below has to be followed when encrypting or decrypting a message:

- (1) Retrieve or generate a key.
- (2) Retrieve the context size required for the chosen cipher.
- (3) Set the context according to the chosen cipher.
- (4) Retrieve the cipher block size.
- (5) Pad the message for encryption if necessary.
- (6) Encrypt or decrypt the message.
- (7) Unpad the message after decryption if necessary (\*).

Due to the nature of the padding, only the padding schemes [SCE\\_LIBSECURE\\_PADDING\\_PKCS1](#) and [SCE\\_LIBSECURE\\_PADDING\\_OAEP](#) can be dropped from a message. For the other paddings, you need to retrieve the size of the original message by other means (transmitting via a communication protocol or including the size in the original message).

The first *real\_data\_size* bytes of the message are valid after calling this function.

## See Also

[sceLibSecureCryptographyMessagePaddingSize](#),  
[sceLibSecureCryptographyMessagePadding](#),  
[SceLibSecureMessage](#), [SceLibSecurePaddingOption](#)



# sceLibSecureCryptographyResetContext

Resets a cipher context thereby allowing separate encryption or decryption matching the same cipher.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographyResetContext (
    SceLibSecureBlock *context,
    const SceLibSecureBlock *iv,
    off_t offset
);
```

## Arguments

[in,out] <i>context</i>	The context already set with the <a href="#">sceLibSecureCryptographySetContext()</a> .
[in] <i>iv</i>	The initialization vector to set to the context (used only for CBC, PCBC, CFB, OFB and CTR block cipher modes).
[in] <i>offset</i>	The block number of the starting block in the stream (used only for the CTR block cipher mode).

## Return Values

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

Resets a cipher context thereby allowing separate encryption or decryption matching the same cipher. The context is required to have been previously initialized with the [sceLibSecureCryptographySetContext\(\)](#). Depending on the block cipher mode set to the context, you must specify an initialization vector or a [SCE\\_LIBSECURE\\_ERROR\\_INVALID\\_IV](#) error will be returned.

## Example

```
char array_key[16] = { ... };
SceLibSecureSymmetricKey key = {sizeof(array_key), array_key};
size_t context_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
...
// Retrieve the context size required
error = sceLibSecureCryptographyGetContextSize(SCE_LIBSECURE_CIPHER_XTEA,
SCE_LIBSECURE_BLOCKCIPHERMODE_ECB, (SceLibSecureCipherKey*)&key, 16,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureCryptographySetContext(&context,
SCE_LIBSECURE_CIPHER_XTEA, SCE_LIBSECURE_BLOCKCIPHERMODE_ECB,
(SceLibSecureCipherKey*)&key, 16, NULL, 0);
```

SCE CONFIDENTIAL

---

```
if(error != SCE_LIBSECURE_OK) { // Error management }
...
// Reset the context
error = sceLibSecureCryptographyResetContext(&context, NULL, 0);
if(error != SCE_LIBSECURE_OK) { // Error management }
}
else { // Not enough memory }
```

**Notes**

---

The process shown below has to be followed when encrypting or decrypting a message:

- (1) Retrieve or generate a key.
- (2) Retrieve the context size required for the chosen cipher.
- (3) Set the context accordingly to the chosen cipher (\*).
- (4) Retrieve the cipher block size.
- (5) Pad the message for encryption if necessary.
- (6) Encrypt or decrypt the message.
- (7) Unpad the message after decryption if necessary.

It is advised for security reasons to destroy the context by calling the [sceLibSecureCryptographyDeleteContext\(\)](#) function if the context is not going to be used again.

When using the CTR cipher mode, you must set the offset in the stream from which the message starts. The CTR cipher mode is automatically incremented after an encryption or a decryption operation. Two cipher contexts need to be created in that situation to allow sequential encryption and decryption in the same application.

**See Also**

---

[sceLibSecureCryptographyGetContextSize](#), [sceLibSecureCryptographySetContext](#), [sceLibSecureCryptographyDeleteContext](#), [SceLibSecureCipherType](#), [SceLibSecureBlockCipherModeType](#), [SceLibSecureCipherKey](#)

# sceLibSecureCryptographySetContext

Sets a context which matches the size returned by the [sceLibSecureCryptographyGetContextSize\(\)](#) function.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureCryptographySetContext (
    SceLibSecureBlock *context,
    SceLibSecureCipherType cipher,
    SceLibSecureBlockCipherModeType blk_mode,
    const SceLibSecureCipherKey *key,
    size_t nb_rounds,
    const SceLibSecureBlock *iv,
    off_t offset
);
```

## Arguments

[in,out] <i>context</i>	The context matching the size returned by <a href="#">sceLibSecureCryptographyGetContextSize()</a> .
[in] <i>cipher</i>	The cipher identifier to set to the context.
[in] <i>blk_mode</i>	The block cipher mode to set to the context.
[in] <i>key</i>	The key to set to the context.
[in] <i>nb_rounds</i>	The number of rounds to set to the context (used only for TEA, XTEA and Blowfish ciphers).
[in] <i>iv</i>	The initialization vector to set to the context (used only for CBC, PCBC, CFB, OFB and CTR block cipher modes).
[in] <i>offset</i>	The block number of the starting block in the stream (used only for the CTR block cipher mode).

## Return Values

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

Sets a context which matches the size returned by the [sceLibSecureCryptographyGetContextSize\(\)](#) function. Depending on the cipher, you must specify a valid number of rounds otherwise an error [SCE\\_LIBSECURE\\_ERROR\\_INVALID\\_NUMBER\\_OF\\_ROUNDS](#) is returned. Depending on the block cipher mode, you must specify an initialization vector or a [SCE\\_LIBSECURE\\_ERROR\\_INVALID\\_IV](#) error will be returned.

## Example

```
char array_key[16] = { ... };
SceLibSecureSymmetricKey key = {sizeof(array_key), array_key};
size_t context_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
...
// Retrieve the context size required
```

## SCE CONFIDENTIAL

---

```

    error = sceLibSecureCryptographyGetContextSize(SCE\_LIBSECURE\_CIPHER\_XTEA,
SCE\_LIBSECURE\_BLOCKCIPHERMODE\_ECB, (SceLibSecureCipherKey\*)&key, 16,
&context_size);
    if(error != SCE\_LIBSECURE\_OK) { // Error management }

    // Allocate the memory for the context
    context.length = context_size;
    context.pointer = malloc(context_size);
    if(context.pointer != NULL)
    {
        // Set the context
        error = sceLibSecureCryptographySetContext(&context,
SCE\_LIBSECURE\_CIPHER\_XTEA, SCE\_LIBSECURE\_BLOCKCIPHERMODE\_ECB,
(SceLibSecureCipherKey\*)&key, 16, NULL, 0);
        if(error != SCE\_LIBSECURE\_OK) { // Error management }
        ...
    }
    else { // Not enough memory }

```

**Notes**

The process shown below has to be followed when encrypting or decrypting a message:

- (1) Retrieve or generate a key.
- (2) Retrieve the context size required for the chosen cipher.
- (3) Set the context according to the chosen cipher (\*).
- (4) Retrieve the cipher block size.
- (5) Pad the message for encryption if necessary.
- (6) Encrypt or decrypt the message.
- (7) Unpad the message after decryption if necessary.

A context can easily be reused later with the same setting using the [sceLibSecureCryptographyResetContext\(\)](#) function.

It is advised for security reasons to destroy the context, if it is not going to be used again, by calling the [sceLibSecureCryptographyDeleteContext\(\)](#) function.

For TEA and XTEA ciphers, *nb\_rounds* can be set to any values. 0 is not recommended at all though, as the ciphertext will be the same as the plaintext. Values below 16 are not recommended at all either. Only values above 20 should be used.

For the Blowfish cipher, *nb\_rounds* must be set between 0 and 16 and we recommended using 16 rounds for this cipher.

For the AES, DES, Triple DES and RSA ciphers, *nb\_rounds* is not used.

When using the CTR cipher mode, you must set the offset in the stream from which the message starts. The CTR cipher mode is automatically incremented after an encryption or a decryption operation. A two-cipher context needs to be created in that situation to allow sequential encryption and decryption in the same application.

**See Also**

[sceLibSecureCryptographyGetContextSize](#), [sceLibSecureCryptographyResetContext](#),  
[sceLibSecureCryptographyDeleteContext](#), [SceLibSecureCipherType](#),  
[SceLibSecureBlockCipherModeType](#), [SceLibSecureCipherKey](#)

SCE CONFIDENTIAL

---

# sceLibSecureDestroy

---

Uninitializes the library.

## Definition

---

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureDestroy(void);
```

## Arguments

---

None

## Return Values

---

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

---

Uninitializes the library. The memory block passed during the initialization may be reclaimed and used after calling this function.

## Example

---

```
SceLibSecureErrorType error;
...
error = sceLibSecureDestroy();
if(error != SCE\_LIBSECURE\_OK) { // Error management }
```

## See Also

---

[sceLibSecureInit](#)

SCE CONFIDENTIAL

# sceLibSecureHashAddMessage

Adds a message to the hash.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureHashAddMessage(
    SceLibSecureBlock *context,
    const SceLibSecureMessage *data
);
```

## Arguments

[in,out] *context*    The context to use.  
[in] *data*            The message to incorporate in the digest.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Adds a message to the hash. This function is used to incorporate a message in a hash calculation.

## Example

```
char array_message[256] = { ... };
SceLibSecureMessage message = {array_message, sizeof(array_message)};
size_t context_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
...
// Retrieve the context size required
error = sceLibSecureHashGetContextSize(SCE_LIBSECURE_HASH_SHA1,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureHashSetContext(&context, SCE_LIBSECURE_HASH_SHA1);
    if(error != SCE_LIBSECURE_OK) { // Error management }

    // Add a message to the hash
    error = sceLibSecureHashAddMessage(&context, &message);
    if(error != SCE_LIBSECURE_OK) { // Error management }
}
else { // Not enough memory }
```

## Notes

The process shown below has to be followed when applying a hash method to a message:

- (1) Retrieve the context size required for the chosen hash.

SCE CONFIDENTIAL

---

- (2) Set the context according to the chosen hash.
- (3) Add messages to the hash (\*).
- (4) Retrieve the digest size.
- (5) Retrieve the digest of the message.

It is possible to add further messages by calling this function with the other messages.

#### **See Also**

---

[sceLibSecureHashSetContext](#), [SceLibSecureMessage](#),  
[sceLibSecureHashGetDigestSize](#), [sceLibSecureHashGetDigest](#)

000004557368

SCE CONFIDENTIAL

# sceLibSecureHashDeleteContext

Deletes (destroys) a hash context.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureHashDeleteContext (
    SceLibSecureBlock *context
);
```

## Arguments

[in,out] *context* The context to delete.

## Return Values

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

Deletes (destroys) a hash context. For security reasons, it is advised to use this function with a context which has been used or will not be used again. It clears the context of random values.

## Example

```
size_t context_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
...
// Retrieve the context size required
error = sceLibSecureHashGetContextSize(SCE_LIBSECURE_HASH_SHA1,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureHashSetContext(&context, SCE_LIBSECURE_HASH_SHA1);
    if(error != SCE_LIBSECURE_OK) { // Error management }
    ...
    // Delete the context
    error = sceLibSecureHashDeleteContext(&context);
    if(error != SCE_LIBSECURE_OK) { // Error management }
}
else { // Not enough memory }
```

## Notes

If the context has been deleted, it cannot be used again. It needs to be initialized again before it is used.



SCE CONFIDENTIAL

---

**See Also**

---

[sceLibSecureHashGetContextSize](#), [sceLibSecureHashSetContext](#),  
[sceLibSecureHashResetContext](#), [SceLibSecureHashType](#)

000004557368

# sceLibSecureHashGetBlockSize

Retrieves the block size of a hash.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureHashGetBlockSize(
    SceLibSecureHashType hash,
    size_t *blk_size
);
```

## Arguments

[in] *hash*            The hash identifier.  
[out] *blk\_size*        Receives the block size in bytes.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Retrieves the block size of a hash.

## Example

```
size_t block_size;
SceLibSecureErrorType error;
...
// Retrieve the block size
error = sceLibSecureHashGetBlockSize(SCE_LIBSECURE_HASH_SHA1, &block_size);
if(error != SCE_LIBSECURE_OK) { // Error management }
```

## Notes

Please find below the list of hashes included in the library and the block size:

Hash	Block Size (bytes)
MD5	64
SHA-1	64
SHA-256	64
SHA-384	128
SHA-512	128

## See Also

[SceLibSecureHashType](#)

# sceLibSecureHashGetContextSize

Retrieves the context size required for a specified hash.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureHashGetContextSize(
    SceLibSecureHashType hash,
    size_t *mem_size
);
```

## Arguments

[in] <i>hash</i>	The hash identifier to use.
[out] <i>mem_size</i>	Receives the required context size.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Retrieves the context size required for a specified hash. This function is used to allocate the required memory block to use as the context.

## Example

```
size_t context_size;
SceLibSecureErrorType error;
...
error = sceLibSecureHashGetContextSize(SCE_LIBSECURE_HASH_SHA1,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }
```

## Notes

The process shown below has to be followed when applying a hash method to a message:

- (1) Retrieve the context size required for the chosen hash (\*).
- (2) Set the context according to the chosen hash.
- (3) Add messages to the hash.
- (4) Retrieve the digest size.
- (5) Retrieve the digest of the message.

## See Also

[sceLibSecureHashSetContext](#), [SceLibSecureHashType](#)

SCE CONFIDENTIAL

# sceLibSecureHashGetDigest

Retrieves the digest value of a hash.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureHashGetDigest(
    SceLibSecureBlock *context,
    SceLibSecureDigest *digest
);
```

## Arguments

[in,out] *context*    The context to use.  
[out] *digest*        Receives the digest value.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Retrieves the digest value of a hash. After calling this function, you cannot add more messages and the context needs to be reset or deleted.

## Example

```
char array_message[256] = { ... };
SceLibSecureMessage message = {array_message, sizeof(array_message)};
size_t context_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
SceLibSecureDigest digest;
...
// Retrieve the context size required
error = sceLibSecureHashGetContextSize(SCE_LIBSECURE_HASH_SHA1,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureHashSetContext(&context, SCE_LIBSECURE_HASH_SHA1);
    if(error != SCE_LIBSECURE_OK) { // Error management }

    // Retrieve the digest size
    error = sceLibSecureHashGetDigestSize(SCE_LIBSECURE_HASH_SHA1,
&digest.length);
    if(error != SCE_LIBSECURE_OK) { // Error management }

    // Allocate the digest value
    digest.pointer = malloc(digest.length);
    if(digest.pointer != NULL)
    {
```

SCE CONFIDENTIAL

---

```
// Add a message to the hash
error = sceLibSecureHashAddMessage(&context, &message);
if(error != SCE\_LIBSECURE\_OK) { // Error management }

// Retrieve the digest value
error = sceLibSecureHashGetDigest(&context, &digest);
if(error != SCE\_LIBSECURE\_OK) { // Error management }
}
else { // Not enough memory }
}
else { // Not enough memory }
```

**Notes**

---

The process shown below has to be followed when applying a hash method to a message:

- (1) Retrieve the context size required for the chosen hash.
- (2) Set the context according to the chosen hash.
- (3) Add messages to the hash.
- (4) Retrieve the digest size.
- (5) Retrieve the digest of the message (\*).

**See Also**

---

[sceLibSecureHashGetDigestSize](#), [SceLibSecureHashType](#), [SceLibSecureDigest](#)

# sceLibSecureHashGetDigestSize

Retrieves the digest size of a hash.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureHashGetDigestSize(
    SceLibSecureHashType hash,
    size_t *mem_size
);
```

## Arguments

[in] *hash*            The hash identifier.  
[out] *mem\_size*        Receives the digest size of the hash in bytes.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Retrieves the digest size of a hash.

## Example

```
size_t digest_size;
SceLibSecureErrorType error;
...
// Retrieve the digest size
error = sceLibSecureHashGetDigestSize( SCE_LIBSECURE_HASH_SHA1,
&digest_size);
if(error != SCE_LIBSECURE_OK) { // Error management }
```

## Notes

The process shown below has to be followed when applying a hash method to a message:

- (1) Retrieve the context size required for the chosen hash.
- (2) Set the context according to the chosen hash.
- (3) Add messages to the hash.
- (4) Retrieve the digest size (\*).
- (5) Retrieve the digest of the message.

Please find below the list of hashes included in the library and the digest size:

Hash	Digest size (bytes)
MD5	16
SHA-1	20
SHA-256	32
SHA-384	48
SHA-512	64

## See Also

[sceLibSecureHashGetDigest](#), [SceLibSecureHashType](#)

SCE CONFIDENTIAL

# sceLibSecureHashHmac

Calculates the Hash Message Authentication Code (HMAC) of a message.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureHashHmac(
    SceLibSecureBlock *context,
    SceLibSecureHmac *hmac,
    const SceLibSecureCipherKey *key,
    const SceLibSecureMessage *data
);
```

## Arguments

[in,out] <i>context</i>	The context already set with the <a href="#">sceLibSecureHashSetContext()</a> .
[out] <i>hmac</i>	Receives the HMAC value. Its size matches the digest size of the hash.
[in] <i>key</i>	A key used for the HMAC calculation. This is known by the sender and receiver.
[in] <i>data</i>	The message used during the calculation.

## Return Values

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

Calculates the Hash Message Authentication Code (HMAC) of a message. This function may be used to ensure a message has not been tampered with after it has been sent and has asked the receiver of the message to process the message with the same HMAC parameters used by the sender. If the values do not match, it indicates the message has been tampered with and its contents should be treated with caution.

## Example

```
char array_key[16] = { ... };
SceLibSecureSymmetricKey key = {sizeof(array_key), array_key};
char array_message[256] = { ... };
SceLibSecureMessage message = {array_message, sizeof(array_message)};
size_t context_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
SceLibSecureHmac hmac;
...
// Retrieve the context size required
error = sceLibSecureHashGetContextSize(SCE_LIBSECURE_HASH_SHA1,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureHashSetContext(&context, SCE_LIBSECURE_HASH_SHA1);
    if(error != SCE_LIBSECURE_OK) { // Error management }
```

SCE CONFIDENTIAL

---

```
// Retrieve the digest size (same as the HASH)
error = sceLibSecureHashGetDigestSize(SCE\_LIBSECURE\_HASH\_SHA1,
&hmac.length);
if(error != SCE\_LIBSECURE\_OK) { // Error management }

// Allocate the HMAC value
hmac.pointer = malloc(hmac.length);
if(hmac.pointer != NULL)
{
    // Calculate the HMAC value
    error = sceLibSecureHashHmac(&context, &hmac, &key, &message);
    if(error != SCE\_LIBSECURE\_OK) { // Error management }
}
else { // Not enough memory }
else { // Not enough memory }
```

**See Also**

---

[sceLibSecureHashGetDigestSize](#), [SceLibSecureHashType](#), [SceLibSecureHmac](#),  
[SceLibSecureSymmetricKey](#), [SceLibSecureAsymmetricKey](#), [SceLibSecureCipherKey](#),  
[SceLibSecureMessage](#)



# sceLibSecureHashMessage

Calculates the digest value of a message directly.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureHashMessage(
    SceLibSecureBlock *context,
    SceLibSecureDigest *digest,
    const SceLibSecureMessage *data
);
```

## Arguments

[in,out] <i>context</i>	The context already set with the <a href="#">sceLibSecureHashSetContext()</a> .
[out] <i>digest</i>	Receives the digest value.
[in] <i>data</i>	The message to incorporate in the digest.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Calculates the digest value of a message directly. This function is used mainly for small messages which can fit in memory. After calling this function, you cannot add more messages and the context needs to be reset or deleted.

## Example

```
char array_message[256] = { ... };
SceLibSecureMessage message = {array_message, sizeof(array_message)};
size_t context_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
SceLibSecureDigest digest;
...
// Retrieve the context size required
error = sceLibSecureHashGetContextSize(SCE_LIBSECURE_HASH_SHA1,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureHashSetContext(&context, SCE_LIBSECURE_HASH_SHA1);
    if(error != SCE_LIBSECURE_OK) { // Error management }

    // Retrieve the digest size
    error = sceLibSecureHashGetDigestSize(SCE_LIBSECURE_HASH_SHA1,
&digest.length);
    if(error != SCE_LIBSECURE_OK) { // Error management }

    // Allocate the digest value
```

SCE CONFIDENTIAL

---

```
digest.pointer = malloc(digest.length);
if(digest.pointer != NULL)
{
    // Add a message to the hash
    error = sceLibSecureHashMessage(&context, &digest, &message);
    if(error != SCE_LIBSECURE_OK) { // Error management }
}
else { // Not enough memory }
}
else { // Not enough memory }
```

**Notes**

---

The process shown below has to be followed when applying a hash method to a message:

- (1) Retrieve the context size required for the chosen hash.
- (2) Set the context according to the chosen hash.
- (3) Add messages to the hash (\*).
- (4) Retrieve the digest size.
- (5) Retrieve the digest of the message (\*).

**See Also**

---

[sceLibSecureHashGetDigestSize](#), [SceLibSecureHashType](#), [SceLibSecureDigest](#),  
[SceLibSecureMessage](#)

SCE CONFIDENTIAL

# sceLibSecureHashResetContext

Resets a hash context thereby allowing a separate hash calculation matching the same hash method to be made.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureHashResetContext (
    SceLibSecureBlock *context
);
```

## Arguments

[in,out] *context* A context that already been set with the [sceLibSecureHashSetContext\(\)](#).

## Return Values

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

Resets a hash context thereby allowing a separate hash calculation matching the same hash method to be made. The context must first be initialized with the [sceLibSecureHashSetContext\(\)](#).

## Example

```
size_t context_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
...
// Retrieve the context size required
error = sceLibSecureHashGetContextSize(SCE_LIBSECURE_HASH_SHA1,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureHashSetContext(&context, SCE_LIBSECURE_HASH_SHA1);
    if(error != SCE_LIBSECURE_OK) { // Error management }
    ...
    // Reset the context
    error = sceLibSecureHashResetContext(&context);
    if(error != SCE_LIBSECURE_OK) { // Error management }
}
else { // Not enough memory }
```

## Notes

The process shown below has to be followed when applying a hash method to a message:

- (1) Retrieve the context size required for the chosen hash.
- (2) Set the context according to the chosen hash (\*).

SCE CONFIDENTIAL

---

- (3) Add messages to the hash.
- (4) Retrieve the digest size.
- (5) Retrieve the digest of the message.

It is advised for security reasons to destroy the context by calling the [sceLibSecureHashDeleteContext\(\)](#) function if it is not going to be used again.

#### **See Also**

---

[sceLibSecureHashGetContextSize](#), [sceLibSecureHashSetContext](#),  
[sceLibSecureHashDeleteContext](#), [SceLibSecureHashType](#)

000004557368

SCE CONFIDENTIAL

# sceLibSecureHashSetContext

Sets a context for a specific hash.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureHashSetContext (
    SceLibSecureBlock *context,
    SceLibSecureHashType hash
);
```

## Arguments

[in,out] <i>context</i>	A context matching the size returned by <a href="#">sceLibSecureHashGetContextSize()</a> .
[in] <i>hash</i>	The hash identifier to set to the context.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Sets a context for a specific hash. The context should match the size returned by the [sceLibSecureHashGetContextSize\(\)](#) function.

## Example

```
size_t context_size;
SceLibSecureErrorType error;
SceLibSecureBlock context;
...
// Retrieve the context size required
error = sceLibSecureHashGetContextSize(SCE_LIBSECURE_HASH_SHA1,
&context_size);
if(error != SCE_LIBSECURE_OK) { // Error management }

// Allocate the memory for the context
context.length = context_size;
context.pointer = malloc(context_size);
if(context.pointer != NULL)
{
    // Set the context
    error = sceLibSecureHashSetContext(&context, SCE_LIBSECURE_HASH_SHA1);
    if(error != SCE_LIBSECURE_OK) { // Error management }
    ...
}
else { // Not enough memory }
```

## Notes

The process shown below has to be followed when applying a hash method to a message:

- (1) Retrieve the context size required for the chosen hash.
- (2) Set the context according to the chosen hash (\*).
- (3) Add messages to the hash.

SCE CONFIDENTIAL

---

- (4) Retrieve the digest size.
- (5) Retrieve the digest of the message.

A context can easily be re-used later, using the same setting and the [sceLibSecureHashResetContext\(\)](#) function.

If the context is not going to be used again, it is advised, for security reasons, to destroy the context by calling the [sceLibSecureHashDeleteContext\(\)](#) function.

#### **See Also**

---

[sceLibSecureHashGetContextSize](#), [sceLibSecureHashResetContext](#),  
[sceLibSecureHashDeleteContext](#), [SceLibSecureHashType](#)

000004557368

SCE CONFIDENTIAL

# sceLibSecureInit

Initializes the libsecure library.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureInit(
    SceLibSecureFlagsType flags,
    SceLibSecureBlock *memory
);
```

## Arguments

[in] <i>flags</i>	The flags indicating the components used.
[in] <i>memory</i>	The memory block used by the library.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

Initializes the libsecure library. This function requires some flags to indicate which components will be used later, and it also requires a memory block.

## Example

```
SceLibSecureErrorType error;
char buffer[5 * 1024];
SceLibSecureBlock mem_block = {buffer, sizeof(buffer)};
...
error = sceLibSecureInit(SCE_LIBSECURE_FLAGS_ALL, &mem_block);
if(error != SCE_LIBSECURE_OK) { // Error management }
```

## Notes

A 1024 bytes memory block is sufficient to hold 14 ciphers and 18 hashes in total.

If the flag [SCE\\_LIBSECURE\\_FLAGS\\_RANDOM\\_GENERATOR](#) is specified, the memory block requires 4160 bytes to be added for a 64 bit platform (PlayStation®3) or 2064 bytes for a 32 bit platform (PlayStation®Portable and PlayStation®Vita). Also, the memory block is used to seed the random number generator and some values retrieved from a random source on the system may be copied into this memory block.

## See Also

[sceLibSecureDestroy](#), [SceLibSecureFlagsType](#), [SceLibSecureBlock](#)

SCE CONFIDENTIAL

# sceLibSecureRandom

Fills a block of memory with random values.

## Definition

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureRandom(
    SceLibSecureBlock *memory
);
```

## Arguments

[out] *memory*      The memory block to fill.

## Return Values

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

Fills a block of memory with random values.

## Example

```
SceLibSecureErrorType error;
char buffer[1024];
SceLibSecureBlock mem_block = {buffer, sizeof(buffer)};
...
error = sceLibSecureRandom(&mem_block);
if(error != SCE\_LIBSECURE\_OK) { // Error management }
```

## Notes

The flag [SCE\\_LIBSECURE\\_FLAGS\\_RANDOM\\_GENERATOR](#) is required to be passed during the initialization. If this is not done, the error code [SCE\\_LIBSECURE\\_ERROR\\_RANDOM\\_PROVIDER\\_UNAVAILABLE](#) is returned.

The algorithm used to generate the random number is ISAAC and is considered to be suitable for cryptographic use.

## See Also

[sceLibSecureInit](#), [SceLibSecureBlock](#)



SCE CONFIDENTIAL

---

## sceLibSecureRemoveCipher

---

Removes a cipher from the library.

### Definition

---

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureRemoveCipher(
    SceLibSecureCipherType cipher
);
```

### Arguments

---

[in] *cipher*      The identifier of the cipher to remove.

### Return Values

---

[SceLibSecureErrorType](#) – Returns a libsecure error code.

### Description

---

Removes a cipher from the library. This may not be used later.

### Example

---

```
SceLibSecureErrorType error;
SceLibSecureCipherType id;
...
error = sceLibSecureRemoveCipher(id);
if(error != SCE\_LIBSECURE\_OK) { // Error management }
```

### See Also

---

[sceLibSecureAddCipher](#), [SceLibSecureCipherType](#)

SCE CONFIDENTIAL

---

## sceLibSecureRemoveHash

---

Removes a hash from the library.

### Definition

---

```
#include <libsecure.h>
SceLibSecureErrorType sceLibSecureRemoveHash(
    SceLibSecureHashType hash
);
```

### Arguments

---

[in] *hash*            The identifier of the hash to remove.

### Return Values

---

[SceLibSecureErrorType](#) - Returns a libsecure error code.

### Description

---

Removes a hash from the library. This may not be used later.

### Example

---

```
SceLibSecureErrorType error;
SceLibSecureHashType id;
...
error = sceLibSecureRemoveHash(id);
if(error != SCE\_LIBSECURE\_OK) { // Error management }
```

### See Also

---

[sceLibSecureAddHash](#), [SceLibSecureHashType](#)

# Callback Functions

000004557368

# SceLibSecureCryptographyDecryptBlockCallback

A callback type to decrypt a block for a specific cipher.

## Definition

```
#include <libsecure.h>
typedef SceLibSecureErrorType
(*SceLibSecureCryptographyDecryptBlockCallback)(
    SceLibSecureBlock *context,
    SceLibSecureMessage *data,
    const SceLibSecureCipherKey *k,
    size_t nb_rounds
);
```

## Arguments

<i>context</i>	The context to use to decrypt the block.
<i>data</i>	The block of data to decrypt.
<i>k</i>	The key to use for the decryption (same as the one set in the context).
<i>nb_rounds</i>	The number of rounds (depending on the cipher, this may not be used).

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

This callback function is called by the function [sceLibSecureCryptographyDecrypt\(\)](#) when the appropriate cipher is used. This is used to add new ciphers in libsecure.

## Notes

When additional ciphers are implemented, a function matching this prototype is required.

## See Also

[SceLibSecureCryptographyEncryptBlockCallback](#),  
[SceLibSecureCryptographyGetContextSizeCallback](#),  
[SceLibSecureCryptographySetContextCallback](#), [sceLibSecureCryptographyDecrypt](#)

# SceLibSecureCryptographyEncryptBlockCallback

A callback type to encrypt a block for a specific cipher.

## Definition

```
#include <libsecure.h>
typedef SceLibSecureErrorType
(*SceLibSecureCryptographyEncryptBlockCallback)(
    SceLibSecureBlock *context,
    SceLibSecureMessage *data,
    const SceLibSecureCipherKey *k,
    size_t nb_rounds
);
```

## Arguments

<i>context</i>	The context to use to encrypt the block.
<i>data</i>	The block of data to encrypt.
<i>k</i>	The key to use for the encryption (same as the one set in the context).
<i>nb_rounds</i>	The number of rounds (depending on the cipher, this may not be used).

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

This callback function is called by the function [sceLibSecureCryptographyEncrypt\(\)](#) when the appropriate cipher is used. This is used to add new ciphers in libsecure.

## Notes

When additional ciphers are implemented, a function matching this prototype is required.

## See Also

[SceLibSecureCryptographyDecryptBlockCallback](#),  
[SceLibSecureCryptographyGetContextSizeCallback](#),  
[SceLibSecureCryptographySetContextCallback](#), [sceLibSecureCryptographyEncrypt](#)

SCE CONFIDENTIAL

---

# SceLibSecureCryptographyGenerateKeyCallback

---

A callback type to generate a key for a specific cipher.

## Definition

---

```
#include <libsecure.h>
typedef SceLibSecureErrorType (*SceLibSecureCryptographyGenerateKeyCallback) (
    SceLibSecureCipherKey *k
);
```

## Arguments

---

*k*                      The key memory block with a size matching the size returned by [SceLibSecureCryptographyGetKeySizeCallback](#).

## Return Values

---

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

---

This callback function is called by the function [sceLibSecureCryptographyGenerateKey\(\)](#) when the appropriate cipher is used. This is used to add new ciphers in libsecure.

## Notes

---

When additional ciphers are implemented, a function matching this prototype is only required if the cipher needs a specific key generation algorithm to be used.

## See Also

---

[SceLibSecureCryptographyGetKeySizeCallback](#),  
[sceLibSecureCryptographyGenerateKey](#), [sceLibSecureCryptographyGetKeySize](#)

# SceLibSecureCryptographyGetContextSizeCallback

A callback type to retrieve the context size for a specific cipher.

## Definition

```
#include <libsecure.h>
typedef SceLibSecureErrorType
(*SceLibSecureCryptographyGetContextSizeCallback)(
    const SceLibSecureCipherKey *k,
    size_t nb_rounds,
    size_t *mem_size
);
```

## Arguments

<i>k</i>	The key to use for the encryption and the decryption process.
<i>nb_rounds</i>	The number of rounds (depending on the cipher, this may not be used).
<i>mem_size</i>	The context size required.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

This callback function is called by the function [sceLibSecureCryptographyGetContextSize\(\)](#) when the appropriate cipher is used. This is used to add new ciphers in libsecure.

## Notes

When additional ciphers are implemented, a function matching this prototype is only required if a context is needed.

## See Also

[SceLibSecureCryptographyEncryptBlockCallback](#),  
[SceLibSecureCryptographyDecryptBlockCallback](#),  
[SceLibSecureCryptographySetContextCallback](#),  
[sceLibSecureCryptographyGetContextSize](#)

SCE CONFIDENTIAL

---

# SceLibSecureCryptographyGetKeySizeCallback

---

The callback type to retrieve the key size for a specific cipher.

## Definition

---

```
#include <libsecure.h>
typedef SceLibSecureErrorType (*SceLibSecureCryptographyGetKeySizeCallback)(
    size_t key_size,
    size_t *size_in_bytes
);
```

## Arguments

---

*key\_size*            The logical size of the key in bytes.  
*size\_in\_bytes*      The physical size of the key required.

## Return Values

---

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

---

This callback function is called by the function [sceLibSecureCryptographyGetKeySize\(\)](#) when the appropriate cipher is used. This is used to add new ciphers in libsecure.

## Notes

---

When additional ciphers are implemented, a function matching this prototype is only required if the cipher needs a specific key generation algorithm to be used.

The logical size of the key (generally expressed in bits) is different to the physical size of the key required to represent this key in memory.

## See Also

---

[SceLibSecureCryptographyGenerateKeyCallback](#),  
[sceLibSecureCryptographyGenerateKey](#), [sceLibSecureCryptographyGetKeySize](#)



SCE CONFIDENTIAL

# SceLibSecureCryptographySetContextCallback

A callback type to set the context for a specific cipher.

## Definition

```
#include <libsecure.h>
typedef SceLibSecureErrorType (*SceLibSecureCryptographySetContextCallback)(
    SceLibSecureBlock *context,
    const SceLibSecureCipherKey *k,
    size_t nb_rounds
);
```

## Arguments

<i>context</i>	The context to set with a size matching the size returned by <a href="#">SceLibSecureCryptographyGetContextSizeCallback</a> .
<i>k</i>	The key to use for the encryption and the decryption process.
<i>nb_rounds</i>	The number of rounds (depending on the cipher, this may not be used).

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

This callback function is called by the function [sceLibSecureCryptographySetContext\(\)](#) when the appropriate cipher is used. This is used to add new ciphers in libsecure.

## Notes

When additional ciphers are implemented, a function matching this prototype is only required if a context is needed.

## See Also

[SceLibSecureCryptographyEncryptBlockCallback](#),  
[SceLibSecureCryptographyDecryptBlockCallback](#),  
[SceLibSecureCryptographyGetContextSizeCallback](#),  
[sceLibSecureCryptographySetContext](#)

SCE CONFIDENTIAL

# SceLibSecureHashAddDataCallback

A callback type to add more data to a specific hash.

## Definition

```
#include <libsecure.h>
typedef SceLibSecureErrorType (*SceLibSecureHashAddDataCallback) (
    SceLibSecureBlock *context,
    const SceLibSecureMessage *data
);
```

## Arguments

<i>context</i>	The context to use to add the message to.
<i>data</i>	The message to add to the hash.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

This callback function is called by the function [sceLibSecureHashAddMessage\(\)](#) when the appropriate hash is used. This is used to add a new hash in libsecure.

## Notes

When additional hashes are implemented, a function matching this prototype is required.

## See Also

[SceLibSecureHashGetContextSizeCallback](#), [SceLibSecureHashSetContextCallback](#), [SceLibSecureHashGetDigestSizeCallback](#), [SceLibSecureHashGetDigestCallback](#), [sceLibSecureHashAddMessage](#)

SCE CONFIDENTIAL

---

# SceLibSecureHashGetContextSizeCallback

---

A callback type to retrieve the context size for a specific hash.

## Definition

---

```
#include <libsecure.h>
typedef SceLibSecureErrorType (*SceLibSecureHashGetContextSizeCallback)(
    size_t *mem_size
);
```

## Arguments

---

*mem\_size*      The context size required.

## Return Values

---

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

---

This callback function is called by the function [sceLibSecureHashGetContextSize\(\)](#) when the appropriate hash is used. This is used to add new hash in libsecure.

## Notes

---

When additional hashes are implemented, a function matching this prototype is required.

## See Also

---

[SceLibSecureHashSetContextCallback](#), [SceLibSecureHashAddDataCallback](#),  
[SceLibSecureHashGetDigestSizeCallback](#), [SceLibSecureHashGetDigestCallback](#),  
[sceLibSecureHashGetContextSize](#)

SCE CONFIDENTIAL

# SceLibSecureHashGetDigestCallback

A callback type to retrieve the digest for a specific hash.

## Definition

```
#include <libsecure.h>
typedef SceLibSecureErrorType (*SceLibSecureHashGetDigestCallback) (
    SceLibSecureBlock *context,
    SceLibSecureDigest *digest
);
```

## Arguments

<i>context</i>	The context to use to retrieve the digest from.
<i>digest</i>	The output digest.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

This callback function is called by the function [sceLibSecureHashGetDigest\(\)](#) when the appropriate hash is used. This is used to add new hashes in libsecure.

## Notes

When additional hashes are implemented, a function matching this prototype is required.

## See Also

[SceLibSecureHashGetContextSizeCallback](#), [SceLibSecureHashSetContextCallback](#), [SceLibSecureHashAddDataCallback](#), [SceLibSecureHashGetDigestSizeCallback](#), [sceLibSecureHashGetDigest](#)

SCE CONFIDENTIAL

---

# SceLibSecureHashGetDigestSizeCallback

---

A callback type to retrieve the digest length for a specific hash.

## Definition

---

```
#include <libsecure.h>
typedef SceLibSecureErrorType (*SceLibSecureHashGetDigestSizeCallback)(
    size_t *mem_size
);
```

## Arguments

---

*mem\_size*      The digest length in bytes.

## Return Values

---

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

---

This callback function is called by the function [sceLibSecureHashGetDigestSize\(\)](#) when the appropriate hash is used. This is used to add new hashes in libsecure.

## Notes

---

When additional hashes are implemented, a function matching this prototype is required.

## See Also

---

[SceLibSecureHashGetContextSizeCallback](#), [SceLibSecureHashSetContextCallback](#),  
[SceLibSecureHashAddDataCallback](#), [SceLibSecureHashGetDigestCallback](#),  
[sceLibSecureHashGetDigestSize](#)

SCE CONFIDENTIAL

---

# SceLibSecureHashSetContextCallback

---

A callback type to set the context for a specific hash.

## Definition

---

```
#include <libsecure.h>
typedef SceLibSecureErrorType (*SceLibSecureHashSetContextCallback) (
    SceLibSecureBlock *context
);
```

## Arguments

---

*context*      The context to set with a size matching the size returned by [SceLibSecureHashGetContextSizeCallback](#).

## Return Values

---

[SceLibSecureErrorType](#) - Returns a libsecure error code.

## Description

---

This callback function is called by the function [sceLibSecureHashSetContext\(\)](#) when the appropriate hash is used. This is used to add new hashes in libsecure.

## Notes

---

When additional hashes are implemented, a function matching this prototype is required.

## See Also

---

[SceLibSecureHashGetContextSizeCallback](#), [SceLibSecureHashAddDataCallback](#),  
[SceLibSecureHashGetDigestSizeCallback](#), [SceLibSecureHashGetDigestCallback](#),  
[sceLibSecureHashSetContext](#)

# SceLibSecureMaskGenerationFunctionCallback

A callback type for generating a mask from a message.

## Definition

```
#include <libsecure.h>
typedef SceLibSecureErrorType (*SceLibSecureMaskGenerationFunctionCallback)(
    SceLibSecureMessage *data,
    SceLibSecureMessage *mask,
    void *user
);
```

## Arguments

<i>data</i>	The input message used to generate the mask from.
<i>mask</i>	The output message receiving the mask that is generated.
<i>user</i>	The user data which may be used by the mask generation function.

## Return Values

[SceLibSecureErrorType](#) – Returns a libsecure error code.

## Description

This type should be used to define a function used for some padding schemes.

## Notes

At the moment, only the OAEP passing scheme may use this callback function.

## See Also

[SceLibSecurePaddingOption](#), [SceLibSecureMaskGenerationFunctionCallback](#)