libslbsss

Library Specification

Version A.0
11 / 2024

CONTENTS

1 Overview

(k, n) threshold secret sharing scheme library.
It is a library that can be used with clang / gcc on Linux and Visual Studio on Windows (R).
Supports x64 and x86 processor architectures.

2 Principle of (k, n) threshold secret sharing scheme

The following polynomials are provided as a threshold value and a number of shares.

$$q(x_i) = a_0 + \sum_{j=1}^{k-1} a_j x_i^{\mathrm{j}} (mod\ p)$$

$i = 1,2, \dots, n$
$2 \leq \mathrm{n} < \mathrm{p}$
$2 \leq \mathrm{k} < \mathrm{p}$
$\mathrm{k} \leq \mathrm{n}$
$0 \leq a_{\mathrm{j}} < \mathrm{p}$
$1 \leq \mathrm{x}_{\mathrm{i}} < \mathrm{p}$

When $\mathrm{q} \neq \mathrm{r}$, $\mathrm{x}_q \neq \mathrm{x}_r$

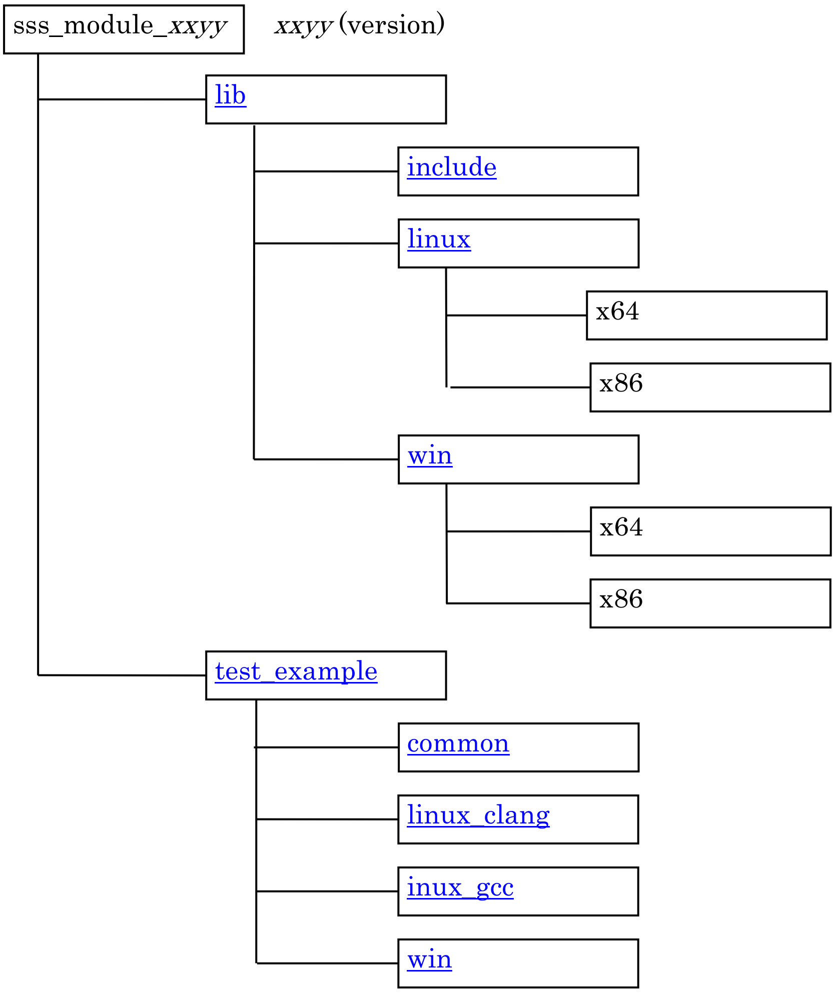
This library creates a finite field where p (prime number) is 65537, and performs arithmetic processing on that finite field.

When encoding, a caller provides the x_{i} and the 16-bit value a_0 that must be kept secret, then this library sets random values to $a_1, a_2, \dots, a_{k-1}$, calculates the $q(x_i)$ for each x_{i} and outputs it as a 17-bit value.

For convenience, this specification refers to the $q(x_i)$ distribution destination as "players".

When decoding, a caller provides the x_{i} and the $q(x_i)$ for each x_{i}, then this library solves the simultaneous polynomials to take a_0 and outputs it as a 16-bit value.

3 sss_module folder tree



(1) lib

Contains the library provided by our company.

(a) include

Contains header files need to include when using the library.
Just include libslbsss.h and all header files will be included.

(b) linux

Contains the shared library for x64 and x86 in Linux.

(c) win

Contains the DLL and import library for x64 and x86 in Windows (R).

(2) test_example

Contains a test example file using the library. The file is a test program that uses the main functions, and when the startup option "-m" is added, it becomes speed measurement mode.

Please adjust each parameter in the source code according to the performance of your PC.

(a) common

Contains common source files of the test example for Linux and Windows (R).

(b) linux_clang / linux_gcc

Contains makefiles for clang and gcc in Linux.
Use makefile64 when compiling for x64, and makefile86 when compiling for x86.

(c) win

Contains the project file for Visual Studio in Windows (R).
The project file has x64 and x86 as solution platforms.

The solution file is for Visual Studio 2022, but it can also be built with Visual Studio 2019, 2017, and 2015. If you use 2019, 2017, or 2015, change the "Windows SDK Version" and "Platform Toolset" in the project properties to the appropriate ones.

To run the built EXE file, you need the OpenMP runtime "vcomp140.dll" that is included as a redistributable file in Visual Studio 2022, 2019, 2017, and 2015.

4 Data Types

Type	Explanation	Bits number	Range
slb_uint8_t	8 bits unsigned integer.	8	0 / SLB_UCHAR_MAX
slb_int8_t	8 bits signed integer.	8	-127 / +127
slb_uint16_t	16 bits unsigned integer.	16	0 / SLB_USHRT_MAX
slb_int16_t	16 bits signed integer.	16	−32,767 / +32,767
slb_uint32_t	32 bits unsigned integer.	32	0 / SLB_UINT32_MAX
slb_int32_t	32 bits signed integer.	32	−2,147,483,647 / SLB_INT_MAX
slb_uint_t	32 bits unsigned integer.	32	0 / SLB_UINT_MAX
slb_int_t	32 bits signed integer.	32	−2,147,483,647 / SLB_INT_MAX
slb_uint64_t	64 bits unsigned integer.	64	0 / SLB_ULLONG_MAX
slb_int64_t	64 bits signed integer.	64	−9,223,372,036,854,775,807 / +9,223,372,036,854,775,807
slb_uintptr_t	Unsigned integer representing an address	32 / 64 (OS bits)	0 / SLB_UINT32_MAX or SLB_ULLONG_MAX
slb_bool_t	Boolean	32	SLB_FALSE(0) / SLB_TRUE(1)
SLB_RC	Result code from a function	32	See Result Codes.
H_SLB_SSS	Handle structure (For type safety, not true constructs)	32 / 64 (OS bits)	

5 Constants

Symbol	Explanation
SLB_NULL	null pointer
SLB_FALSE	value (0) of slb_bool_t
SLB_TRUE	value (1) of slb_bool_t
SLB_UCHAR_MAX	maximum value of uint8
SLB_USHRT_MAX	maximum value of uint16
SLB_INT_MAX	maximum value of int
SLB_UINT_MAX	maximum value of uint
SLB_UINT32_MAX	maximum value of uint32
SLB_ULLONG_MAX	maximum value of uint64
SLB_BITS_OF_NIBBLE	number of nibble bits
SLB_BITS_OF_UINT8	number of bits in uint8
SLB_BITS_OF_UINT16	number of bits in uint16
SLB_BITS_OF_UINT32	number of bits in uint32
SLB_BIT_MSB_OF_UINT8	MSB bit of uint8
SLB_BIT_LSB_OF_UINT8	LSB bit of uint8
SLB_BIT_MSB_OF_UINT16	MSB bit of uint16
SLB_BIT_LSB_OF_UINT16	LSB bit of uint16
SLB_BIT_MSB_OF_UINT32	MSB bit of uint32
SLB_BIT_LSB_OF_UINT32	LSB bit of uint32
SLB_SHIFT_OF_1B	1 byte shift size
SLB_SHIFT_OF_2B	2 byte shift size
SLB_SHIFT_OF_3B	3 byte shift size
SLB_SHIFT_OF_4B	4 byte shift size
SLB_SHIFT_OF_5B	5 byte shift size
SLB_SHIFT_OF_6B	6 byte shift size
SLB_SHIFT_OF_7B	7 byte shift size
SLB_MASK_OF_UINT8	uint8 mask
SLB_MASK_OF_UINT16	uint16 mask
SLB_MASK_OF_UINT32	uint32 mask

6 Enumerations

Symbol	Explanation	Enumerator
SLB_MP_TYPE	Multi process type	**MP_NONE** Not use MP **MP_OMP** OpenMP
SLB_SSS_HANDLE_STATE	Handle state of SSS module	**HS_SSS_INVALID** Invalid handle **HS_SSS_ENCODE** Encoding handle **HS_SSS_ENCODE_STARTED** Encoding handle (started) **HS_SSS_DECODE** Decoding handle **HS_SSS_DECODE_STARTED** Decoding handle (started)

7 Macros

Symbol	Explanation
SLB_R_SUCCEEDED(r)	Judges the success of return code(r).
SLB_R_FAILED(r)	Judges the failure of return code(r).
SLB_R_NOT_FATAL(r)	Judges the non fatal of return code(r).
SLB_UNREFERENCED(p)	Declares unused parameter(p).
SLB_MEMBER_OFFSET (str, member)	Gets the offset value to the specified member in the structure(str).
SLB_WRITE16(pd, val)	Writes a 16-bit value(val) to pointer(pd).
SLB_WRITE32(pd, val)	Writes a 32-bit value(val) to pointer(pd).
SLB_READ16(ps, p_val)	Reads a 16-bit value(p_val) from pointer(ps).
SLB_READ32(ps, p_val)	Reads a 32-bit value(p_val) from pointer(ps).

8 Result Codes

Symbol	Explanation
R_SUCCESS	Success.
R_TERMINATE	Higher data end detection.
R_COMPLETE	Coding complete.
R_RAND_FAIL	Random number generation not possible.
R_LOW_MEMORY	Insufficient memory.
R_NOT_CONFIG	Not configured.
R_INVALID_HANDLE	Invalid handle.
R_INVALID_PARAM	Invalid parameter.
R_SSS_STOP	User ordered stop.
R_SSS_INVALID_X	Incorrect x coordinate.
R_SSS_IDENTIC_X	Have the same x-coordinate.
R_SSS_NOT_STARTED	Not started.
R_SSS_NOT_INIT	Not initialized.

9 Functions

The functions listed below are not thread-safe except libslbsss_get_version() slb_sss_get_max_players_encode() and slb_sss_get_max_players_decode().

9.1 libslbsss_get_version (libslbsss.h)

Gets the version of .libslbsss.

(1) Syntax

const char* libslbsss_get_version();

(2) Return value

Version string of libslbsss.

(3) Description

This function returns the pointer to the null-terminated string indicating the version of libslbsss.

9.2 slb_config (slb.h)

Configures common functions in SLB.

(1) Syntax

```
void slb_config(
        SLB_ALLOC     alloc_func,
        SLB_FREE      free_func
);
```

(2) Parameters

alloc_func
User-provided allocate memory function to be called back.
See alloc_func().

free_func
User-provided free memory function to be called back.
See free_func().

(3) Return value

This function does not return a value.

(4) Description

This function sets user-specified allocate / free memory functions.
When using this library, call it only once for the first time.

9.3 alloc_func (slb.h)

User-provided allocate memory function to be called back.

(1) Syntax

```
void* alloc_func(
        void*           param,
        slb_uint_t      size
);
```

(2) Parameters

param

The user's arbitrary address specified when opening each SLB module.

size

Byte size that must be allocated.

(3) Return value

Pointer to allocated memory.

If memory cannot be allocated , return SLB_NULL.

9.4 free_func (slb.h)

User-provided free memory function to be called back.

(1) Syntax

```
void free_func(
        void*           param,
        void*           mem,
        slb_uint_t      size,
        slb_bool_t      cleared
);
```

(2) Parameters

param

The user's arbitrary address specified when opening each SLB module.

mem

Pointer to memory returned by alloc_func().

size

Allocated byte size by alloc_func().

cleared

If =SLB_TRUE, the memory contents has been zero cleated.

(3) Return value

This function does not return a value.

9.5 slb_is_config (slb.h)

Returns whether the configuration is completed.

(1) Syntax

slb_bool_t slb_is_config();

(2) Return value

=SLB_TRUE: configured
=SLB_FALSE: not configured

(3) Description

This function returns whether or not configuration has been completed.

If the memory operation functions is set, it is determined that configuration has been completed.

9.6 slb_alloc (slb.h)

Allocates memory.

(1) Syntax

```
void* slb_alloc(
        void*           param,
        slb_uint_t      size
);
```

(2) Parameters

param

User-specified parameter.

size

Allocation byte size.

(3) Return value

Allocated memory.

Returns SLB_NULL if allocation fails.

(4) Description

This function allocates memory using alloc_func() configured in slb_config() .

Note that this function does not check whether alloc_func() is set by slb_config() .

9.7 slb_alloc_aligned (slb.h)

Allocates memory with an alignment specification.

(1) Syntax

```
void* slb_alloc_aligned(
          void*            param,
          slb_uint_t       size,
          slb_uint_t       alignment
);
```

(2) Parameters

param

User-specified parameter.

size

Allocation byte size.

alignment

Alignment value. Must be a power of 2 and less than or equal to 128.

(3) Return value

Allocated memory.

Returns SLB_NULL if allocation fails.

(4) Description

This function allocates memory aligned to the specified alignment using alloc_func() configured in slb_config().

Because alloc_func() is called with a size that considers alignment, the alloc_func() side does not need to consider alignment.

Note that this function does not check whether alloc_func() is set by slb_config().

9.8 slb_free (slb.h)

Frees the memory.

(1) Syntax

```
void slb_free(
          void*             param,
          void*             mem,
          slb_bool_t        clear
);
```

(2) Parameters

param

User-specified parameter.

mem

The pointer to the memory previously allocated with slb_alloc() or slb_alloc_aligned().

clear

=SLB_TRUE: Erase before release.

(3) Return value

This function does not return a value.

(4) Description

This function frees the memory using the free_func() configured in slb_config().
Note that this function does not check whether free_func() is set by slb_config().

9.9 slb_get_nmb_of_cores (slb.h)

Gets number of logical cores..

(1) Syntax

```
slb_int_t slb_get_nmb_of_cores(void);
```

(2) Return value

Number of logical cores.

9.10 slb_sss_get_config (slb_sss.h)

Gets SSS configuration.

(1) Syntax

```
void slb_sss_get_config(
        SLB_SSS_CONFIG*      p_config
);
```

(2) Parameters

p_config

The configuration on the next page contents are stored.

The default values obtained by this function were derived from testing in our environment.

Please change them to values that are judged to be efficient according to the user's environment.

(3) Return value

This function does not return a value.

Member	Contents
enc_paran_ratio_to_cores	In encoding, ratio of the number of parallel processes to the number of logical cores. Range: 1 - 100 The multiplication result saturates at 256.
enc_cores_ratio_to_k	In encoding, ratio of the number of used cores to the threshold. Range: 1 - 100 The multiplication result saturates at the number of logical cores.
dec_paran_ratio_to_cores	In decoding, ratio of the number of parallel processes to the number of logical cores. Range: 1 - 100 The multiplication result saturates at 256.
dec_cores_ratio_to_k	In decoding, ratio of the number of used cores to the threshold. Range: 1 - 100 The multiplication result saturates at the number of logical cores.
dec_paran_expand_limit_k	In decoding, the threshold to allow parallelism growth. Range: SLB_SSS_MIN_PLAYERS - USHRT_MAX
k_max_x_diff	In decoding, maximum threshold for which the x-coordinate difference inverse array can be used. Range: SLB_SSS_MIN_PLAYERS - 1000 Note that calling slb_sss_open_as_decode() requests the following size memory allocation. k_max_x_diff * (k_max_x_diff - 1) / 2
callback_steps	Number of boundary steps to call user callback function. Each time one data is encoded / decoded, the current step number is incremented by 1, (Strictly speaking, the add value is tuned depending on the SIMD and parallelism used) and when the total amount reaches this value, the user callback function is called to rewind the current steps to zero. Range: 1 - 1000
dec_addsteps_k_max_x_diff	In decoding, steps to add to user callback boundary steps when threshold exceeds k_max_x_diff. Range: 1 - 1000 This is provided because if the threshold exceeds k_max_x_diff, the calculation will slow down during decoding and the frequency of calling the user callback function will decrease.

9.11 slb_sss_change_config (slb_sss.h)

Changes SSS configuration.

(1) Syntax

```
SLB_RC slb_sss_change_config(
        const SLB_SSS_CONFIG*          p_config
);
```

(2) Parameters

p_config

Configuration contents.

See slb_sss_get_config().

(3) Return value

Processing result. Returns one of the following.

R_SUCCESS

R_INVALID_PARAM

9.12 slb_sss_set_simd (slb_sss.h)

Sets SIMD usage.

(1) Syntax

```
void slb_sss_set_simd(
          slb_bool_t          sse2,
          slb_bool_t          avx2,
          slb_bool_t          avx512
);
```

(2) Parameters

sse2
=SLB_TRUE: Uses SIMD SSE2.

avx2
=SLB_TRUE: Uses SIMD AVX2.

avx512
=SLB_TRUE: Uses SIMD AVX-512

(3) Return value

This function does not return a value.

(4) Description

This function sets whether or not each SIMD used for calculation can be used.

Please note that this function does not check whether the specified SIMD of the processing system CPU can be used.

This function can be called at any time, but the settings when slb_sss_open_as_encode() / slb_sss_open_as_decode() is called are adopted to the handle resource.

9.13 slb_sss_set_mp (slb_sss.h)

Sets MP type.

(1) Syntax

```
void slb_sss_set_mp(
        SLB_MP_TYPE         mp_type,
        slb_int_t           cores
);
```

(2) Parameters

mp_type
MP type

cores
The number of used cores.
=0 means the maximum number of cores.

(3) Return value

This function does not return a value.

(4) Description

This function sets the MP (parallel programming) type used for operations.
Currently OpenMP is the only MP type available.
This function can be called at any time, but the settings when slb_sss_open_as_encode() / slb_sss_open_as_decode() is called are adopted to the handle resource.

9.14 slb_sss_init_decode_res (slb_sss.h)

Initializes the common resource for decryption.

(1) Syntax

```
void slb_sss_init_decode_res();
```

(2) Return value

This function does not return a value.

(3) Description

This function initializes the common resources required for the decryption process.

If the app is supports decoding, it must initialize common resources before calling slb_sss_open_as_decode() for the first time.

This function initializes with parallel processing when MP is enabled with slb_sss_set_mp().

9.15 slb_sss_open_as_encode (slb_sss.h)

Opens as encoding.

(1) Syntax

```
H_SLB_SSS slb_sss_open_as_encode(
        const SLB_SSS_ENCODE_OPEN_PARAM*        open_param,
        SLB_RC*                                 rc
);
```

(2) Parameters

open_param

The following open parameter.

Member	Contents
k_max	Maximum k (threshold). Does not return R_INVALID_PARAM even if k_max > n_max.
n_max	Maximum n (number of shares).
mem_param	User arbitrary address passed to the memory management function specified by slb_config().
rand_func	User-provided random function to be called back. See rand_func()
rand_param	User arbitrary address passed to rand_func().

rc

Pointer to processing result.

R_SUCCESS
R_LOW_MEMORY
R_NOT_CONFIG
R_INVALID_PARAM

(3) Return value

SSS control handle.
Returns SLB_NULL on failure.

(4) Description

This function opens SSS in encoding mode.

Once you have a control handle, you can use that handle to perform any number of encoding loops starting with slb_sss_start_encode() calls.

Allocating memories by the memory management function specified by slb_config() is performed only when opening, and freeing only when closing.

The call to rand_func() is guaranteed to be the same thread that called slb_sss_encode().

9.16 rand_func (slb.h)

User-provided generate random value function to be called back.

(1) Syntax

```
void* rand_func(
        void*           param,
        slb_uint_t      len,
        void*           rnd_buff
);
```

(2) Parameters

param

The user's arbitrary address specified when opening each SLB module.

len

Byte size that must be generated.

rnd_buff

A buffer that stores the generated random numbers.

(3) Return value

Returns SLB_TRUE if successful, SLB_FALSE if unsuccessful.

9.17 slb_sss_start_encode (slb_sss.h)

Starts encoding.

(1) Syntax

```
SLB_RC slb_sss_start_encode(
        H_SLB_SSS       handle,
        slb_uint_t      k,
        slb_uint_t      n,
        slb_bool_t      xassign,
        slb_uint16_t    x[]
);
```

(2) Parameters

handle

SSS control handle.

k

k (threshold).

n

n (number of shares).

xassign

x-coordinate assignment method.
SLB_FALSE: Store in the assignment argument x[] in ascending order from 1.
SLB_TRUE: The contents of the user-specified argument x[] are adopted.

x[]

Each x-coordinate corresponding to N data.
If SLB_FALSE is specified in xassign, this function stores in x[0] to x[n-1].
If xassign is SLB_TRUE, the user must set the x-coordinate in x[0] to x[n-1].
This function does not check the validity of the user-specified x-coordinate.

Each user-specified x-coordinate must be different within the range of 1 to the return value of slb_sss_get_max_players_decode().
This function does not check the validity of the user-specified x coordinate.

(3) Return value

Processing result. Returns one of the following.
R_SUCCESS
R_INVALID_HANDLE
R_INVALID_PARAM

(4) Description

This function starts SSS encoding.
Can be called anytime the control handle is valid.
k and n must be less than or equal to k_max and n_max specified at the time of opening.

9.18 slb_sss_encode (slb_sss.h)

Encodes user-specified data.

(1) Syntax

```
SLB_RC slb_sss_encode(
          H_SLB_SSS               handle,
          slb_int_t               nmb,
          const slb_uint16_t*     plain
          slb_uint32_t**          share
);
```

(2) Parameters

handle
SSS control handle.

nmb
Number of data.

plain
Plain data.

share
Shared data per player.
It is a double pointer, and when expressed as a two-dimensional array, it has the following structure.
share[players][data for each player]

(3) Return value

Processing result. Returns one of the following.
R_SUCCESS
R_RAND_FAIL
R_SSS_STOP
R_SSS_NOT_STARTED

(4) Description

This function encodes the specified 16 bits plain data.
Encoding must have started with slb_sss_start_encode().

Each shared data stored in share[] has a value in the range 0 to 65536, and only the LSB to 17 bits have meaning.
Other bits are guaranteed to be '0'.

This function does not consider the endianness of the processing system.

9.19 slb_sss_open_as_decode (slb_sss.h)

Opens as decoding.

(1) Syntax

```
H_SLB_SSS slb_sss_open_as_decode(
        const SLB_SSS_DECODE_OPEN_PARAM*        open_param,
        SLB_RC*                                 rc
);
```

(2) Parameters

open_param

The following open parameter.

Member	Contents
k_max	Maximum k (threshold).
mem_param	User arbitrary address passed to the memory management function specified by slb_config().

rc

Pointer to processing result.
R_SUCCESS
R_LOW_MEMORY
R_NOT_CONFIG
R_INVALID_PARAM
R_SSS_NOT_INIT (slb_sss_init_decode_res has not been called.)

(3) Return value

SSS control handle.
Returns SLB_NULL on failure.

(4) Description

This function opens SSS in decryption mode.

Once you have a control handle, you can use that handle to perform any number of decoding loops starting with slb_sss_start_decode() call.

Allocating memories by the memory management function specified by slb_config() is performed only when opening, and freeing only when closing.

9.20 slb_sss_start_decode (slb_sss.h)

Starts decoding.

(1) Syntax

```
SLB_RC slb_sss_start_decode(
        H_SLB_SSS               handle,
        slb_uint_t              k,
        const slb_uint16_t      x[]
);
```

(2) Parameters

handle

SSS control handle.

k

k (threshold).

x[]

Each x-coordinate corresponds to its shared data.
Store in x[0] to x[k-1] by the user.
Must set the passed value when encoding.

If =SLB_NULL, the value specified in the most recent slb_sss_start_decode() is adopted.
If the x-coordinate has never been specified in the past, this function returns R_SSS_INVALID_X.

(3) Return value

Processing result. Returns one of the following.
R_SUCCESS
R_INVALID_HANDLE
R_INVALID_PARAM
R_SSS_INVALID_X
R_SSS_IDENTIC_X

(4) Description

This function starts SSS decryption.
Can be called anytime the control handle is valid.
k must be less than or equal to k_max specified at the time of opening.

9.21 slb_sss_decode (slb_sss.h)

Decodes user-specified data.

(1) Syntax

```
SLB_RC slb_sss_decode(
        H_SLB_SSS               handle,
        slb_int_t               nmb,
        const slb_uint32_t**    share,
        slb_uint16_t*           plain
);
```

(2) Parameters

handle
SSS control handle.

nmb
Number of data.

share
Shared data per player.
It is a double pointer, and when expressed as a two-dimensional array, it has the following structure.
share[players][data for each player]

plain
Plain data.

(3) Return value

Processing result. Returns one of the following.
R_SUCCESS
R_SSS_STOP
R_SSS_NOT_STARTED

(4) Description

This function decodes the specified 17-bit encoded data.
Decoding must be started by slb_sss_start_decode().

Although each shared data stored in share[] has meaning from the LSB to 17 bits, other bits must be '0' for convenience of processing within this function.

This function does not consider the endianness of the processing system.

9.22 slb_sss_set_callback (slb_sss.h)

Sets callback function.

(1) Syntax

```
SLB_RC slb_sss_set_callback(
        H_SLB_SSS           handle,
        SLB_SSS_CALLBACK    events_func,
        void*               events_param
);
```

(2) Parameters

handle
SSS control handle.

events_func
User-specified function to be called back (=SLB_NULL to cancel callback).
See events_func().

events_param
User arbitrary address passed to the events_func.
See events_func().

(3) Return value

Processing result. Returns one of the following.
R_SUCCESS
R_INVALID_HANDLE

(4) Description

This function sets user-specified callback function is called at code time.
This function can be called at any time after calling slb_sss_open_as_encode() / slb_sss_open_as_decode().

Calling events_func is guaranteed to be the same thread that called slb_sss_encode() or slb_sss_decode().

9.23 events_func (slb.h)

User-specified function to be called back.

(1) Syntax

```
slb_bool_t events_func (
        void*     events_param
);
```

(2) Parameters

events_param
The events_param value specified by slb_sss_set_callback() is stored.

(3) Return value

Whether or not to abort processing.
The user must return SLB_TRUE when it wants to abort processing.

9.24 slb_sss_close (slb_sss.h)

Closes the SSS control handle.

(1) Syntax

```
void slb_sss_close(
        H_SLB_SSS      handle
);
```

(2) Parameters

handle

SSS control handle.

(3) Return value

This function does not return a value.

(4) Description

This function frees the memories allocated by slb_sss_open_as_encode() / slb_sss_open_as_decode().

Must call when the handle is no longer needed.

9.25 slb_sss_get_handle_state (slb_sss.h)

Gets the handle state.

(1) Syntax

```
SLB_SSS_HANDLE_STATE slb_sss_get_handle_state(
        H_SLB_SSS       handle
);
```

(2) Parameters

handle

SSS control handle.

(3) Return value

Returns one of the SLB_SSS_HANDLE_STATE enumerators.

(4) Description

This function returns the state of the SSS control handle.

SLB-SSS does not check the validity of control handles in the encoding and decoding functions.

9.26 slb_sss_rand (slb_sss.h)

Generates random numbers.

(1) Syntax

```
SLB_RC slb_sss_rand(
        H_SLB_SSS       handle,
        slb_uint_t      len,
        void*           rnd_buff
);
```

(2) Parameters

handle

SSS control handle.
This must be encoding handle.

len

Acquisition byte size.

rnd_buff

Random number storage buffer.

(3) Return value

Processing result. Returns one of the following.
R_SUCCESS
R_INVALID_HANDLE
R_RAND_FAIL

(4) Description

This function gets random numbers using the user-provided random function.

In addition to the random numbers handled by SLB-SSS, call when the user needs random numbers.

9.27 slb_sss_get_bestnmb (slb_sss.h)

Gets the best number of data for the maximum number of data used by caller.

(1) Syntax

```
slb_int_t slb_sss_get_bestnmb(
        H_SLB_SSS       handle,
        slb_int_t       maxnmb
);
```

(2) Parameters

handle

SSS control handle.

maxnmb

Maximum number of data used in the caller coding loop.

(3) Return value

Best number.

(4) Description

The optimal number varies depending on the SIMD vector length and the number of parallel processing used.

This function returns the maximum value that will be an integer multiple of optimal number within the range of maxnmb.

This function returns the significant value, only if slb_sss_start_encode() or slb_sss_start_decode() has been called.

When coding long data continuously, it is desirable to specify the number of data by the return value of this function except for the last coding.

9.28 slb_sss_start_statistics (slb_sss.h)

Starts statistics update.

(1) Syntax

```
SLB_RC slb_sss_start_statistics(
        H_SLB_SSS       handle,
        slb_int_t       maxnmb
);
```

(2) Parameters

handle

SSS control handle.
The statistical information is independent for each handle.

maxnmb

Maximum number of data used in the caller coding loop.

(3) Return value

Processing result. Returns one of the following.
R_SUCCESS
R_INVALID_HANDLE

(4) Description

This function starts the statistical information update process.
Calling this function clears the statistical information up to that point.

9.29 slb_sss_stop_statistics (slb_sss.h)

Stops statistics update.

(1) Syntax

```
SLB_RC slb_sss_stop_statistics(
        H_SLB_SSS       handle
);
```

(2) Parameters

handle

SSS control handle.
The statistical information is independent for each handle.

(3) Return value

Processing result. Returns one of the following.
R_SUCCESS
R_INVALID_HANDLE

(4) Description

This function stops the statistical information update process.

9.30 slb_sss_get_statistics (slb_sss.h)

Gets current statistics.

(1) Syntax

```
SLB_RC slb_sss_get_statistics(
        H_SLB_SSS                   handle,
        SLB_SSS_STATISTICS*         p_stat,
        slb_uint_t                  bytes
);
```

(2) Parameters

handle

SSS control handle.
The statistical information is independent for each handle.

p_stat

The statistical information shown on the next page are stored.

bytes

Byte size of *p_stat.

Member	Contents
maxnmb	User-specified maximum number of data.
optimalnmb	Optimal number of data per one coding calculated from maxnmb.
bestnmb	Number of best data calculated from optimalnmb.
called_cnt	Number of times slb_sss_encode() / slb_sss_decode() were called.
best_called_cnt	The number of best data calculated from maxnmb, optimalnmb. The value will be an integer multiple of optimalnmb within the range of maxnmb.
max_nmb_called	Maximum number of data when slb_sss_encode() / slb_sss_decode() are called.
coding_cnt	Number of calls to lower coding functions in the data processing loop within slb_sss_encode() / slb_sss_decode().
best_coding_cnt	The number of times the lower coding function was called with the best data count in the data processing loop within slb_sss_encode() / slb_sss_decode().
optimal_cnt	Number of times the data processing loop in slb_sss_encode() / slb_sss_decode() called the lower coding function with the optimal number of data.
best_optimal_cnt	Number of times the data processing loop in slb_sss_encode() / slb_sss_decode() called with the best data count state AND it called the lower coding function with the optimal data count.
sse2_cnt	Number of times SSE2 usage function was called in the data processing loop within slb_sss_encode() / slb_sss_decode().
avx2_cnt	Number of times AVX2 usage function was called in the data processing loop within slb_sss_encode() / slb_sss_decode().
avx512_cnt	Number of times AVX-512 usage function was called in the data processing loop within slb_sss_encode() / slb_sss_decode().
parallel_cnt	Number of times parallel processing usage function was called in the data processing loop within slb_sss_encode() / slb_sss_decode().
max_cores	Maximum number of logical cores have used by data processing loops in slb_sss_encode() / slb_sss_decode().
max_paran	Maximum number of parallelism in data processing loops in slb_sss_encode() / slb_sss_decode().

(3) Return value

Processing result. Returns one of the following.

R_SUCCESS
R_INVALID_HANDLE

9.31 slb_sss_get_max_players_encode (slb_sss.h)

Gets maximum number of players for encoding.

(1) Syntax

```
slb_uint_t slb_sss_get_max_players_encode();
```

(2) Return value

Maximum number of players.

9.32 slb_sss_get_max_players_decode (slb_sss.h)

Gets maximum number of players for decoding.

(1) Syntax

```
slb_uint_t slb_sss_get_max_players_decode();
```

(2) Return value

Maximum number of players.

9.33 slb_sss_get_encode_param (slb_sss.h)

Gets the open parameter in encoding.

(1) Syntax

```
SLB_RC slb_sss_get_encode_param(
        H_SLB_SSS                      handle,
        SLB_SSS_ENCODE_OPEN_PARAM*     open_param
);
```

(2) Parameters

handle

SSS control handle.

open_param

The open parameter is stored.
See slb_sss_open_as_encode()

(3) Return value

Processing result. Returns one of the following.
R_SUCCESS
R_INVALID_HANDLE

9.34 slb_sss_get_decode_param (slb_sss.h)

Gets the open parameter in decoding.

(1) Syntax

```
SLB_RC  slb_sss_get_decode_param(
        H_SLB_SSS                      handle,
        SLB_SSS_DECODE_OPEN_PARAM*     open_param
);
```

(2) Parameters

handle

SSS control handle.

open_param

The open parameter is stored.
See slb_sss_open_as_decode()

(3) Return value

Processing result. Returns one of the following.
R_SUCCESS
R_INVALID_HANDLE

9.35 slb_sss_get_info_k_max (slb_sss.h)

Gets k_max of the open parameter.

(1) Syntax

```
slb_uint_t slb_sss_get_info_k_max(
        H_SLB_SSS handle
);
```

(2) Parameters

handle

SSS control handle.

(3) Return value

k_max (maximum threshold).

9.36 slb_sss_get_info_n_max (slb_sss.h)

Gets n_max of the open parameter in encoding, or the same value as the return value of slb_sss_get_max_players_decode() in decoding.

(1) Syntax

```
slb_uint_t slb_sss_get_info_n_max(
        H_SLB_SSS handle
);
```

(2) Parameters

handle

SSS control handle.

(3) Return value

n_max (maximum number of shares).

9.37 slb_sss_get_info_k (slb_sss.h)

Gets k (threshold).

(1) Syntax

```
slb_uint_t slb_sss_get_info_k(
        H_SLB_SSS handle
);
```

(2) Parameters

handle

SSS control handle.

(3) Return value

k (threshold).

(4) Description

This function returns the significant value , only if slb_sss_start_encode() or slb_sss_start_decode() has been called.

9.38 slb_sss_get_info_n (slb_sss.h)

Gets n (number of shares).

(1) Syntax

```
slb_uint_t slb_sss_get_info_n(
        H_SLB_SSS handle
);
```

(2) Parameters

handle

SSS control handle.

(3) Return value

n (number of shares).

(4) Description

This function returns the significant value , only if slb_sss_start_encode() has been called.

Specifying a decryption handle always returns zero.

9.39 slb_sss_get_info_x (slb_sss.h)

Gets x-coordinate.

(1) Syntax

```
slb_uint16_t slb_sss_get_info_x(
          H_SLB_SSS       handle,
          slb_uint_t      index
);
```

(2) Parameters

handle

SSS control handle.

index

X-coordinate index (0-)

(3) Return value

X-coordinate.

(4) Description

This function returns the significant value, only if slb_sss_start_encode() or slb_sss_start_decode() has been called.

Specify the index of the x-coordinate specified when calling slb_sss_start_encode() or slb_sss_start_decode().

Even if R_SSS_INVALID_X / R_SSS_IDENTIC_X is returned at the start of decryption, that incorrect X coordinate can be obtained with this function.

10 License Agreement

Enactment Date: 11 / 2024

libslbsss (referred to as "this software") is software manufactured by Some Fellow System, Inc. (referred to as "our company").

Under the condition that the person is agreed to this contract, we grant a license to use this software to the person (referred to as the "user") who receives this software.

[License]

Only those who agree with this Agreement, anyone can use this software freely.

This software is free to use regardless of whether it is non-commercial or commercial, and must not require permission from our company.

[Prohibitions]

The user shall not do the following:

(a) Modifications, reverse engineering, decompilation, and disassembly of this software.
(b) Use of this software for criminal actions, such as hacking.

[Warranty]

(a) Our company makes no warranty with respect to this software, including warranties such as merchantability and fitness for a particular purpose.

[Limitation of Liability and Indemnity]

(a) Our company does not take any liability for the failure of electronic devices such as computers, information loss, data corruption, or any other damage or loss caused by the use or inability to use this software.
(b) If our company, the user, or a third party detects a defect or document error in the Software, our company is not obligated to correct it.

[Agreement of contract]

(a) By using this software, user agree to this agreement. If user does not agree, stop using this software and delete this software.

[Governing law]

(a) This agreement shall be governed by Japanese law.

11 Copyright

Intellectual property rights such as the copyright of libslbsss and this document are owned by Some Fellow System, Inc. This software and this document are protected by copyright law, international copyright treaties, and other intangible property laws and treaties.