

---

**libsc**

***Release 0.0.0***

**IAIK**

**May 27, 2019**



# CONTENTS

<b>1</b>	<b>API</b>	<b>3</b>
	<b>Index</b>	<b>11</b>



libsc is a library written in C in order to speed-up the development of microarchitectural side-channel attacks



struct **libsc\_eviction\_strategy**  
Structure containing the access strategy for eviction.

struct **libsc\_eviction\_set\_s**  
Structure containing virtual addresses of an eviction set.

**libsc\_timestamp\_function\_t**  
Timestamp function

struct **libsc\_s**  
Structure containing all information of a libsc instance.

int **llc\_miss\_threshold**  
LLC Miss timing information

int **llc\_hit\_threshold**  
LLC Hit timing information

*libsc\_timestamp\_function\_t* **timestamp**  
High-resolution timestamp function

**libsc\_timer\_t**  
Available sources for high-resolution timestamps

**LIBSC\_TIMER\_NATIVE**  
Native timer

**LIBSC\_TIMER\_COUNTING\_THREAD**  
Counting thread

**LIBSC\_TIMER\_PERF**  
Linux perf interface

**LIBSC\_TIMER\_MONOTONIC\_CLOCK**  
Monotonic clock

libsc\_t \* **libsc\_init** ()  
Initializes libsc context

**Returns** Returns a libsc context

void **libsc\_cleanup** (libsc\_t \* *ctx*)  
Clean-up libsc context

**Parameters**

- **ctx** – The libsc context

void **libsc\_flush** (libsc\_t \* *ctx*, void \* *addr*)

Flushes the given address from the cache

**Parameters**

- **ctx** – The libsc context
- **addr** – The address to flush

void **libsc\_flush\_b** (libsc\_t \* *ctx*, void \* *addr*)

Flushes the given address from the cache (with memory barriers)

**Parameters**

- **ctx** – The libsc context
- **addr** – The address to flush

void **libsc\_access** (libsc\_t \* *ctx*, void \* *addr*)

Accesses the given address

**Parameters**

- **ctx** – The libsc context
- **addr** – The address The address to load

void **libsc\_access\_b** (libsc\_t \* *ctx*, void \* *addr*)

Accesses the given address (with memory barriers)

**Parameters**

- **ctx** – The libsc context
- **addr** – The address

void **libsc\_barrier\_start** (libsc\_t \* *ctx*)

Begin memory barrier

**Parameters**

- **ctx** – The libsc context

void **libsc\_barrier\_end** (libsc\_t \* *ctx*)

End memory barrier

**Parameters**

- **ctx** – The libsc context

uint64\_t **libsc\_timestamp** (libsc\_t \* *ctx*)

Returns the timestamp (depending on the used timer)

**Parameters**

- **ctx** – The libsc context

**Returns** The current timestamp

void **libsc\_measure\_start** (libsc\_t \* *ctx*)

Begin a timing measurement

**Parameters**

- **ctx** – The libsc context

uint64\_t **libsc\_measure\_end** (libsc\_t \* *ctx*)

End a timing measurement



**Parameters**

- **ctx** – The libsc context

**Returns** The time that has passed between the begin of the measurement

void **libsc\_set\_timer** (libsc\_t \* *ctx*, *libsc\_timer\_t* *timer*)

Set the used timer

**Parameters**

- **ctx** – The libsc context
- **timer** – The timer to use

int **libsc\_flush\_reload** (libsc\_t \* *ctx*, void \* *addr*)

Measures the access time to *addr* to distinguish between a cache hit and miss and flushes *addr*

**Parameters**

- **ctx** – The libsc context
- **addr** – The address

**Returns** 1 if the address was in the cache, 0 if the address was not cached

void **libsc\_calibrate\_flush\_reload** (libsc\_t \* *ctx*)

Calibrates the threshold to distinguish between a cache hit and cache miss using Flush+Reload

**Parameters**

- **ctx** – The libsc context

void **libsc\_speculation\_barrier** (libsc\_t \* *ctx*)

Barrier that is not crossed during speculative execution

**Parameters**

- **ctx** – The libsc context

int **libsc\_try\_start** (libsc\_t \* *ctx*)

Begin a try/catch block using TSX or signal handling

**Parameters**

- **ctx** – The libsc context

**Returns** 1 to enter the block

void **libsc\_try\_end** (libsc\_t \* *ctx*)

Ends the try/catch block

**Parameters**

- **ctx** – The libsc context

void **libsc\_try\_abort** (libsc\_t \* *ctx*)

Aborts the transaction using TSX or by triggering an exception.

**Parameters**

- **ctx** – The libsc context

int **libsc\_get\_hypercenthread** (libsc\_t \* *ctx*, int *logical\_core*)

Get the sibling hyperthread of the given logical core

**Parameters**

- **ctx** – The libsc context
- **logical\_core** – The logical core

**Returns** The id of the sibling hyperthread or -1

void **libsc\_pin\_to\_core** (libsc\_t \* *ctx*, int *pid*, int *core*)

Pins the pid to the given core

**Parameters**

- **ctx** – The libsc context
- **pid** – The pid to pin
- **core** – The core where pid should be pinned to

int **libsc\_get\_physical\_cores** (libsc\_t \* *ctx*)

Get the number of physical cores

**Parameters**

- **ctx** – The libsc context

**Returns** Number of physical cores

int **libsc\_get\_logical\_cores** (libsc\_t \* *ctx*)

Get the number of logical cores

**Parameters**

- **ctx** – The libsc context

**Returns** Number of logical cores

int **libsc\_get\_cache\_slice** (libsc\_t \* *ctx*, int *paddr*)

Returns the cache slice of the given physical address

**Parameters**

- **ctx** – The libsc context
- **paddr** – The physical address

**Returns** Cache slice of the physical address

int **libsc\_get\_cache\_set** (libsc\_t \* *ctx*, int *paddr*)

Returns the cache set of the given physical address

**Parameters**

- **ctx** – The libsc context
- **paddr** – The physical address

**Returns** Cache set of the physical address

int **libsc\_get\_physical\_address** ()

Returns the physical address of the given virtual address

**Parameters**

- **ctx** – The libsc context
- **vaddr** – The virtual address

**Returns** The corresponding physical address

int **libsc\_build\_eviction\_set** (libsc\_t \* *ctx*, libsc\_eviction\_set\_t \* *set*, int *paddr*)

Builds an eviction set for the physical address

#### Parameters

- **ctx** – The libsc context
- **set** – The built eviction set
- **paddr** – The physical address

**Returns** 0 on success

void **libsc\_evict** (libsc\_t \* *ctx*, libsc\_eviction\_set\_t *set*)

Runs eviction using the given eviction set

#### Parameters

- **ctx** – The libsc context
- **set** – The eviction set

int **libsc\_evict\_reload** (libsc\_t \* *ctx*, void \* *addr*, libsc\_eviction\_set\_t *set*)

Performs Evict+Reload :param *ctx*: The libsc context :param *addr*: The address :param *set*: The eviction set that should be used :return: 1 if *addr* was cached

void **libsc\_prime** (libsc\_t \* *ctx*, libsc\_eviction\_set\_t *set*)

Performs the prime step using the given eviction set :param *ctx*: The libsc context :param *set*: The eviction set that should be used

int **libsc\_prime\_probe** (libsc\_t \* *ctx*, libsc\_eviction\_set\_t *set*)

Performs Prime+Probe: Builds an eviction set for the given address if one does not exist :param *ctx*: The libsc context :param *set*: The eviction set :return: The executim time of the probe step

void \* **libsc\_map\_file\_by\_offset** (libsc\_t \* *ctx*, const char \* *filename*, int *offset*)

Maps a page of the given file at the defined offset to the programs address space and returns its address.

Note: This function leaks memory.

#### Parameters

- **ctx** – The libsc context The libsc instance
- **filename** – The path to the file
- **offset** – The offset that should be mounted

**Returns** Mapped address or NULL if any error occurs

void \* **libsc\_map\_file** (libsc\_t \* *ctx*, const char \* *filename*, int \* *filesize*)

Maps a the entire file to the programs address space and returns its address.

Note: This function leaks memory.

#### Parameters

- **ctx** – The libsc context The libsc instance
- **filename** – The path to the file
- **filesize** – Returns the size of the file (if not NULL)

**Returns** Mapped address or NULL if any error occurs

int **libsc\_find\_index\_of\_nth\_largest\_int** (libsc\_t \* *ctx*, int \* *list*, int *nmemb*, int *skip*)

Finds the index of the nth largest integer in the list

#### Parameters

- **ctx** – The libsc context
- **list** – The list
- **nmemb** – Number of list entries
- **skip** – The index to select

**Returns** The index

int **libsc\_find\_index\_of\_nth\_largest\_size\_t** (libsc\_t \* *ctx*, int \* *list*, int *nmemb*, int *skip*)  
Finds the index of the nth largest size\_t in the list

**Parameters**

- **ctx** – The libsc context
- **list** – The list
- **nmemb** – Number of list entries
- **skip** – The index to select

**Returns** The index

int **libsc\_wrmsr** (libsc\_t \* *ctx*, int *cpu*, uint32\_t *reg*, uint64\_t *val*)  
Writes to MSR

**Parameters**

- **ctx** – The libsc context
- **cpu** – The core id
- **reg** – The register
- **val** – The value

**Returns** 0 on success

int **libsc\_rdmsr** ()  
Reads from MSR

**Parameters**

- **ctx** – The libsc context
- **cpu** – The core id
- **reg** – The register

**Returns** The value of the register or 0

int **libsc\_measure\_slice** ()  
Calculates the slice id of the address (requires MSR access)

**Parameters**

- **ctx** – The libsc context
- **addr** – The addressess

**Returns** The slice id

void **libsc\_cache\_encode** (libsc\_t \* *ctx*, unsigned char *value*)  
Encodes a value into the cache

**Parameters**

- **ctx** – The libsc context

- **value** – The value to encode

unsigned char **libsc\_cache\_decode\_from\_to** (libsc\_t \* *ctx*, int *from*, int *to*)

Decodes the value in a given range from the cache

**Parameters**

- **ctx** – The libsc context
- **from** – Range begin
- **to** – Range end

**Returns** The decoded value

unsigned char **libsc\_cache\_decode** (libsc\_t \* *ctx*)

Decodes the value from the cache

**Parameters**

- **ctx** – The libsc context

**Returns** The decoded value

unsigned char **libsc\_cache\_decode\_nonnull** (libsc\_t \* *ctx*)

Decodes the value from the cache (no null version)

**Parameters**

- **ctx** – The libsc context

**Returns** The decoded value



## L

- `libsc_access` (*C function*), 4
- `libsc_access_b` (*C function*), 4
- `libsc_barrier_end` (*C function*), 4
- `libsc_barrier_start` (*C function*), 4
- `libsc_build_eviction_set` (*C function*), 6
- `libsc_cache_decode` (*C function*), 9
- `libsc_cache_decode_from_to` (*C function*), 9
- `libsc_cache_decode_nonull` (*C function*), 9
- `libsc_cache_encode` (*C function*), 8
- `libsc_calibrate_flush_reload` (*C function*), 5
- `libsc_cleanup` (*C function*), 3
- `libsc_evict` (*C function*), 7
- `libsc_evict_reload` (*C function*), 7
- `libsc_eviction_set_s` (*C type*), 3
- `libsc_eviction_strategy` (*C type*), 3
- `libsc_find_index_of_nth_largest_int` (*C function*), 7
- `libsc_find_index_of_nth_largest_size_t` (*C function*), 8
- `libsc_flush` (*C function*), 3
- `libsc_flush_b` (*C function*), 4
- `libsc_flush_reload` (*C function*), 5
- `libsc_get_cache_set` (*C function*), 6
- `libsc_get_cache_slice` (*C function*), 6
- `libsc_get_hyperthread` (*C function*), 5
- `libsc_get_logical_cores` (*C function*), 6
- `libsc_get_physical_address` (*C function*), 6
- `libsc_get_physical_cores` (*C function*), 6
- `libsc_init` (*C function*), 3
- `libsc_map_file` (*C function*), 7
- `libsc_map_file_by_offset` (*C function*), 7
- `libsc_measure_end` (*C function*), 4
- `libsc_measure_slice` (*C function*), 8
- `libsc_measure_start` (*C function*), 4
- `libsc_pin_to_core` (*C function*), 6
- `libsc_prime` (*C function*), 7
- `libsc_prime_probe` (*C function*), 7
- `libsc_rdmsr` (*C function*), 8
- `libsc_s` (*C type*), 3
- `libsc_s.llc_hit_threshold` (*C member*), 3
- `libsc_s.llc_miss_threshold` (*C member*), 3
- `libsc_s.timestamp` (*C member*), 3
- `libsc_set_timer` (*C function*), 5
- `libsc_speculation_barrier` (*C function*), 5
- `LIBSC_TIMER_COUNTING_THREAD` (*C macro*), 3
- `LIBSC_TIMER_MONOTONIC_CLOCK` (*C macro*), 3
- `LIBSC_TIMER_NATIVE` (*C macro*), 3
- `LIBSC_TIMER_PERF` (*C macro*), 3
- `libsc_timer_t` (*C type*), 3
- `libsc_timestamp` (*C function*), 4
- `libsc_timestamp_function_t` (*C type*), 3
- `libsc_try_abort` (*C function*), 5
- `libsc_try_end` (*C function*), 5
- `libsc_try_start` (*C function*), 5
- `libsc_wrmsr` (*C function*), 8