MoLE: Mitigation of Side-channel Attacks against SGX via Dynamic Data Location Escape

Fan Lang[1,2], Wei Wang[1], Lingjia Meng[1,2], Jingqiang Lin[3], Qiongxiao Wang[1], Linli Lu[1]

[1] State Ley Laboratory of Information Security, Institute of Information Engineering, China Academy of Sciences, Beijing 100089, China

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100089, China

[3] School of Cyber Security, University of Sciences and Technology of China, Hefei 230027, Anhui, China

Abstract

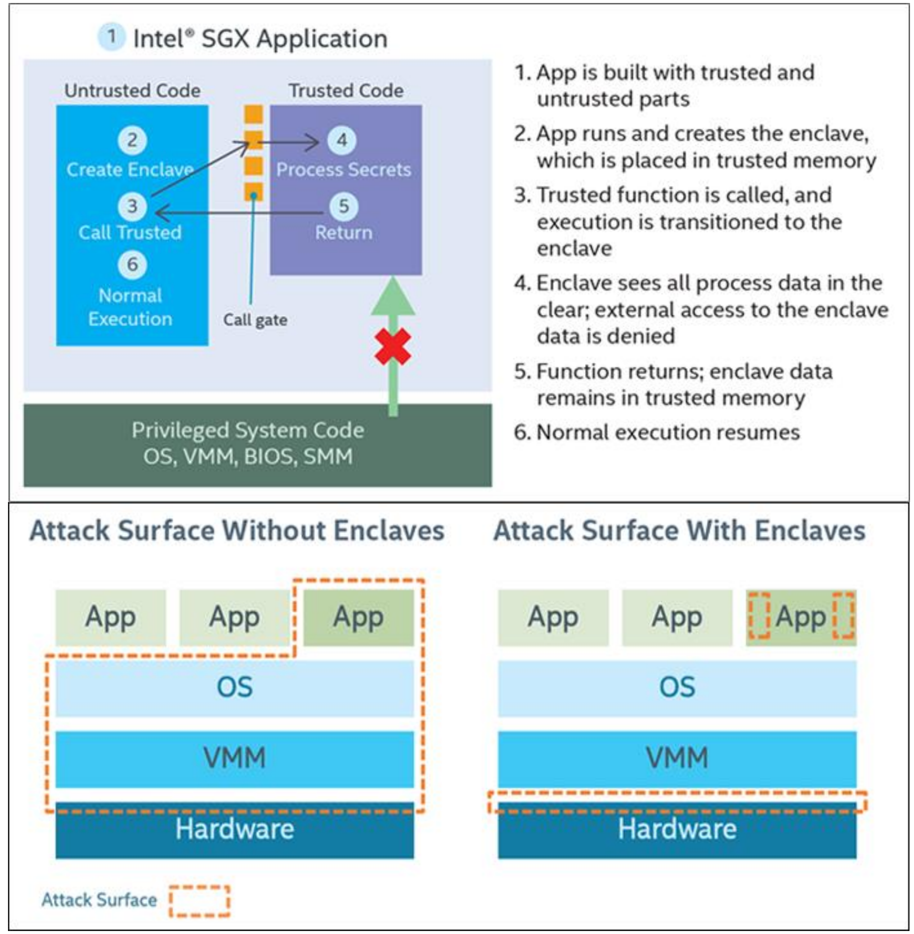
Software Guard eXtension(SGX) is a set of instructions and mechanisms for memory accesses added to Intel architecture processors, which aim to provide integrity and confidentiality assurance.

Unfortunately, the security mechanism still has weakness, such as side-channel attacks(SCAs).

Therefore, we propose MoLE, a dynamic data location randomization scheme to defend against SCAs and transient execution attacks that target sensitive data within enclaves. By continuously obfuscating the location of sensitive data at runtime, MoLE prevents the adversary from directly obtaining or disclosing data based on data access patterns.

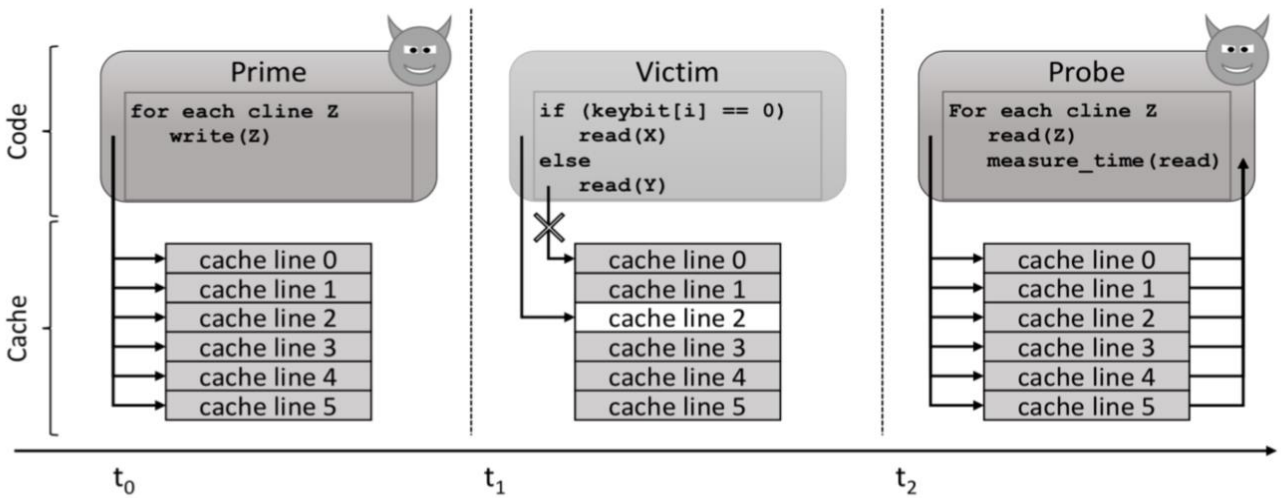
SGX（Software Guard Extensions）

- SGX can provide applications trusted execution environments, called enclave
  - Isolated from the OS by the hardware processor
  - Encrypted using processor-specific keys
  - Ensure the confidentiality and integrity of data and code in enclave
- Vulnerable to SCA



Cache SCA

- Characteristics of cache side-channel attacks
  - No need to access the code or data in the isolated execution environment
  - By analyzing the cache side-channel information to indirectly obtain the key information
  - Such as, Prime + Probe

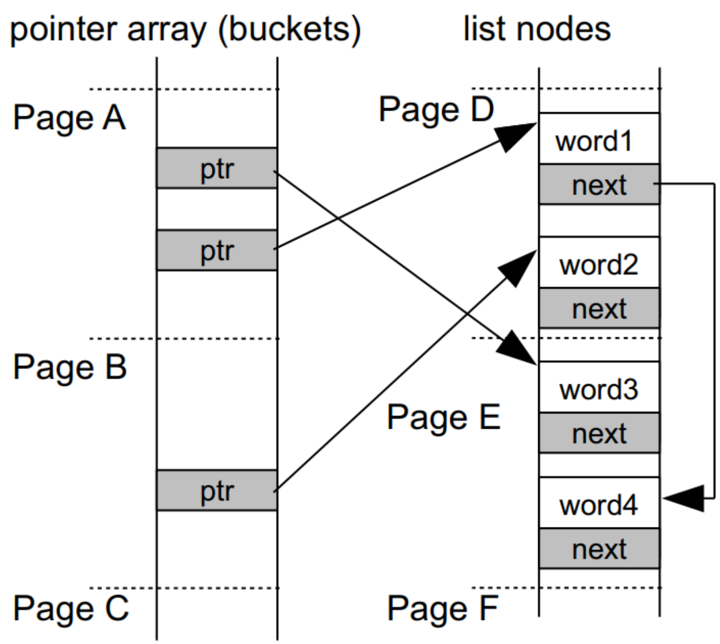


Page Table SCA

- Characteristics of page table side-channel attacks
  - No need to access the target pages
  - By analyzing the page table side-channel information to indirectly obtain the key information
  - Such as,

    Controlled-

    Channel

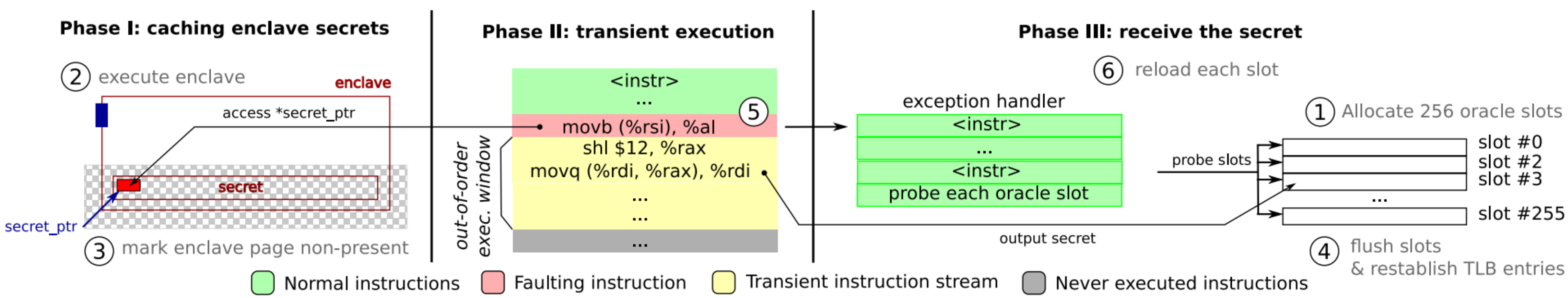


Pages associated with each word

Word	Pages
word1	A, D
word2	B, D
word3	A, E
word4	B, D, F

Meltdown-type Attack

- Characteristics of page table side-channel attacks
  - Getting unauthorized access to the data with transient execution driven by fault or assist
  - Combining Flush + Reload Cache SCA
  - Such as, Foreshadow



Solution——MoLE

- Basic idea
  - Constantly changing location of sensitive data at runtime to obfuscate the data access patterns
- Challenge
  - How to achieve untraceable data obfuscation against privileged attackers with single-step capability?
  - How to design a defense that is generalizable and relatively transparent to the target application?

Technology

- TSX（Transactional Synchronization Extension）
  - TSX is Intel’s implementation of hardware transactional memory (HTM)
    - Operations within a transaction are atomic
    - An interruption or exception causes a rollback
    - XBEGIN, XEND, XABORT, and XTEST
  - Support for multi-threading
  - Resistant to single-step behavior

Threat Model

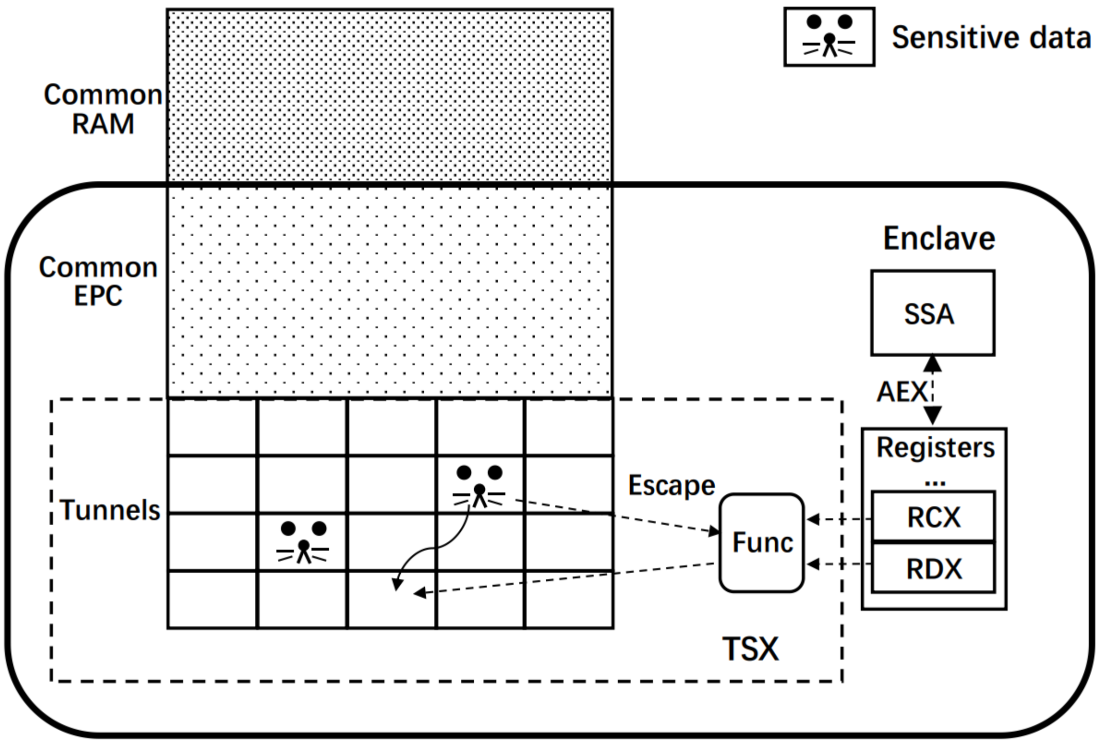
- Attackers have full control over the OS
  - Generate interruptions to single-step the target application
  - Block, replay, read, and modify all message outside enclaves

- Attackers have the ability to launch the following attacks
  - Cache SCAs
  - Page table SCAs
  - Meltdown-type transient execution attacks

- Do not consider LLC SCAs

MoLE

- MoLE defeats attacks by altering the sensitive data's location dynamicly at runtime, which we call dynamic data location *Escape*.

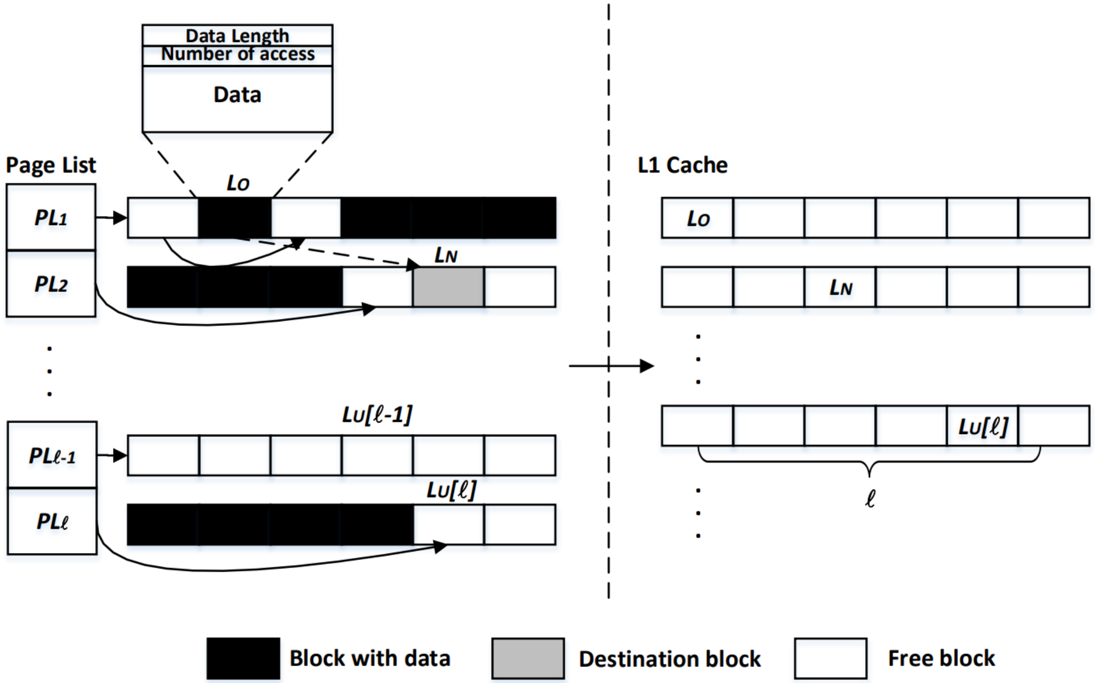


- *Tunnels*
- *Escape* function

- Embraced in a single transaction

Tunnels

- A designated portion of the enclave heap to store the sensitive data



- A page list
- Several doubly-linked lists

- The *Tunnels* has *l* (Each L1d cache set consists of *l* cache lines) items in the page list

Escape Function

- Computing new addresses for data *Escape*

Algorithm 1 The process of *Escape*.

Input: The page list of MoLE PL, PL has l items; The last location of data $L_O \in PL[i]$, $0 \leq i \leq l$ and the size of data $Size_{Data}$; The random numbers stored in RCX and RDX, R_C, R_D;

Output: The new location of data, L_N; The location of unrelated data, L_U;

$L_B = [\ L_O \oplus R_C]_{NearBlock}$; L_B is the nearest free block to the result;

$L_N = L_B + R_D \bmod (Size_B - Size_{Data})$, $L_N \in PL[i]$, $0 \leq j \leq l$; $Size_B$ is the size of L_B;

for $k = 0 \rightarrow l - 1$ **do**

$\quad L_U[k] = PL[k] + (\ L_N - PL[j] + k * Size_{Data}) \bmod (Size_{PL[j]} - Size_{Data}), k \neq i, k \neq j$;

end for

- Feint accesses

Implementation

- Runtime Library

Interface	Purpose
int MoLEInit()	(ECALL) Allocate memory for the *Tunnels* and organize it RET: the *Tunnels* status
void* MoLEMalloc(uint64_t ptr_size)	Allocate buffer from the *Tunnels* RET: allocation pointer, ptr_size: allocation size
uint64 MoLEMallocSize(void* ptr)	Get allocation buffer size RET: buffer size, ptr: buffer pointer
void MoLEscape(void* ptr)	If the buffer access reaches the access threshold, data *Escapes* ptr: buffer pointer
void* MoLERealloc(void* ptr, uint64_t ptr_size)	Expand buffer size RET: new buffer pointer, ptr: original pointer, ptr_size: new buffer size
void MoLEFree(void* ptr)	Release buffer to the *Tunnels* ptr: buffer pointer

- Encapsulating the operations for the *Tunnels* and the *Tunnels* data structure

Implementation

- LLVM Pass

```
%4 = alloca [64 x i8], align 4
%5 = tail call noalias i8* @malloc(i8 1024)
...
store i8* %5, i8** %23, align 8
%24 = bitcast [64 x i8]* %4 to i8*
...
call void @free(i8* %4 )
call void @free(i8* %6 )
```

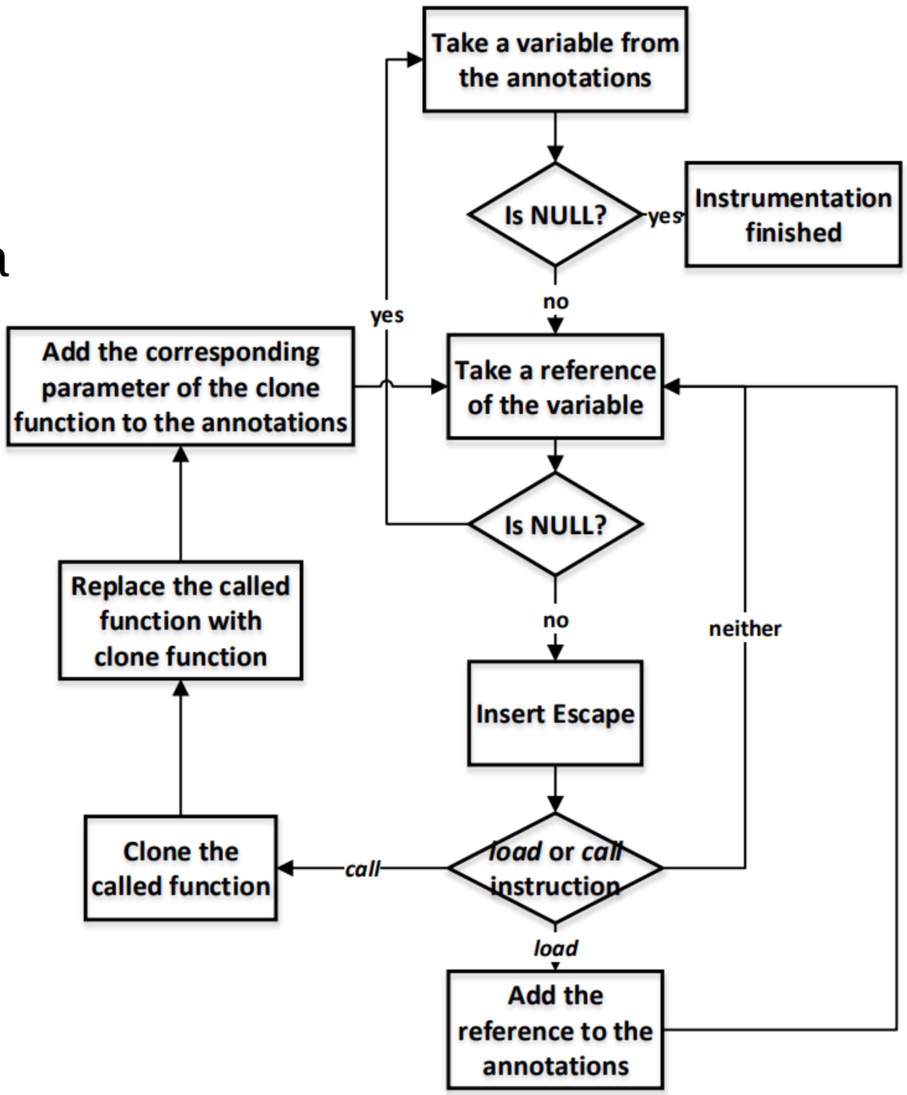
MoLE PASS

```
%4 = tail call noalias i8* @MoLEMalloc(i8 64)
%5 = tail call noalias i8* @MoLEMalloc(i8 1024)
...
%23 = getelementptr inbounds i8*, i8** %5, i32 0
%24 = bitcast i8** %23 to i8*call void @MoLEEscape(i8* %24)
store i8* %5, i8** %25, align 8
%26 = getelementptr inbounds i8*, i8** %4, i32 0
%27 = bitcast i8** %26 to i8*call void @MoLEEscape(i8* %27)
%28 = bitcast [64 x i8]* %4 to i8*
...
call void @MoLEFree(i8* %4 )
call void @MoLEFree(i8* %6 )
```

- Instrumenting the application enclave code about the annotation variables at the intermediate representation (IR) level

Implementation

- Taint Tracking
  - There may be taint propagation when accessing sensitive data by some instructions
    - *load*
    - *call*

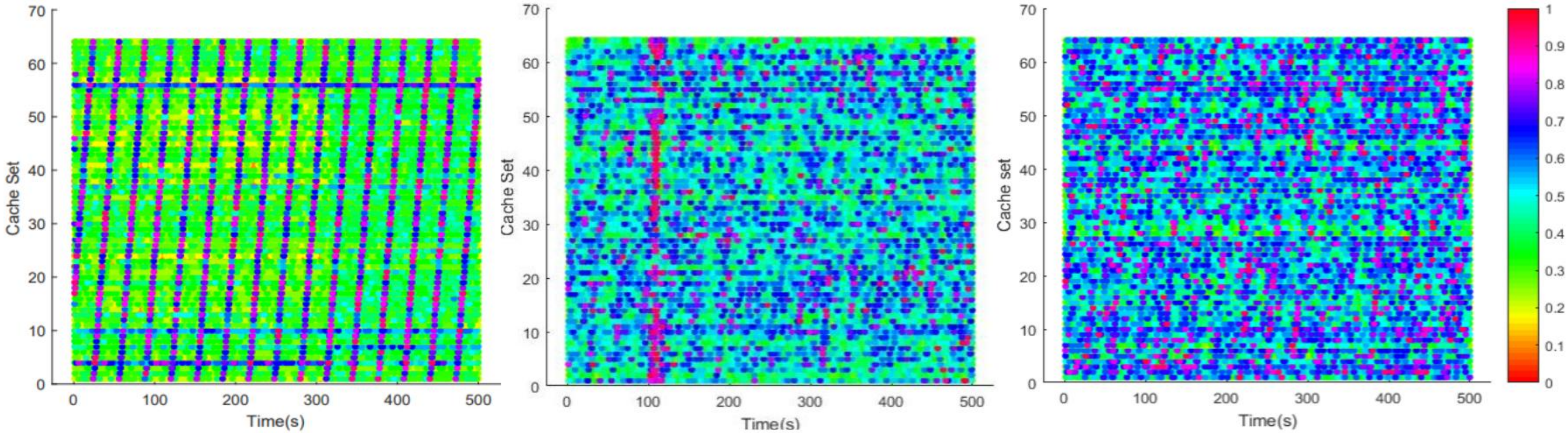


Analysis

- Cache SCA
  - Prime + Probe
    - *Escape* evicts the preloaded data of various cache lines
- Page table SCA
  - *Escape* would allow sensitive data to be located on any page in the *Tunnels*
- Meltdown-type attack
  - *Escape* make it difficult to get the exact location of the data
- Single-step tracking
  - TSX will roll operations within a transaction back to invalidate single-step tracking

Evaluation

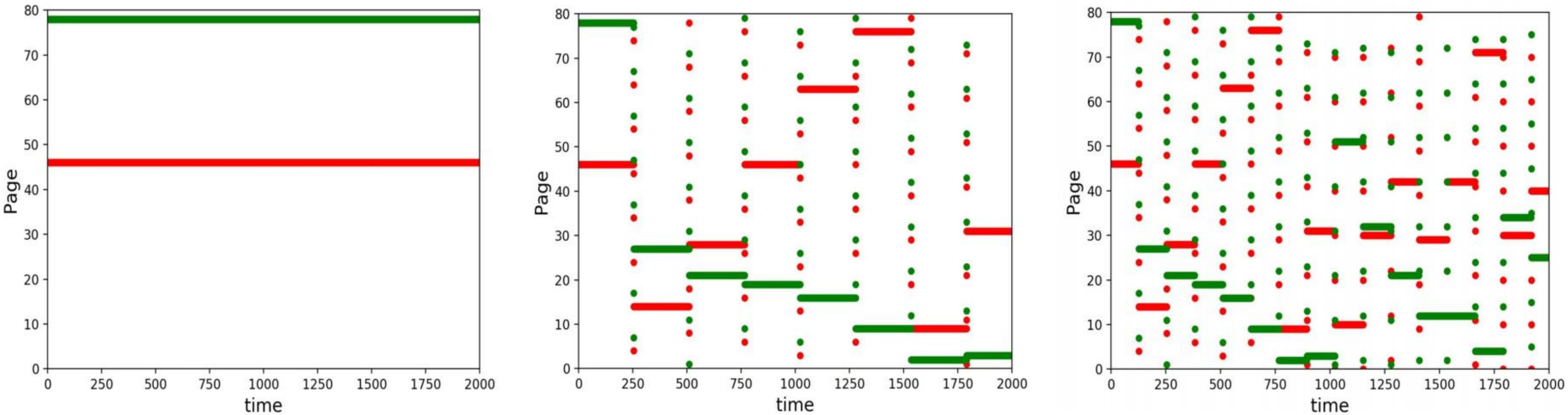
- Cache SCA
  - As the access threshold T decreases, the cache access pattern becomes chaotic



(a) Without *Escape* (b) The threshold of *Escape* $T = 256$ (c) The threshold of *Escape* $T = 128$

Evaluation

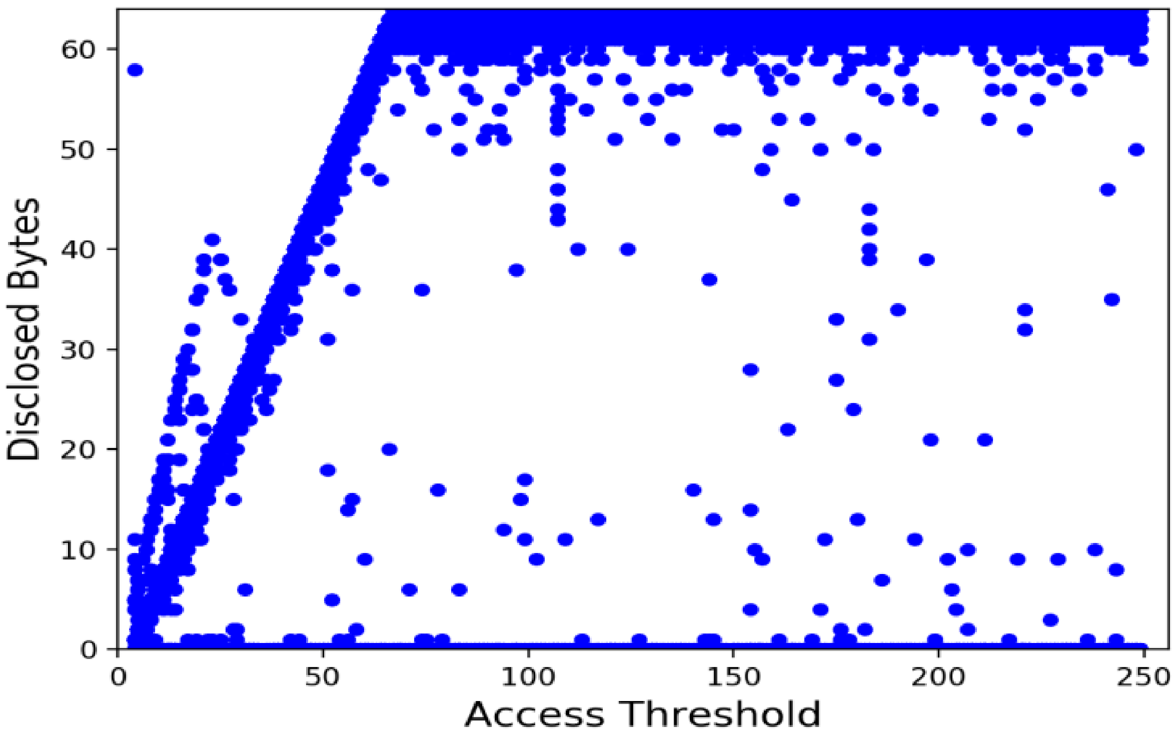
- Page table SCA
  - With the *Escape*, the patterns of these two variables become confusing



(a) Without *Escape* (b) The threshold of *Escape* $T = 256$ (c) The threshold of *Escape* $T = 128$

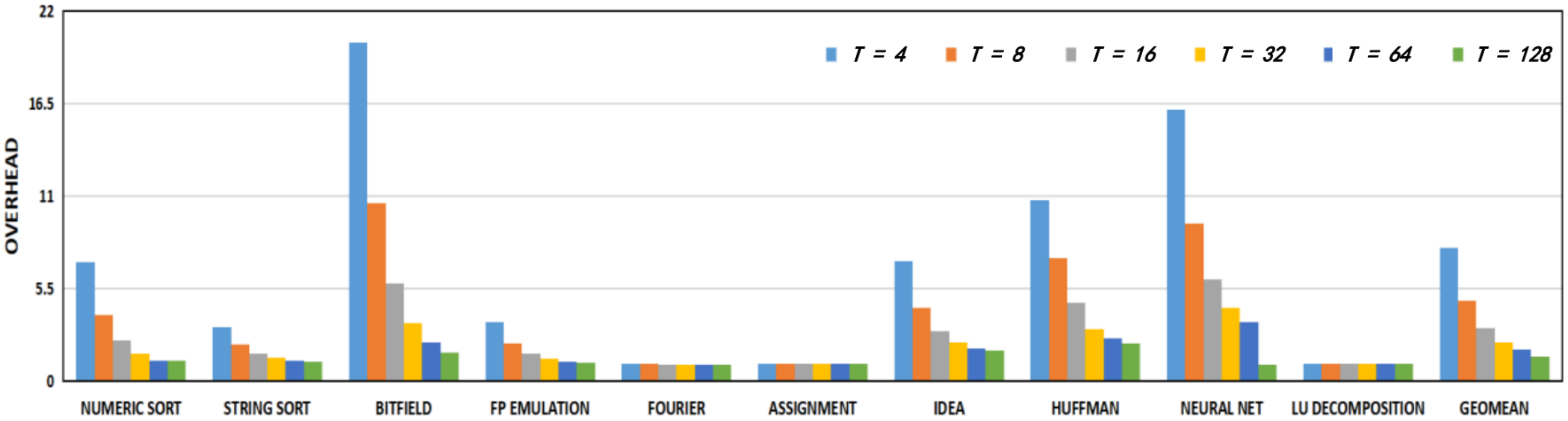
Evaluation

- Meltdown-type attack
  - Foreshadow
    - The amount of stolen data decreases as the threshold T decrease



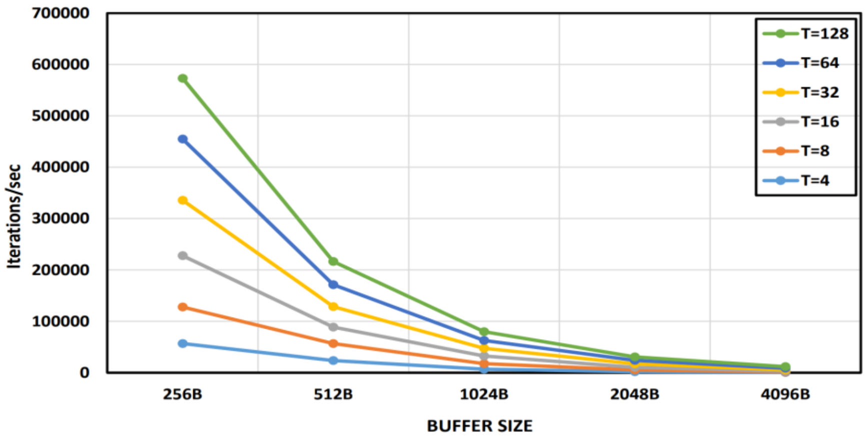
Evaluation

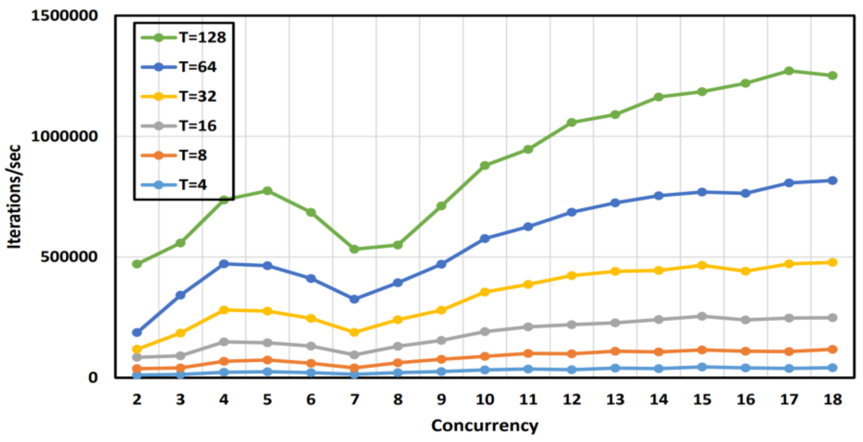
- Nbench
  - When $T \in \{4,8,16,32,64,128\}$, the overhead is 7.89×, 4.78×, 3.14×, 2.29×, 1.89x, and 1.48×.



Evaluation

- Nbench
  - Data size
  - Multi-threading





Related Work

- Limiting concurrency
  - Usage restrictions (multi-threading)
- Obfuscation
  - Weak for single-step tracking

Defensive concept	Scheme	Protection scope			Muti-threading	Single-step tracking
		Cache SCA	Page table SCA	Meltdown-type		
Limit interruption or concurrency	*Déjà Vu* [14]	✔	✔	✗	✔	✔
	Racing [13]	✔	✗	✗	*restricted*	✔
	VARYS [32]	✔	✗	✗	*restricted*	✔
Isolate or obfuscate shared resources	*T-SGX* [39]	✗	✔	✗	✔	✔
	Cloak [19]	✔	✗	✗	✔	✔
	KLOTSKI [48]	✗	✔	✗	–	✗
	OBFSCURO [5]	✔	✔	✗	–	✗
	DR.SGX [6]	✔	✔	✔	✗	✗
	MoLE	✔	✔	✔	✔	✔

- Limited defense range

Thank you!