Data-Oblivious External Shuffling on SGX

Hao Zhou

Under Supervision of Dr. ***Yuzhe Tang***

Who Am I

- Hao Zhou
- Junior
- Computer Sciencer major

Goal: **Secure** Data **Analysis** on **Emerging TEE**

1. **Emerging TEE**: Intel SGX
    - Intel® Software Guard Extensions (Intel® SGX) is an Intel technology for application developers who are seeking to protect select code and data from disclosure or modification.

Goal: **Secure** Data **Analysis** on **Emerging TEE**

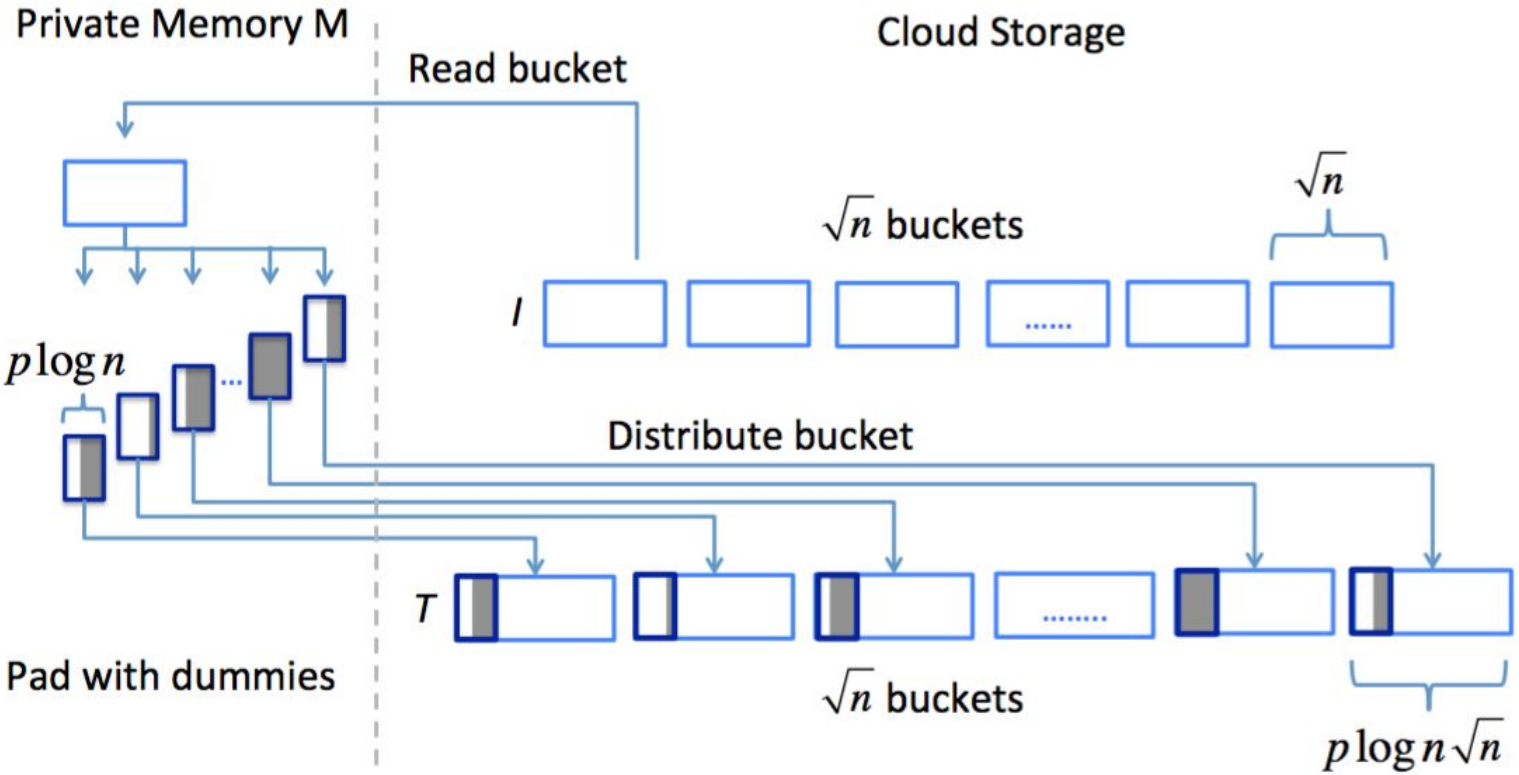
1. **Security**: Data-Obliviousness
   - Meaning: Data access trace independent of data values
   - Example: Bubble sort versus Merge sort
   - Time complexity: $O(n_2)$ versus $O(n \log n)$
   - Tradeoff: Obliviousness and performance
   - Our goal: We want both.
2. **Data Analysis**: Data shuffling
   - Meaning: Permutation multiplication
   - Data shuffling is a fundamental operator of oblivious data analysis

Goal: **Secure** Data **Analysis** on **Emerging TEE**

Focus of this work:

*Implementing **Melbourn shuffle** on **Intel SGX***

Melbourne Shuffle [1]



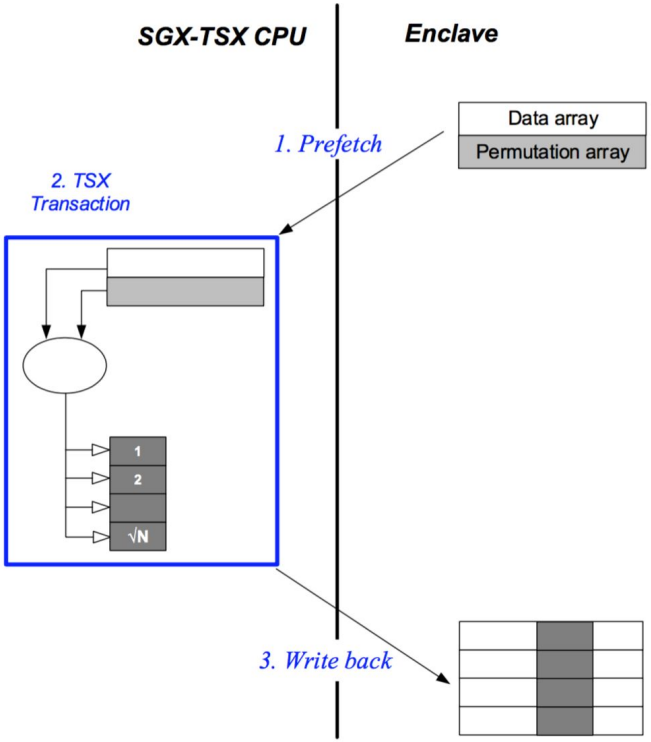
Implementing Melbourne Shuffling with SGX-TSX

- TSX (Intel Transactional Synchronization eXtensions)
  - Two new instructions “XBEGIN”, “XEND”
  - All the code between “XBEGIN” and “XEND” is in a TSX ***transaction***
  - When cache miss happens inside a TSX transaction, the transaction abort

Implementing Melbourne Shuffling with SGX-TSX

*My work: Implemented the Melbourne shuffle in **X86_64** and **C (~400 LoC)***

Demo: Functioning on real Intel SGX machine in our lab.




```
.globl  asm_compute_CPU
.type   asm_compute_CPU, @function
asm_compute_CPU:
  pushq %rbp
  movq  %rsp, %rbp
  movl  $0, -4(%rsp) #index pointer for slot 3
  movl  $0, -8(%rsp) #index pointer for slot 2
  movl  $0, -12(%rsp) #index pointer for slot 1
  movl  $0, -16(%rsp) #index pointer for slot 0
  movl $0,-16(%rsp)
  movl $8,-12(%rsp)
  movl $16,-8(%rsp)
  movl $24,-4(%rsp)
  #preload start
  movq E_data@GOTPCREL(%rip), %rax
  mov (%rax),%rcx
  mov 4(%rax),%rcx
  mov 8(%rax),%rcx
  mov 12(%rax),%rcx
  movq E_perm@GOTPCREL(%rip), %rax
  mov (%rax),%rcx
  mov 4(%rax),%rcx
  mov 8(%rax),%rcx
  mov 12(%rax),%rcx
  movq E_output@GOTPCREL(%rip), %rax
  mov (%rax),%rcx
  mov 4(%rax),%rcx
  mov 8(%rax),%rcx
  mov 12(%rax),%rcx
  movq E_output@GOTPCREL(%rip), %rcx
  movq E_data@GOTPCREL(%rip), %rax
  movq E_perm@GOTPCREL(%rip), %rbx
  sub $16,%rsp
  #preload end
```

```
xbegin ll
  movl (%rax),%esi  #esi = data[0]
  movl (%rbx),%edi  #edx = perm[0]
  sarl $2,%edi      #edx=edx/4
  movl (%rsp,%rdi,4),%edx #pointer
  movl %esi,(%rcx,%rdx,4)
  addl $1,%edx  #pointer increase
  movl %edx,(%rsp,%rdi,4)

  movl 4(%rax),%esi  #esi = data[1]
  movl 4(%rbx),%edi  #edx = perm[1]
  sarl $2,%edi       #edx=edx/4
  movl (%rsp,%rdi,4),%edx #pointer
  movl %esi,(%rcx,%rdx,4)
  addl $1,%edx  #pointer increase
  movl %edx,(%rsp,%rdi,4)

  movl 8(%rax),%esi  #esi = data[2]
  movl 8(%rbx),%edi  #edx = perm[2]
  sarl $2,%edi       #edx=edx/4
  movl (%rsp,%rdi,4),%edx #pointer
  movl %esi,(%rcx,%rdx,4)
  addl $1,%edx  #pointer increase
  movl %edx,(%rsp,%rdi,4)

  movl 12(%rax),%esi  #esi = data[3]
  movl 12(%rbx),%edi  #edx = perm[3]
  sarl $2,%edi       #edx=edx/4
  movl (%rsp,%rdi,4),%edx #pointer
  movl %esi,(%rcx,%rdx,4)
  addl $1,%edx  #pointer increase
  movl %edx,(%rsp,%rdi,4)
  xend

  mov $0, %rax
  add $16,%rsp
  popq %rbp
  ret
ll:
  mov $777,%rax
  add $16,%rsp
  popq %rbp
  ret
```

Reference

- *The Melbourne Shuffle: Improving Oblivious Storage in the Cloud. ICALP (2) 2014: 556-567, Olga Ohrimenko, Michael T. Goodrich, Roberto Tamassia, Eli Upfal*

Thank You

- Acknowledgement: *FSS* research group
  - Professor: Dr. Yuzhe (Richard) Tang
  - PhD: Ju Chen, Amin Fallahi

- Q/A?