SGX-MR

Regulating Dataflows for Protecting Access Patterns of Data-Intensive SGX Applications

AKM Mubashwir Alam [1] **Sagar Sharma** [2] **Keke Chen** [1]

[1] Trustworthy and Intelligent Computing Lab (TIC), Northwestern Mutual Data Science Institute, Department of Computer Science, Marquette University

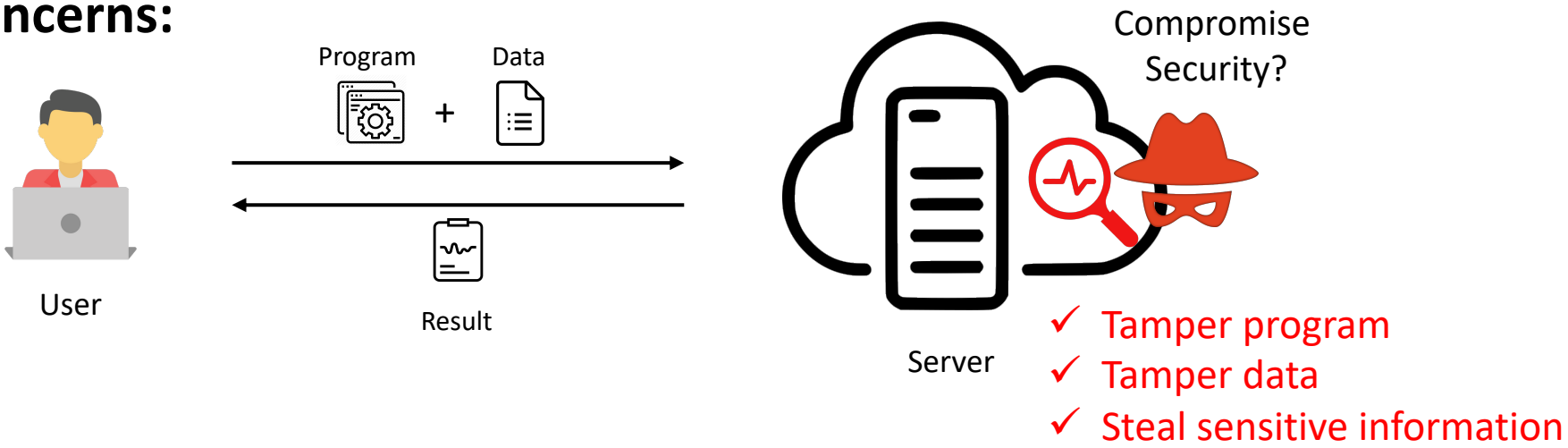
[2] HP Inc.

Cloud Computing and Security Concerns

Benefits of cloud computing:

- **Economics:** pay-as-you-use billing model
- **Scalability**: plenty of resources for storage and computing
- **Accessibility**: Easy access regardless of time and location

Security concerns:



Goal

- **Confidentiality**: Server learns nothing
- **Integrity**: Server returns accurate result
- **Efficiency**: Faster execution time

Existing solutions

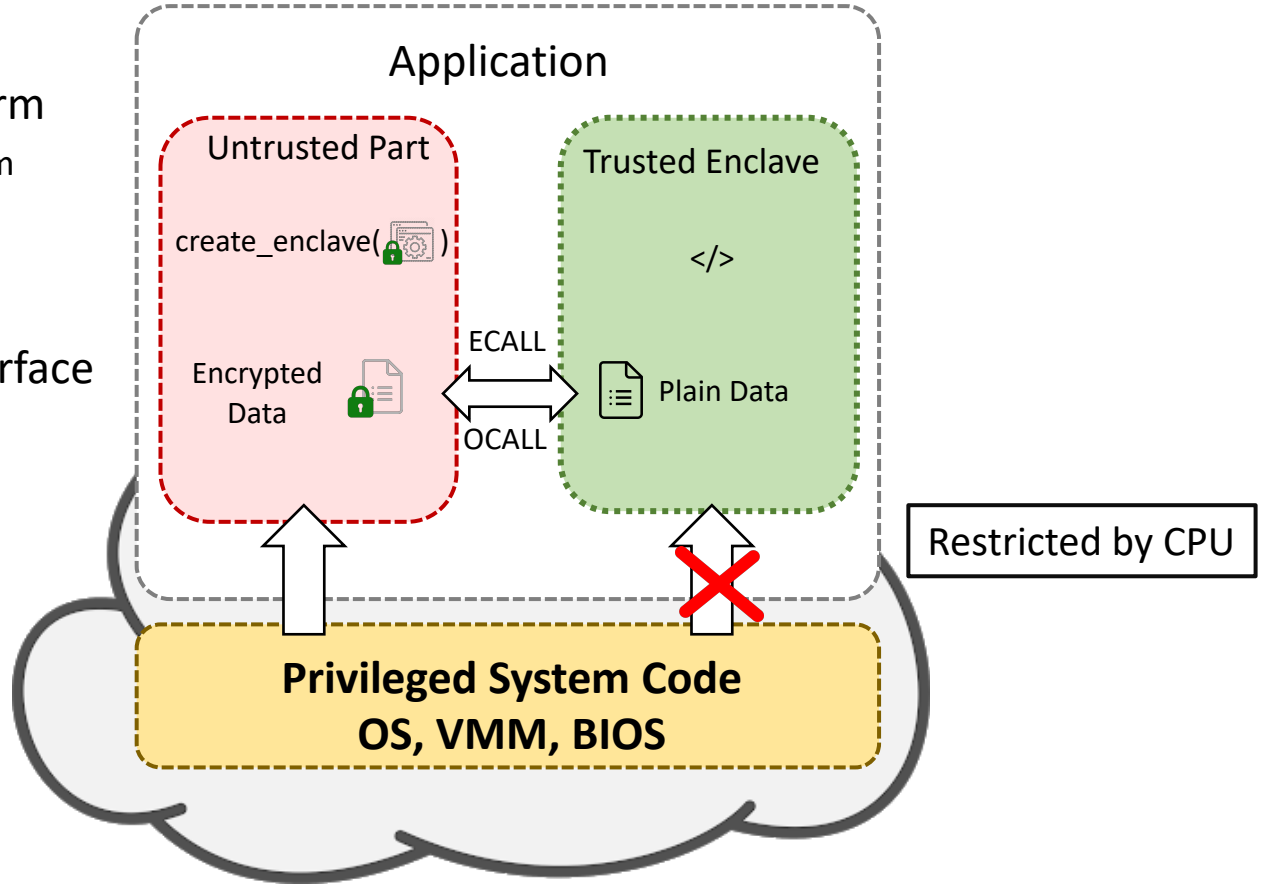
- Software-based crypto approaches
  - Fully Homomorphic Encryption (FHE)
  - Secure Multi-party Computation (SMC)

- Trusted Execution Environment (TEE)
  - Confidentiality
  - Integrity
  - Efficiency

Intel SGX as Trusted Execution Environment

Example Procedure

- Gain trust via Remote Attestation
- Deliver sensitive data in Encrypted form
  - Signed Binary executes as Enclave Program
  - User data
- Create Enclave with signed binary
- Call trusted functions via Enclave Interface
  - ECALL
  - OCALL



Privacy Issues of SGX

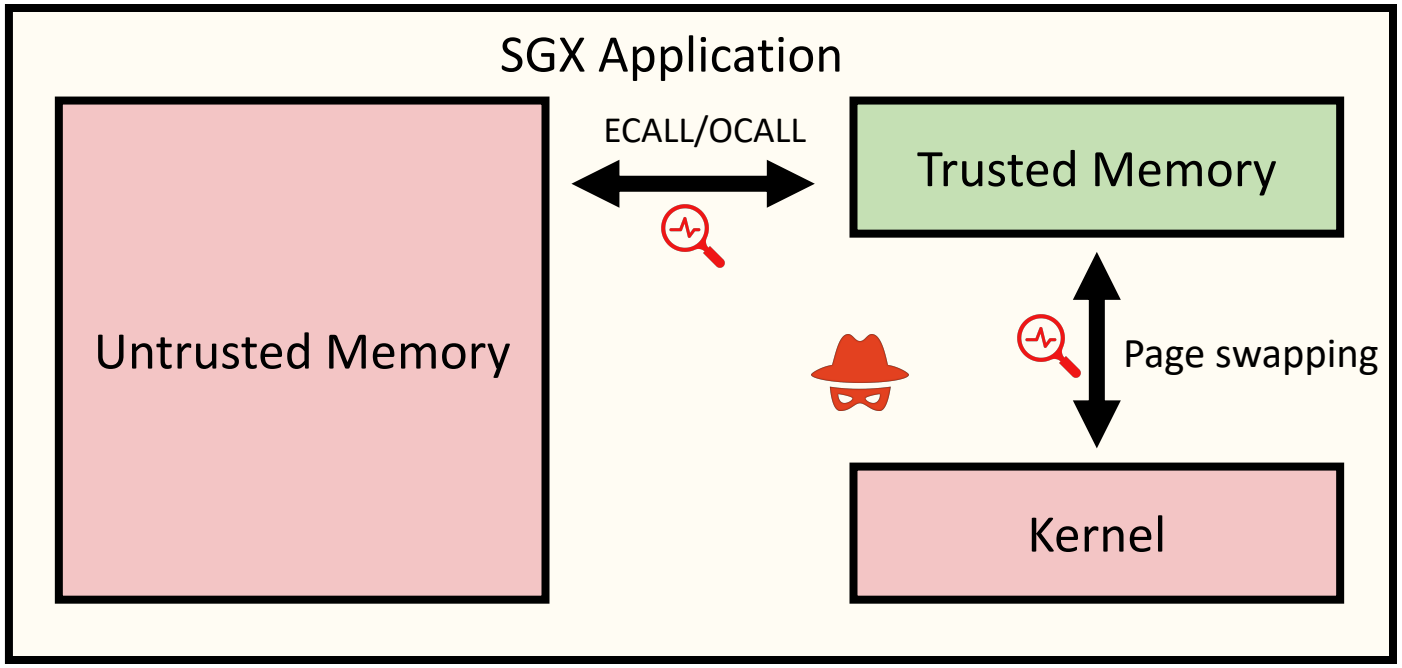
- Intel SGX vulnerable from software attacks.
  - Page Fault Attack
  - Data/Page Access Pattern Attacks
  - Branch Shadowing Attack

- Other side Channels (Out of our scope)
  - Speculative Execution Attack
  - Injection based Attack

✓ Software based mitigations are not sufficient
✓ Require Microarchitectural level patch

Traceable Memory Access of SGX Application

- OS observes memory interaction
  - access untrusted memory via ECALL: - data/block access pattern
  - Kernel dependency for Enclave page swapping - page fault attacks



Existing Approach to Access Pattern Protection

Oblivious RAM

- ORAM Controller in Enclave
- Path ORAM and Circuit ORAM algorithms

- Limitations
  - Still not efficient: O(log n) operations per access
  - Only protects a set of data access
  - Cannot protect application specific code access, e.g., conditional branching

Towards Data Access Pattern Protection

Our Observation

- ORAM incurs high computation cost
- Regulating data-flow can be more efficient than ORAM primitive
- Some Application Specific Access Pattern does not leak additional information
  - E.g., Sequential Block Access
- Other access patterns can be replaced with specific Oblivious Algorithms
  - E.g., Oblivious Sorting, Oblivious Merge, Oblivious Swap, etc.

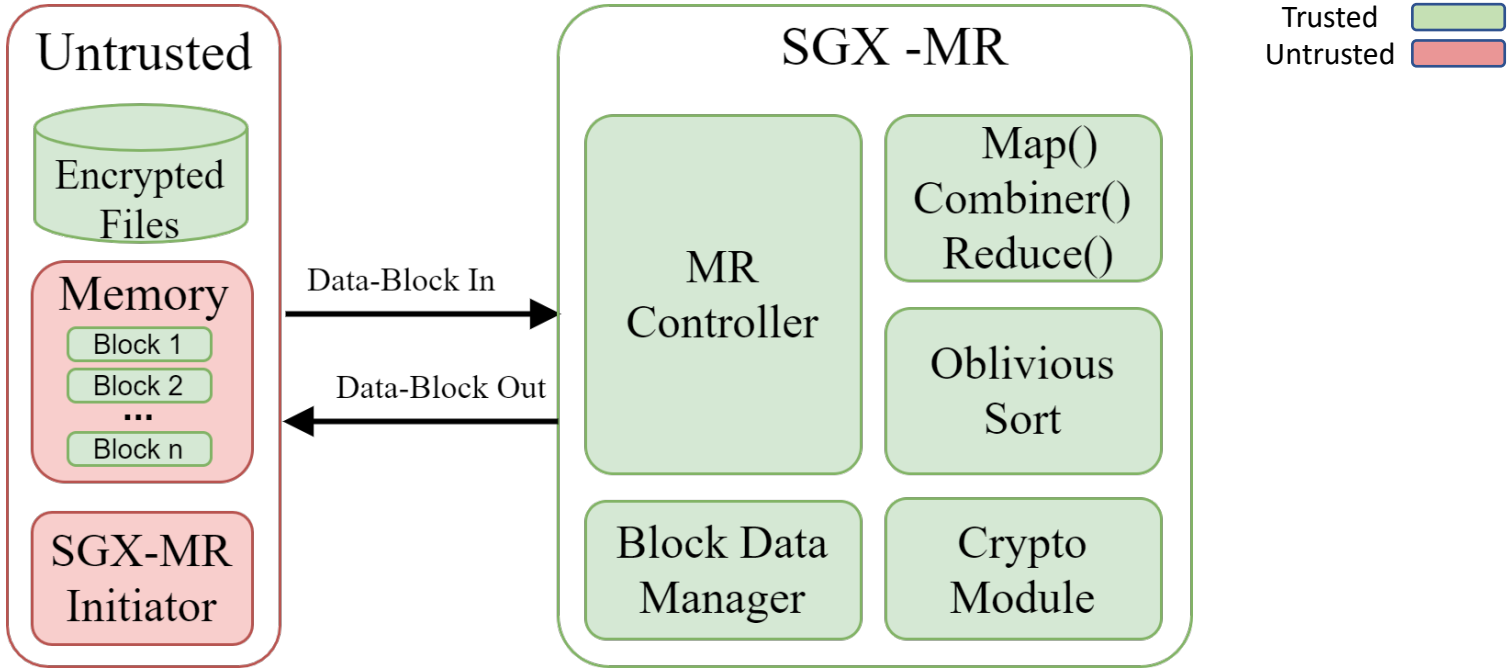
- Challenges
  - Need detail analysis for each application specific access patterns
    - Time Consuming
    - Error Prone

SGX-MR: Our Contribution

- Regulating the dataflow with MapReduce
- Designed robust mitigation methods to prevent
  - Untrusted memory access pattern leakages
  - In-Enclave memory access pattern leakages
- Implemented the lightweight SGX-MR framework
  - Flexible to adapt to the enclave's restriction. E.g., limited memory
- Conducted extensive component-wise experiments
  - Understand the cost and performance
  - Compare with ORAM based SGX approach.
  - Result: SGX-MR can be several times faster than ORAM based solutions

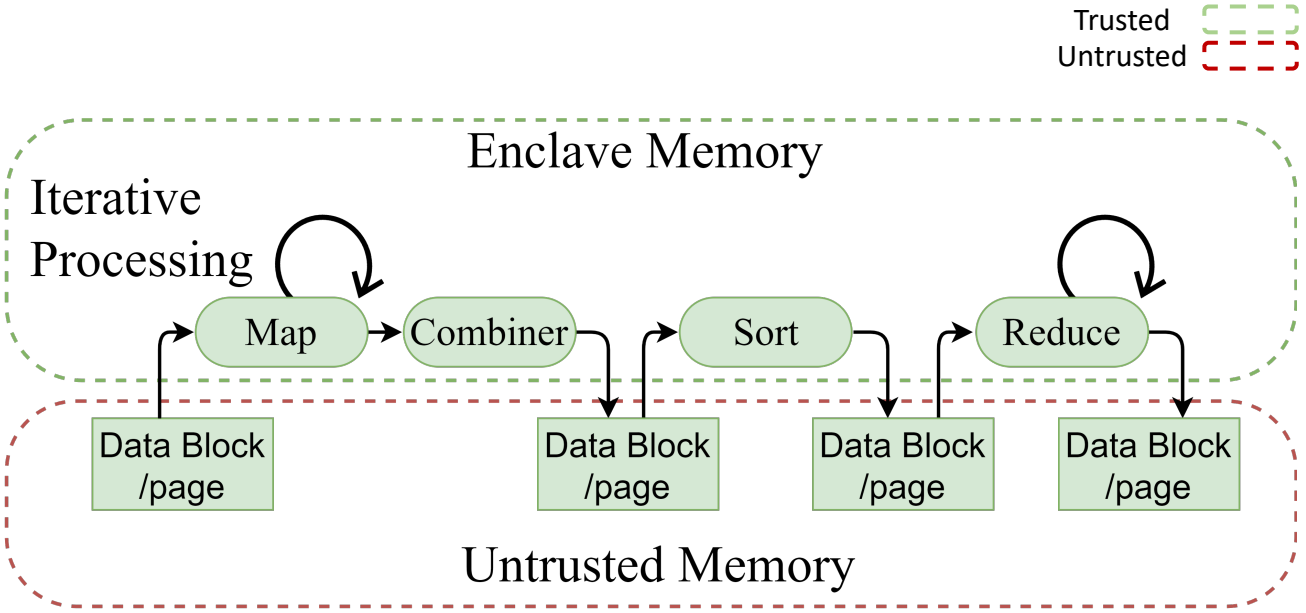
SGX-MR: High Level Design

- Framework divides into two parts
- Untrusted component contains
  - small initiator program
  - encrypted files and data blocks
- Enclave holds the rest
  - MR Controller manages the data flow
  - Only access equal size blocks
    - sequentially
    - obliviously
  - User define functions execute in Enclave
    - Mapper
    - Combiner
    - Reducer



SGX-MR: Regulating Data-flows

- Process 3-4 blocks in Enclave at a time
- Map phase
  - Read blocks sequentially
  - Perform mapping and send results to combiner
  - Combiner aggregates output per block
  - Sequentially writes blocks
- Sort Phase
  - Perform block-wise sorting
  - Reads and Writes data blocks
- Reduce
  - Sequentially reads sorted blocks
  - Aggregate group-wise results
  - Sequentially writes the blocks

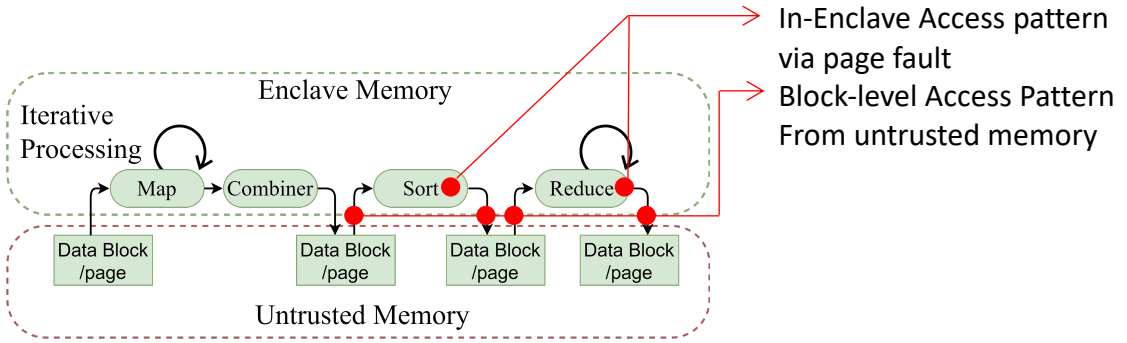


Regulating Data-flows between Trusted and Untrusted Memory

SGX-MR: Analyzing Access Pattern Leakages

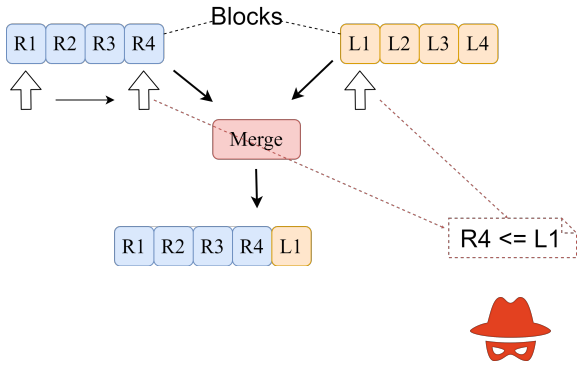
Our Analysis

- Multiple critical section leaks sensitive information
- Analyze each of the leakages
- Propose mitigation methods for each leakages



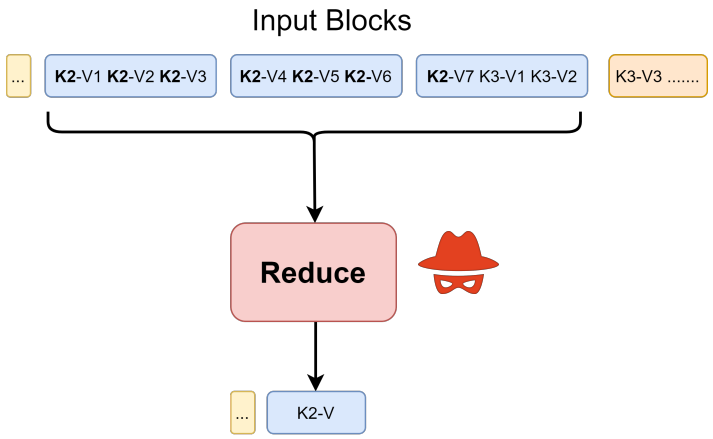
1

Leakages in Merge Sort



2

Reduce phase leaks group size



3

In-Enclave Access Pattern

```
...
if ( a > c )
    // page access
    swap(a,c)
else
    // no page access
...
```

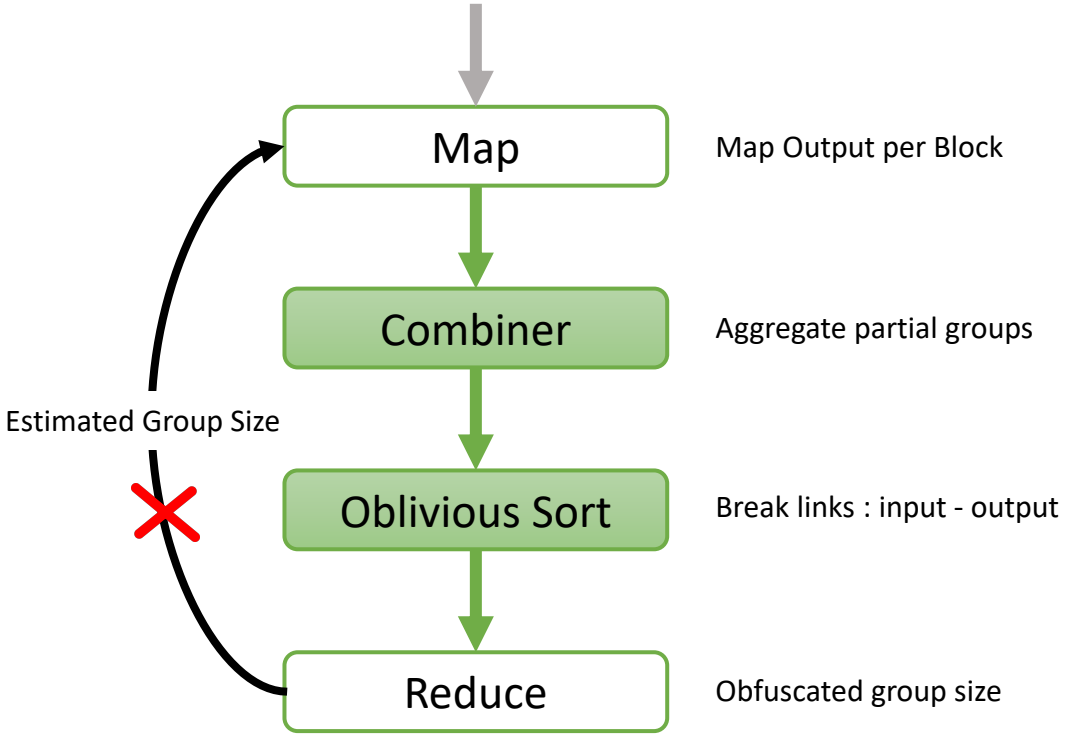
Data dependent branching

SGX-MR: Access Pattern Leakages - Mitigation

Mitigation Methods

- Replacing Merge Sort with Oblivious Sort. E.g., Bitonic Sort
- Applying oblivious_swap, oblivious_copy, oblivious_merge, etc.
- Making the block-level combiner mandatory

Synergy of curated components efficiently hides the access pattern leakages



SGX-MR: Experimental Evaluation

Experimental Setup

- SGX-MR implemented in C++
- Core framework written within 2000 lines of code
- Entire framework runs in Enclave
  - Except starter program
- Implemented custom Bitonic Sort
  - Adopt block level operation
  - Covers all proposed mitigation methods
- Leverages CMOV instruction
  - Conditional swaps
  - Sensitive branching patterns
- Relies on 128-bit AES-CTR mode encryption (SGX SDK)

SGX-MR: Experimental Evaluation

Sample Application

- Experimented SGX-MR with two Applications
  - WordCount Problem
  - KMeans Clustering

Baseline Version

- Compare results with ORAM based approach
- Applied ORAM integrated Merge Sort with CMOV protection
- We utilized ZeroTrace's ORAM implementation

SGX-MR: Experimental Results

Protecting Group Size

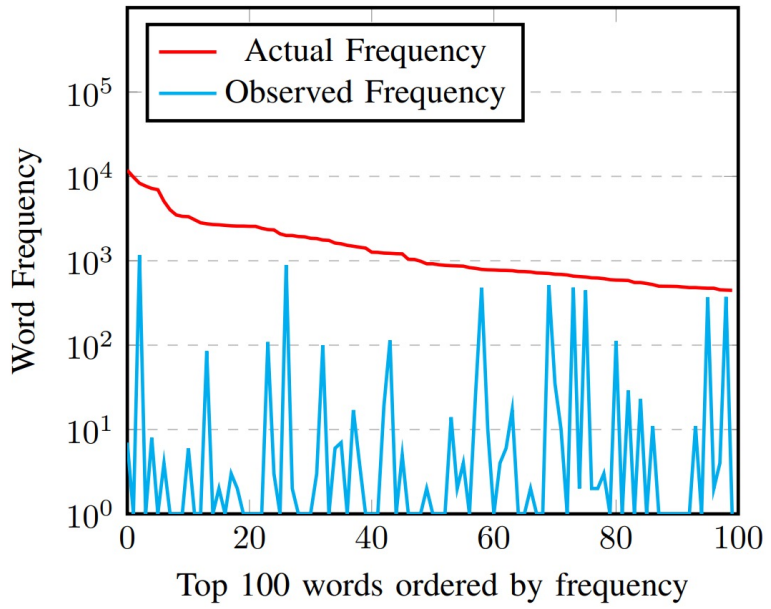


Figure A

WordCount: Observed frequency vs actual frequency

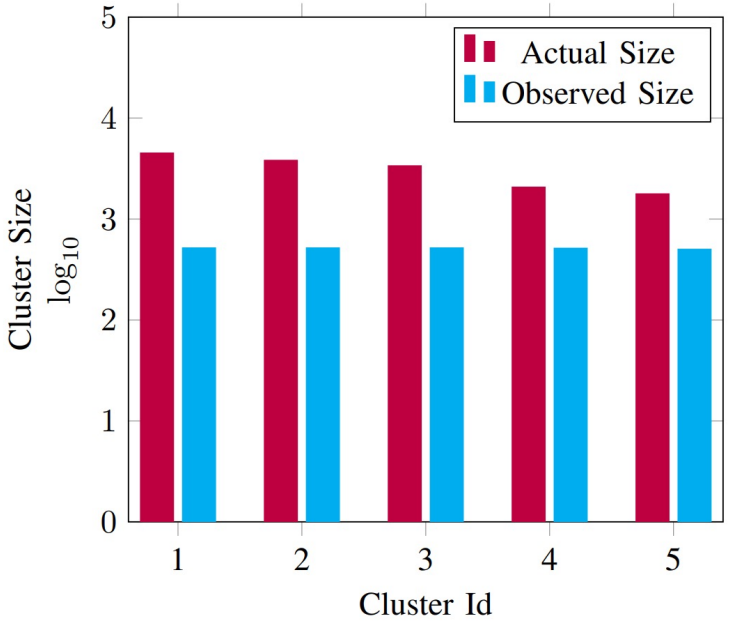


Figure B

KMeans: Observed cluster size vs actual cluster size

SGX-MR: Experimental Results

Finally, Application-based Evaluation

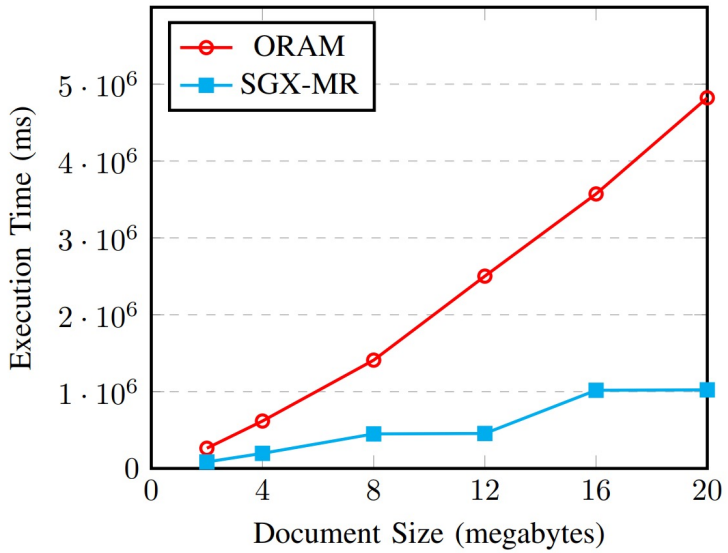


Figure A

Application Level Comparison of WordCount

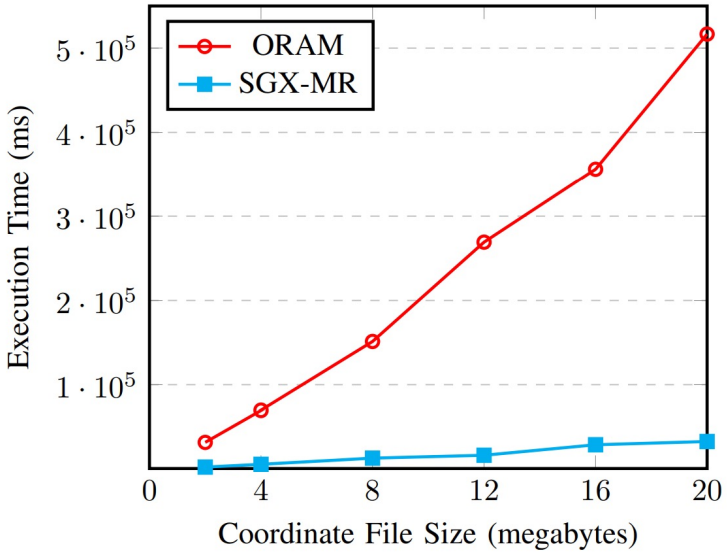


Figure B

Application Level Comparison of KMeans Clustering

SGX-MR: Summary

- SGX-MR avoids
  - expensive ORAM as block I/O and
  - error-prone application-specific design of access pattern protection
- It uses MapReduce to regulate application dataflows – protects a large class of data-intensive applications
- It addresses both access-pattern attacks and page-fault attacks

Additional Slides

Traceable Memory Access: Example

Consider a Malicious OS

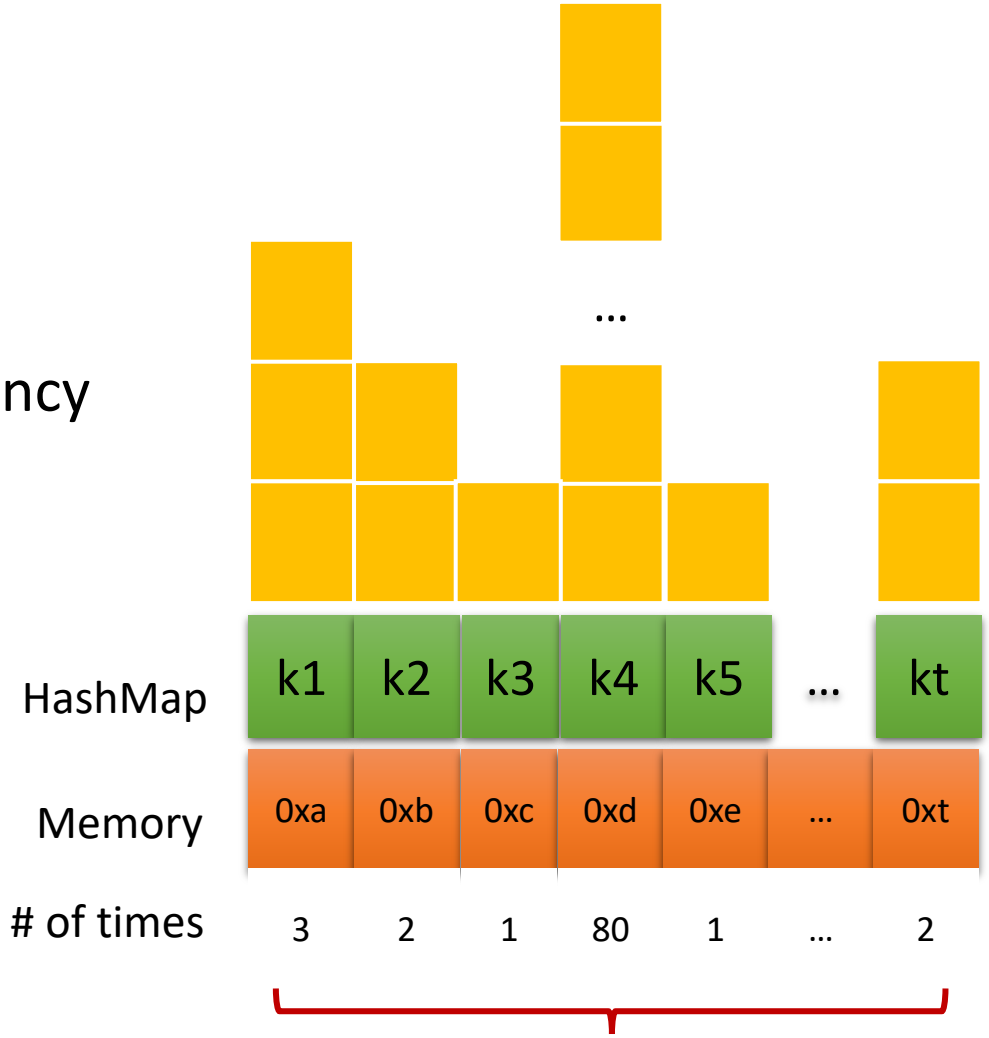
- Cannot see actual data
- Capable of observing Memory Accesses

e.g., Applying HashMap to compute frequency

Sensitive Array: x1 x2 x3 x4 x5 … xn

Sequentially read the array (safe)

- Update the HashMap
- Direct Access to memory

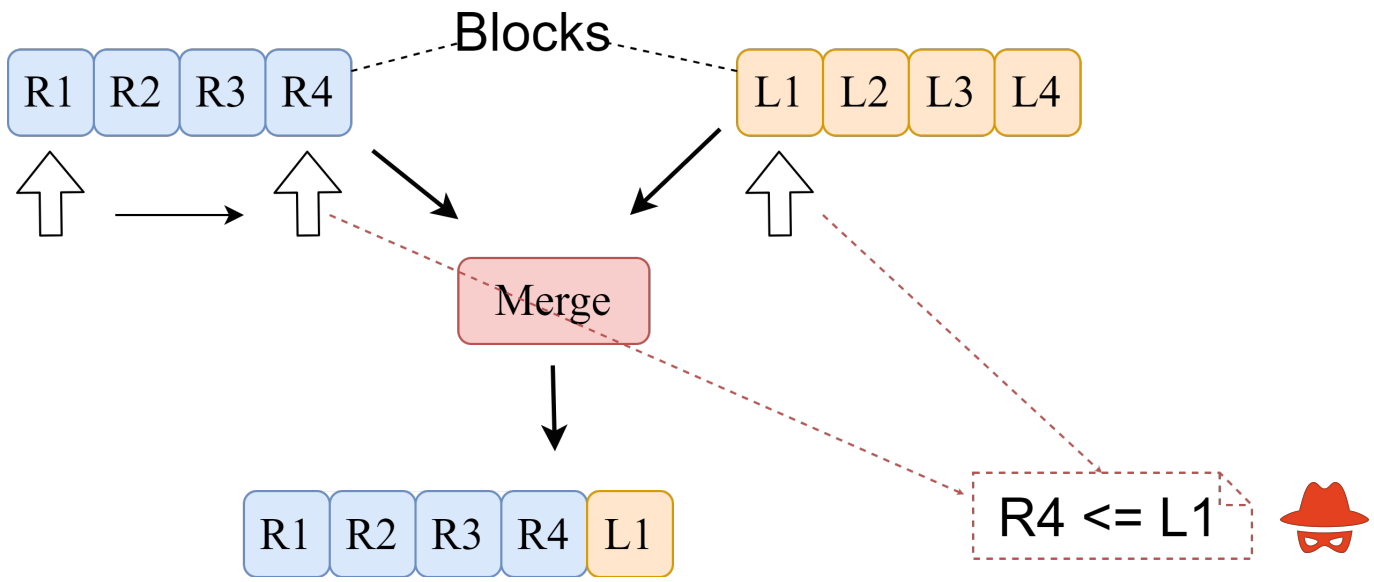


Reveals frequency distribution

SGX-MR: Access Pattern Leakages - Sorting

Sorting Algorithm (i.e., Merge Sort) leaks relative order

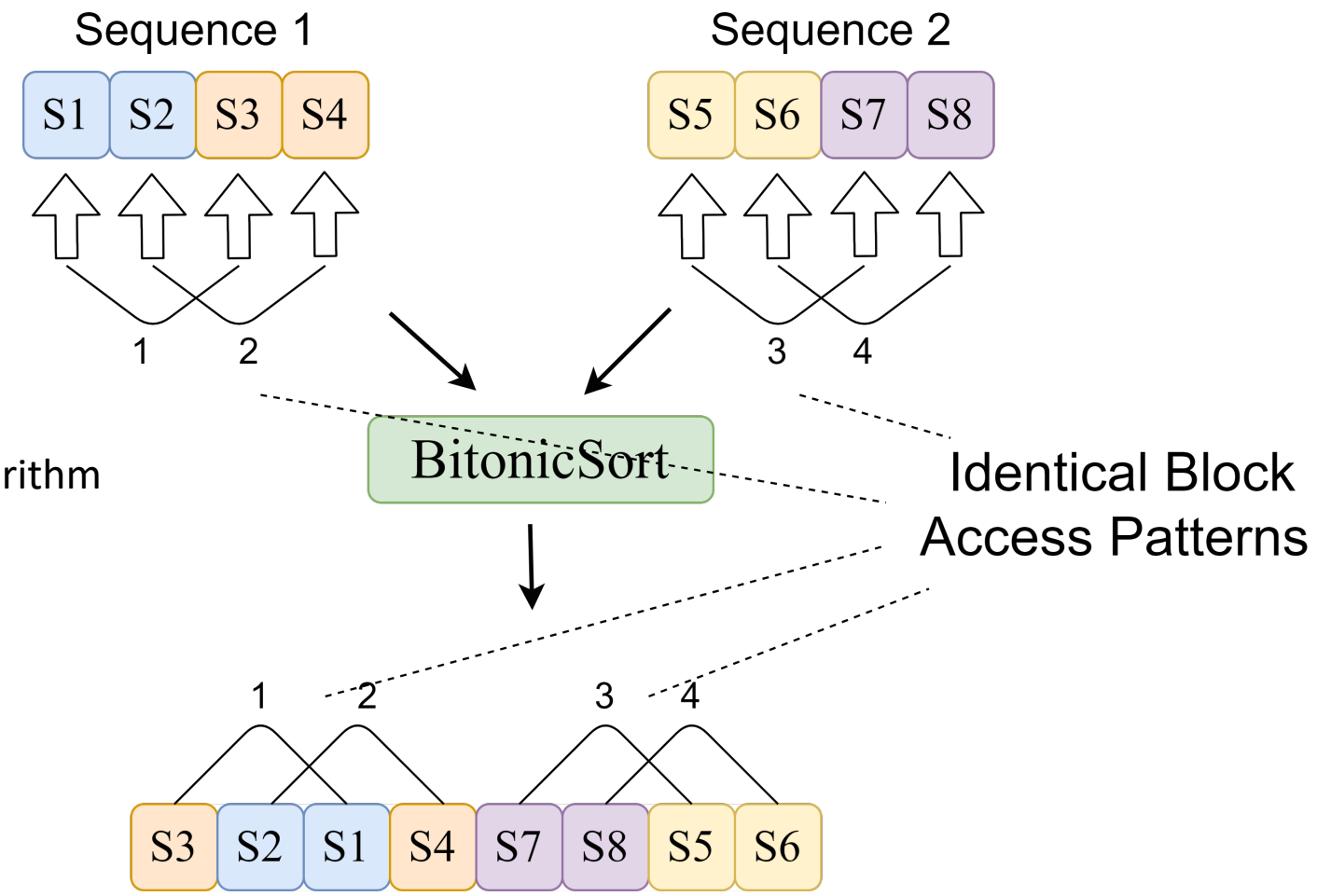
- Adversary observes block reading pattern from Untrusted Memory
- If R4 ≤ L1, then the Merge Phase reads R1 to R4 one by
- Then reads L1 to L4.
- Reading pattern leaks relative order of the blocks



SGX-MR: Access Pattern Leakages - Sorting

Mitigation for Block Access

- Replacing Merge Sort with Oblivious Sorting algorithm
- We implemented block-level Bitonic Sort
- Uniform block access does not leak pattern



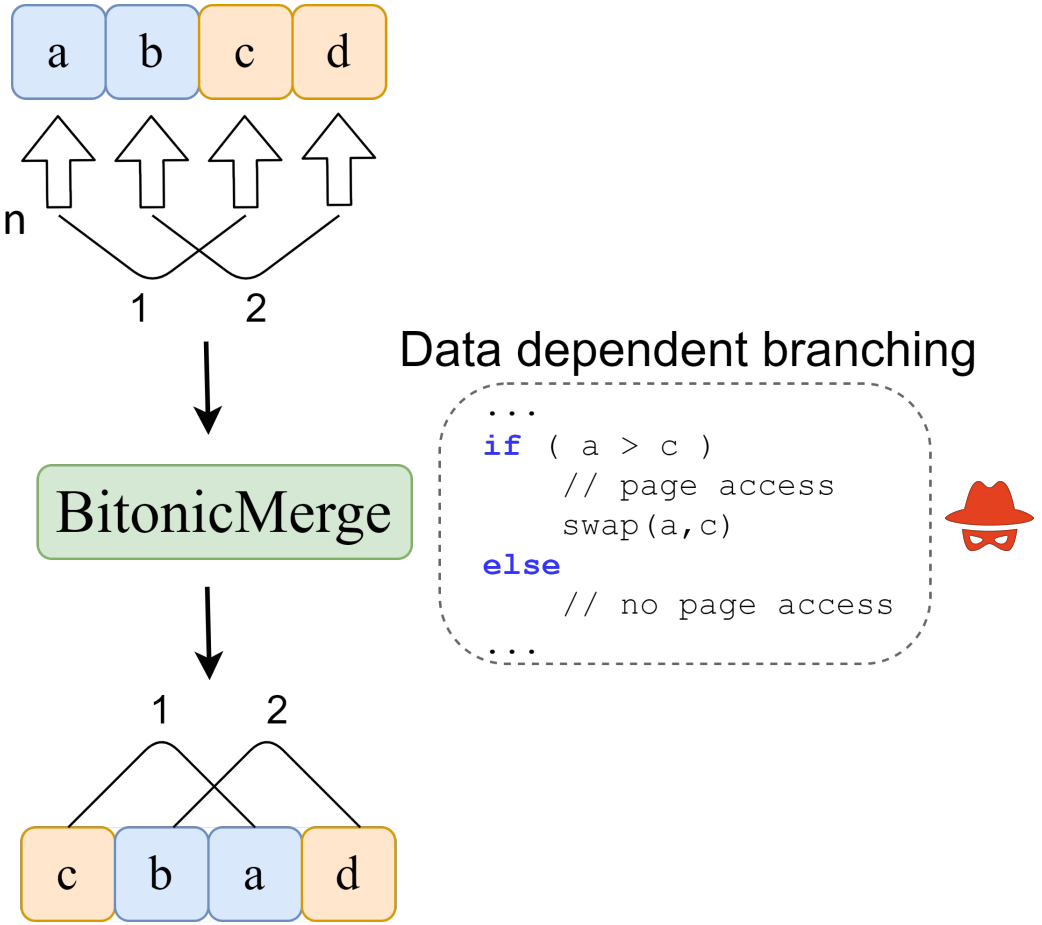
SGX-MR: Access Pattern Leakages - Sorting

Leakage for In-Enclave Memory Access

- Page-fault attack reveals access pattern for Enclave Memory
- Reveals memory page access
- Sequential memory access does not reveal additional information

Compare and Swap in Bitonic Sort

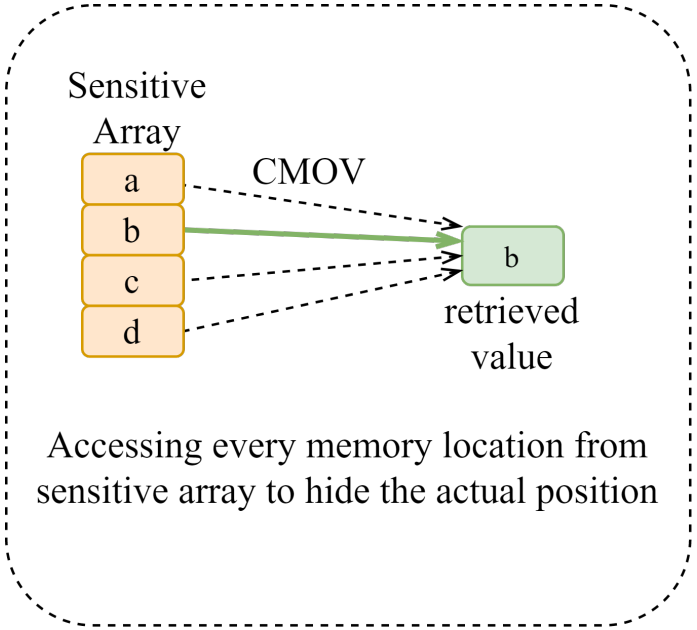
- Generates data dependent branching
- Reveals partial ordering of records



SGX-MR: Access Pattern Leakages - Sorting

Mitigation for In-Enclave Memory Access

- Adopt oblivious swap, oblivious move, etc.
- Leverage CMOV (conditional move) instruction from x86 instruction set



Copy Information from sensitive array

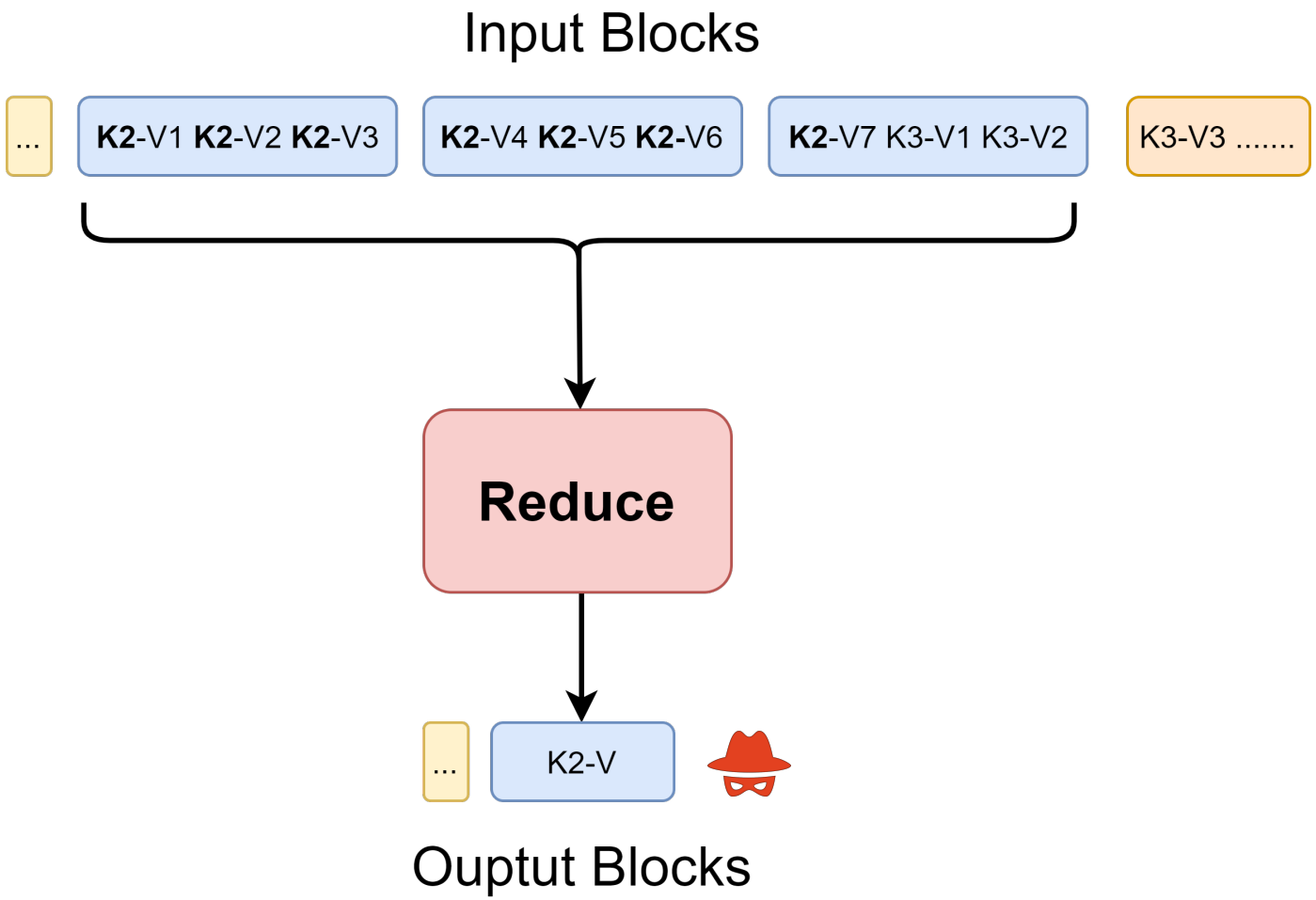
```
...
if ( a > c )
    // page access
    o_swap(a,c,true)
else
    // page access
    O_swap(a,c,false)
...
```

Oblivious swap

SGX-MR: Access Pattern Leakages - Reducing

Reduce phase leaks group size

- K2 spread over three blocks
- Reads three blocks but write one block
- Adversary observes group wise aggregation
- Reveals average group size



SGX-MR: Experimental Results

Block Access

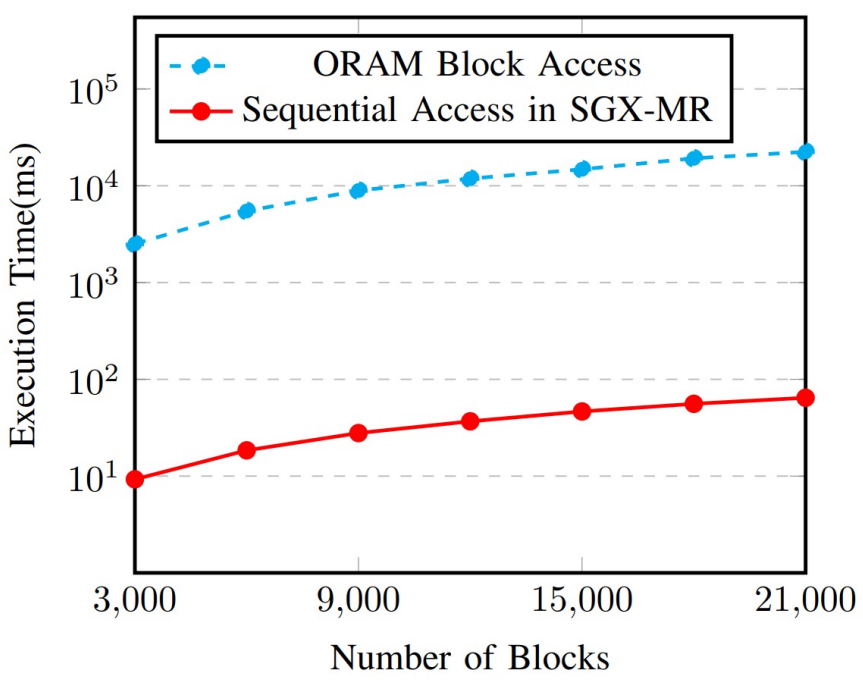


Figure A

ORAM protected Block access can be x1000 times slower than sequential Block Access

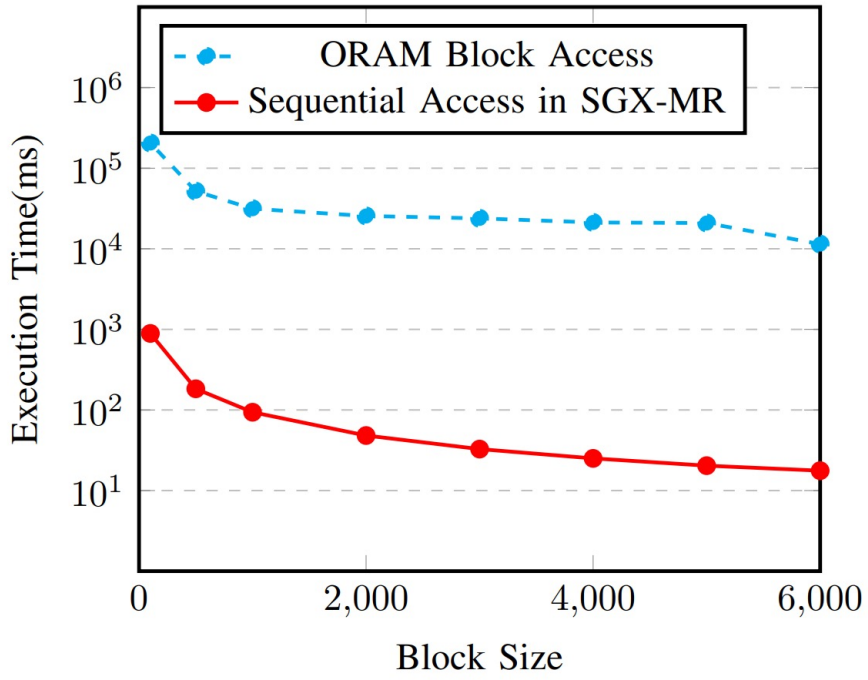


Figure B

While the block size is increased from 1 KB to 6 MB, the execution time is reduced by about 50 to 100 times

SGX-MR: Experimental Results

Oblivious Sort vs ORAM integrated Merge Sort

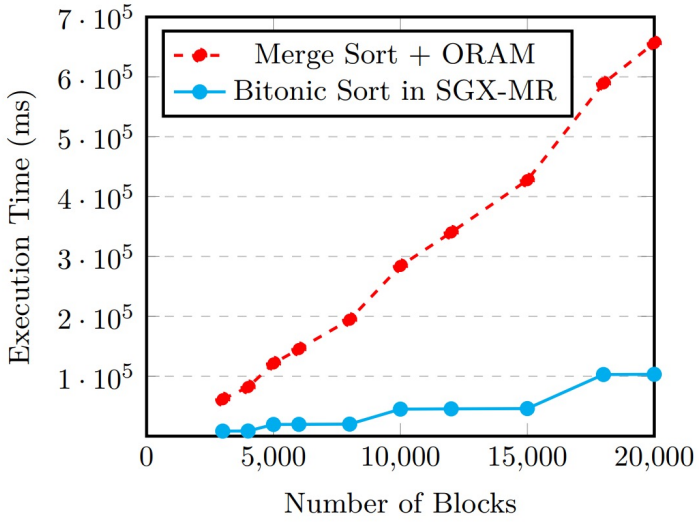


Figure A

MergeSort with the ORAM block I/O results
significantly higher costs than
a dedicated oblivious sorting

SGX-MR: Experimental Results

Mitigations for Sorting

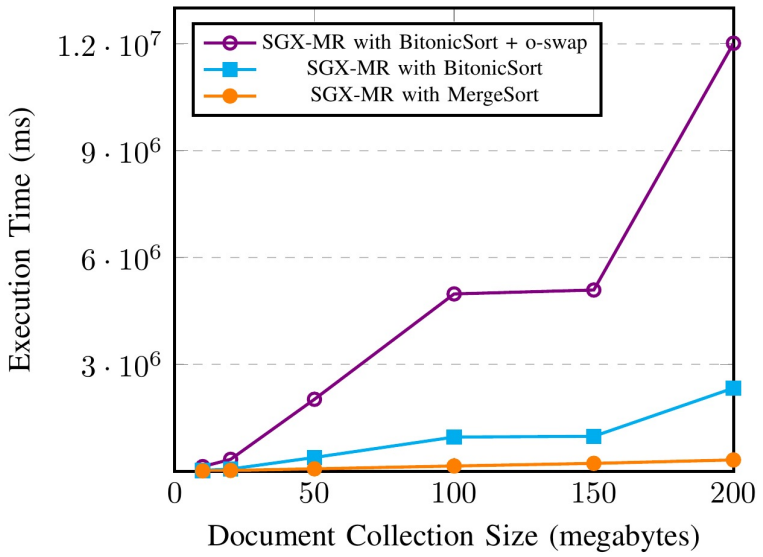


Figure A

Mitigation method brings noticeable cost

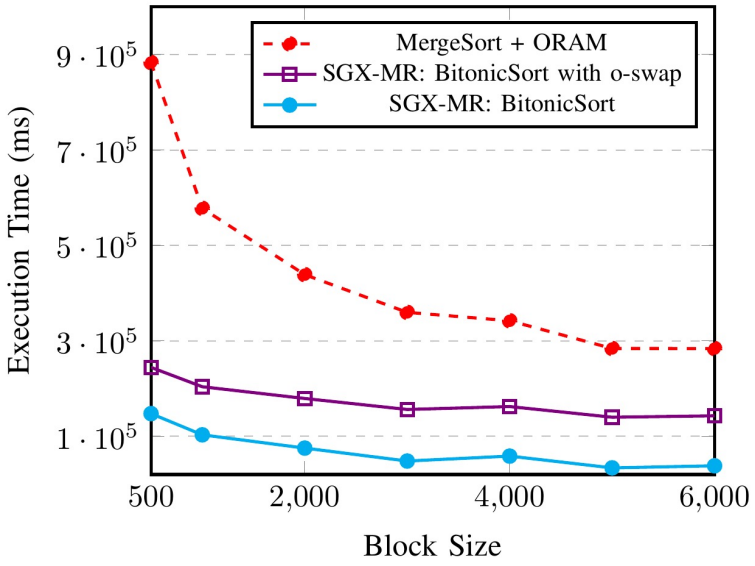


Figure B*

Still, efficient than ORAM based solution

*Figure B also shows block size have significant impact on execution time