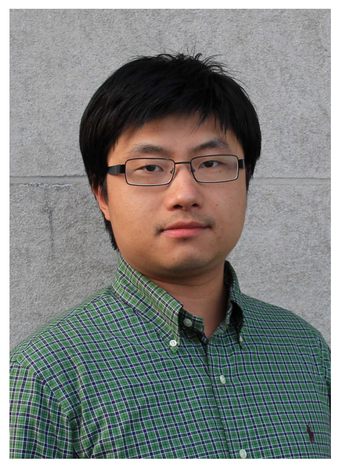
Scalable Secure Computations on Intel SGX via Lazy Program Partitioning



Yuzhe Tang, Syracuse University

https://exo-sgx.github.io/

Introduction

•Intel Software Guard eXtension (SGX) is ISA extension for security, released in recent Intel Skylake CPU: Encrypted memory, Data-path access control, Security-isolated exec. environment.

•Intel SGX, despite its security on other aspects, is known to suffer memory-access side-channel attacks (SP'15)

•This work is about defending such attacks:

- Security goal: Defending all such attacks at both software and hardware levels.
- Performance goal: with minimal/ideally small overhead.

Goal: Cache-miss Obliviousness

- Cache miss obliviousness:
  - Data obliviousness is a property that the trace of memory access during a program execution is independent to the value of data accessed.
  - Cache miss obliviousness:
    - Cache-miss sequence is data oblivious (for security).
    - Cache-hit sequence is non-oblivious (for performance).

Technique 1: Mapping Melbourne Shuffle

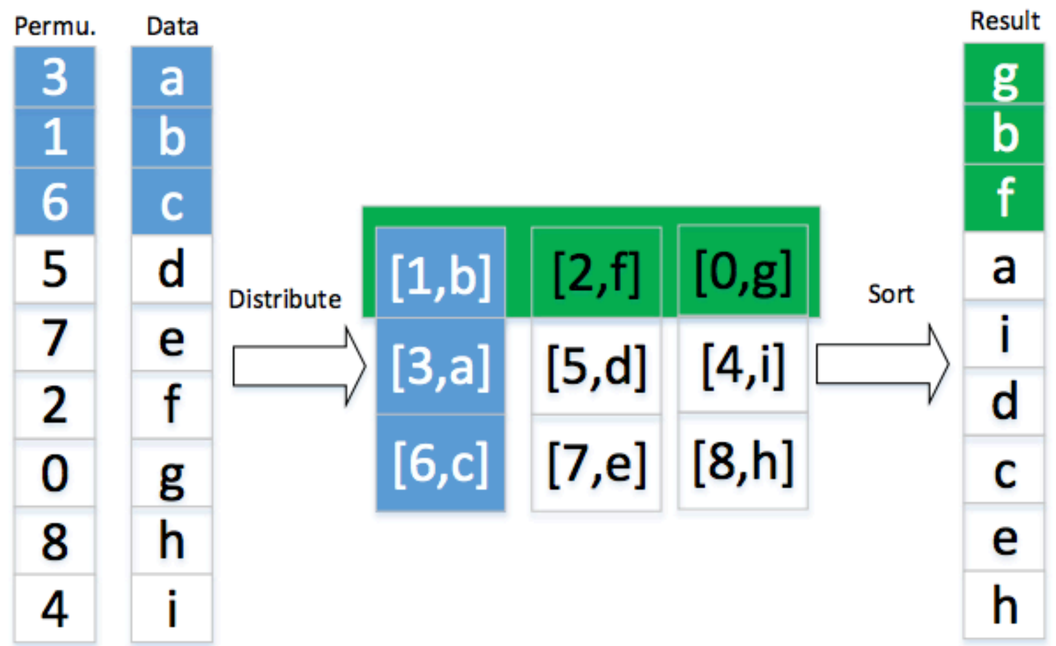
- Two data passes.
- Each pass obliviously access the external storage.
- At the same time, each pass obliviously access the internal storage.
- Mapping Melbourne shuffle to SGX
  - Isolate leaky internal storage to cache
  - Map the oblivious external storage safely to untrusted memory.

Our Approach

•Key idea: Combining isolation with obliviousness.

•This work focuses on data shuffling which is a fundamental operation in many computations.

•Engineering Melbourne Shuffle on SGX with resilience of memory-access attacks.



•Technique 2: Isolate Cached data by Intel TSX

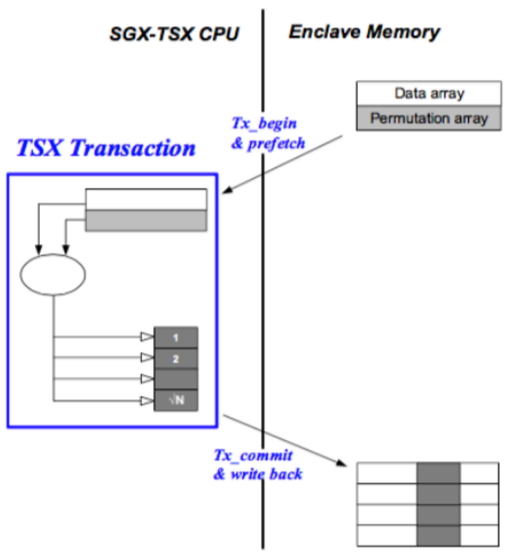
Useful TSX capability: Abort transaction upon cache write-back

•Proposed idea for cache isolation:

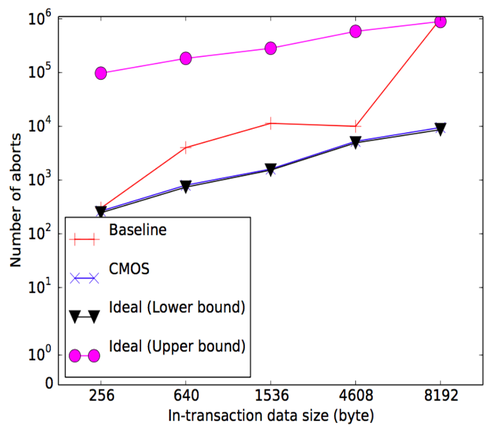
- Equating cache misses with cache write-back (making all cache lines dirty in the transaction)

•Technique: Data preloading

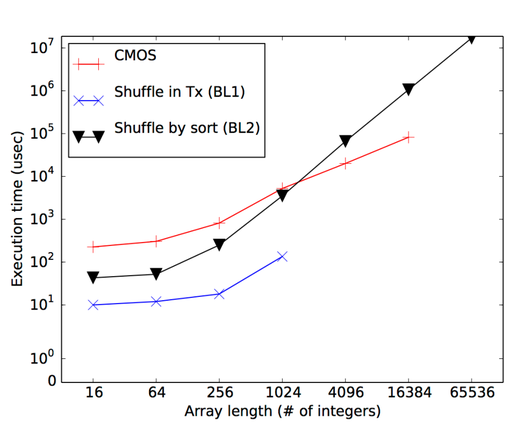
- Assurance of isolation
- Low transaction abort rate



Evaluation Results



Abort Rate: Our approach is close to ideal lower bound



Perf.: Our approach is faster than shuffling by sort
Scalability: Our approach supports larger dataset than naively shuffling in transaction