Privacy-Preserving Computation with Trusted Computing via Scramble-then-Compute

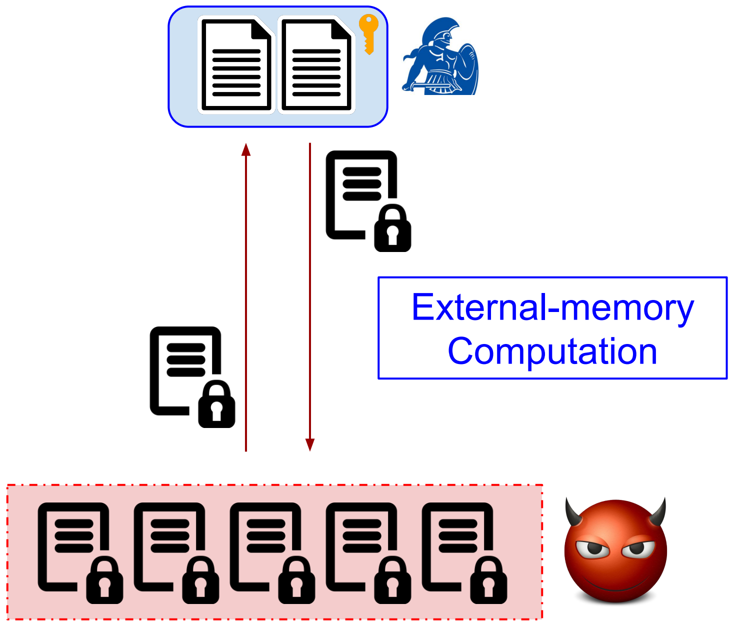
Hung Dang, Anh Dinh, Ee-Chien Chang, Beng Chin Ooi

School of Computing
National University of Singapore

PETS 2017

The Problem

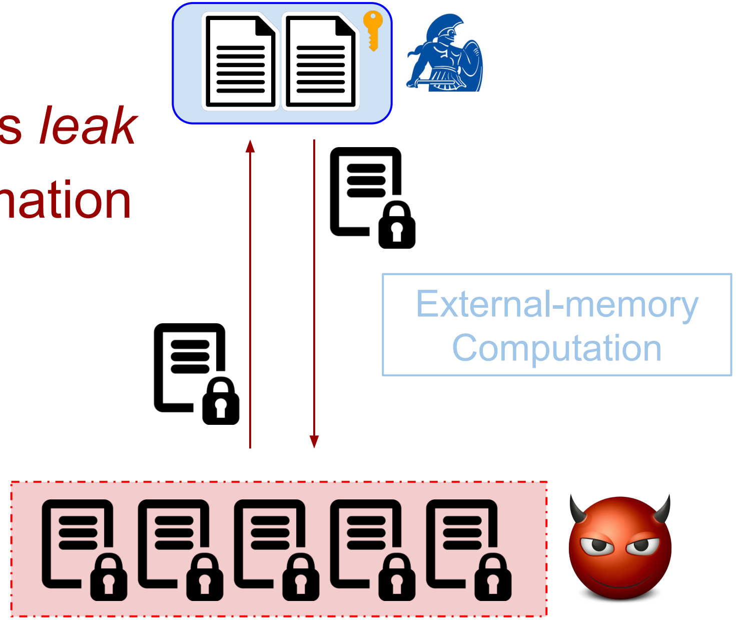
- Context: Processing large dataset with bounded private memory
- System and Threat Model:
    - Data is processed in an trusted execution environment with *bounded private memory*
    - Data *remains encrypted outside* the trusted enviroment
    - The adversary observes access patterns, but cannot see the trusted environment's internal state



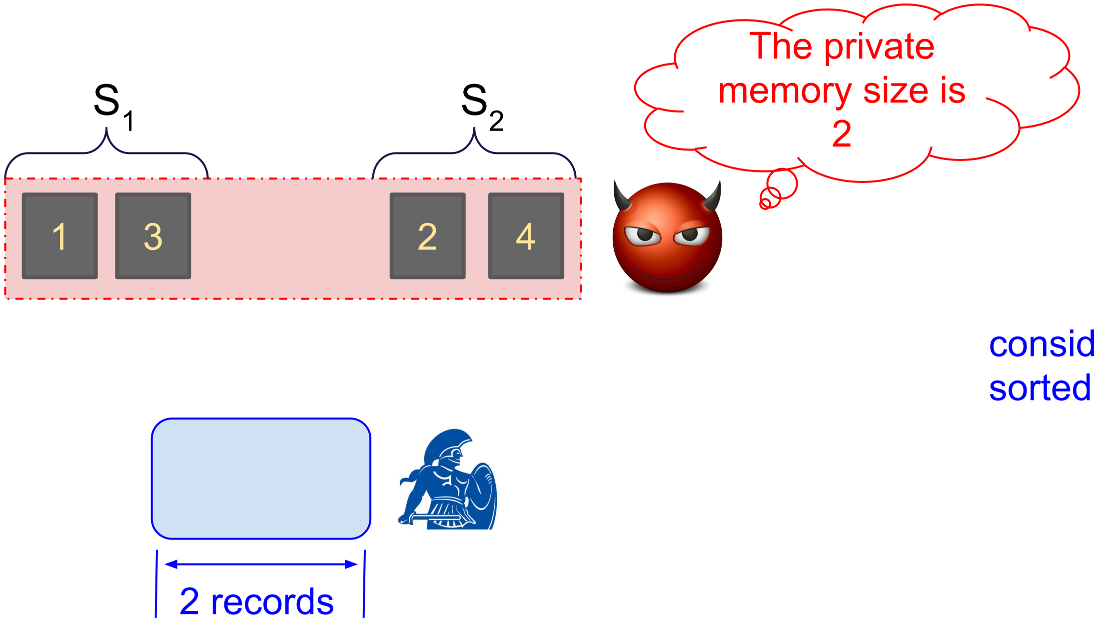
The Problem

- Context: Processing large dataset with bounded private memory
- System and Threat Model:
  - Data is processed in an trusted execution environment with *bounded private memory*
  - Data *remains encrypted outside* the trusted enviroment
  - The adversary observes access patterns, but cannot see the trusted environment's internal state

Access patterns *leak* sensitive information

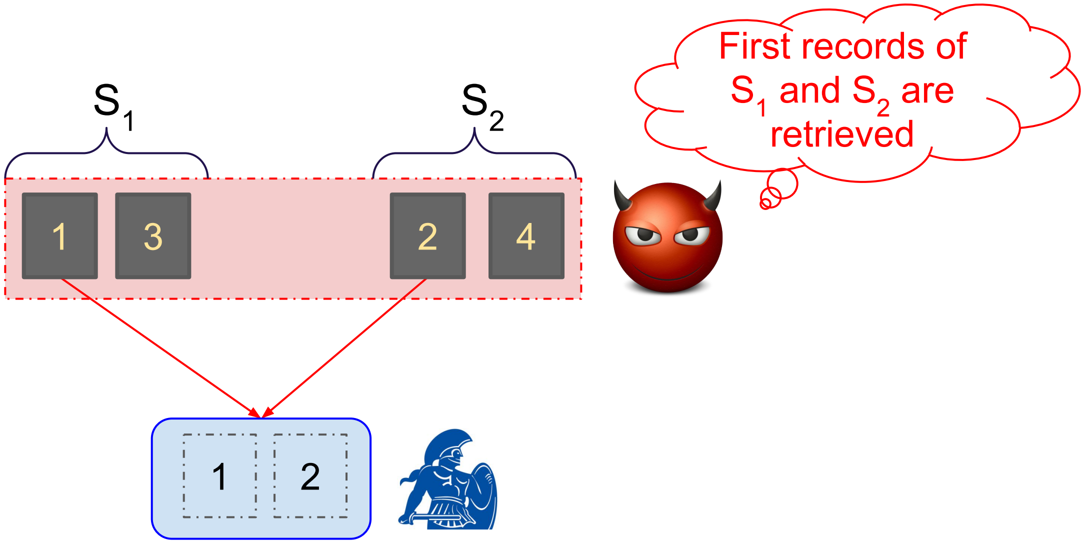


Access Pattern Leakage: Example

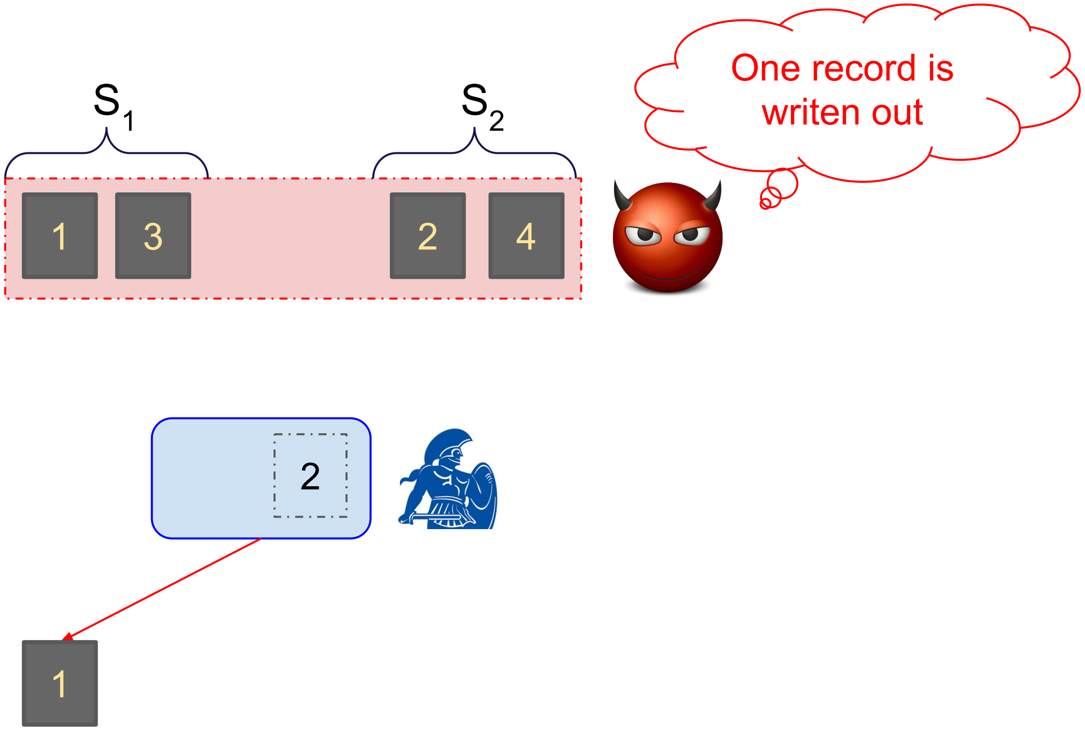


consider merging two sorted sub-arrays

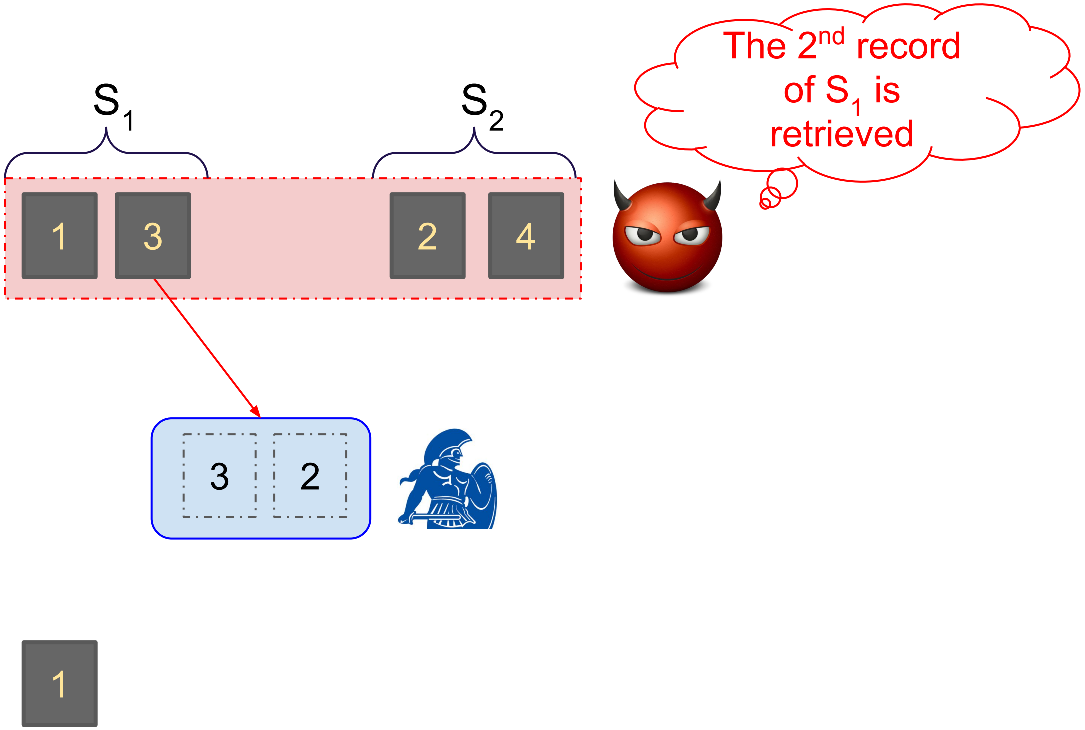
Access Pattern Leakage: Example



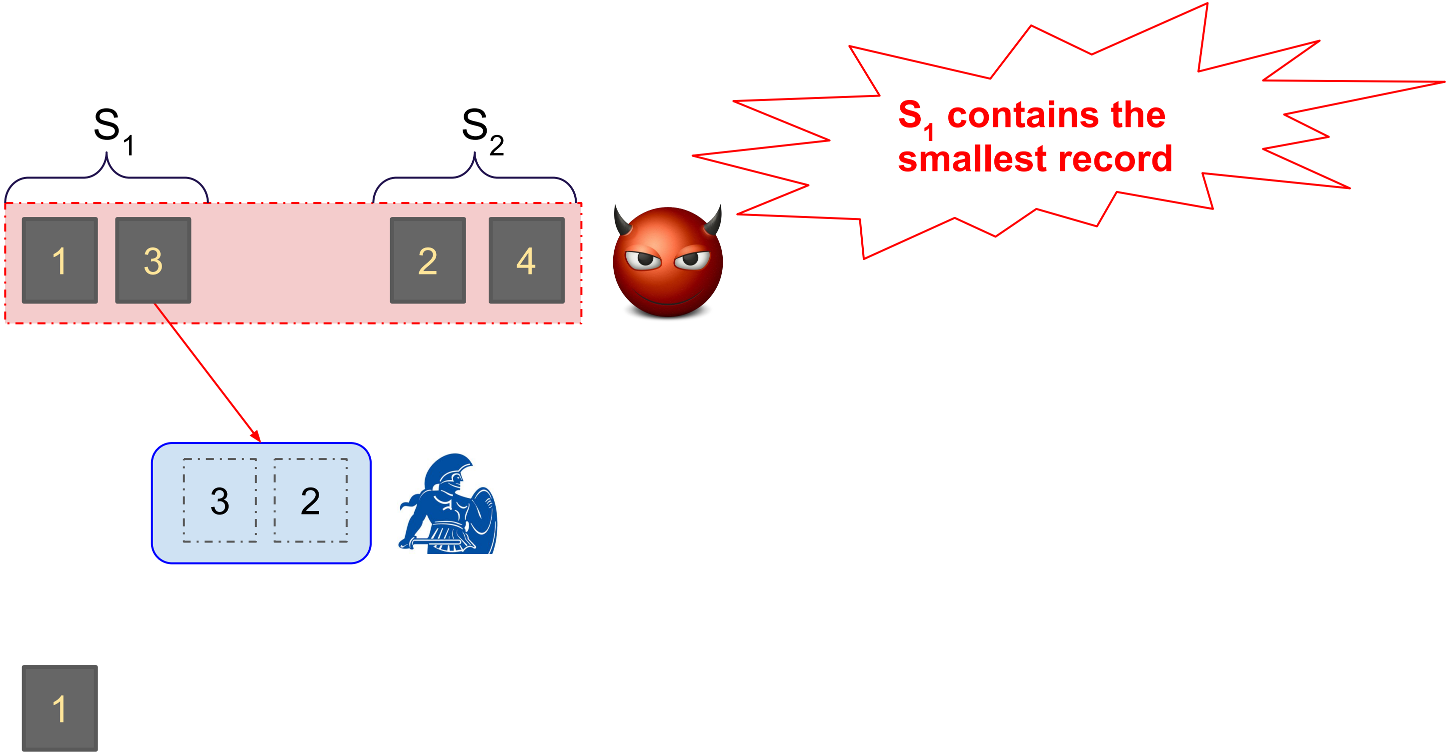
Access Pattern Leakage: Example



Access Pattern Leakage: Example



Access Pattern Leakage: Example



Possible Mitigations

- ORAM (Oblivious RAM)
  - Generic
  - Expensive: incurs $\Omega(\log n)$ (amortized) overheads *per each access*
    - Not suitable for applications accessing entire dataset (e.g., sort, aggregation)
- Tailor-made Algorithms (Data-Oblivious algorithms)
  - Application-specific
  - More efficient (than employing ORAM)
  - Complex construction
    - Hard to implement and vet the trusted code base (TCB)

Our Solution

We seek an approach to design *privacy-preserving algorithms* that is:

- Expressive
    - Enable adoption of state-of-the-art external memory algorithms
- Simple
    - Ease of implementation and TCB vetting
- Low overhead

Scramble-then-Compute ($\mathcal{STC}$)

Derive a *privacy-preserving* algorithm from an efficient but not necessarily privacy-preserving one:

- Privately scramble the input
  - Conceal correspondences between the original input and the scrambled data
- Apply the original (external-memory) algorithm on the scrambled data
  - Leverage on extensive studies to adopt the most suitable algorithm with the most well-tuned parameteres for a particular application at hand

Scramble then Compute ($\mathcal{STC}$)

Derive a *privacy-preserving* algorithm from an efficient but not necessarily privacy-preserving one:

- Privately scramble the input
  - Conceal correspondences between the original input and the scrambled data
- Apply the original (external-memory) algorithm on the scrambled data
  - Leverage on extensive studies to adopt the most suitable algorithm with the most well-tuned parameteres for a particular application at hand

Simplicity ✔

$\mathcal{STC}$ - Scope

$\mathcal{STC}$ supports a permutation-invariant[#] algorithm $\mathcal{P}$ if there exists an imitator $\langle \mathcal{T}, \mathcal{P}^{\star} \rangle$ of $\mathcal{P}$

- $\mathcal{T}$, given $\mathcal{X}$, outputs a permuted sequence of $\langle 1,2,...,n \rangle$
- $\mathcal{P}^{\star}$ operates on $\mathcal{T}(\mathcal{X})$ *exactly the same* as $\mathcal{P}$ does on $\mathcal{X}$ (i.e., incur the same access pattern)

[#] outputs the same $\mathcal{Y}$ for any permutation of $\mathcal{X}$

$\mathcal{STC}$ - Scope

$\mathcal{STC}$ supports a permutation-invariant# algorithm $\mathcal{P}$ if there exists an imitator $\langle \mathcal{T}, \mathcal{P}^{\star} \rangle$ of $\mathcal{P}$

- $\mathcal{T}$, given $\mathcal{X}$, outputs a permuted sequence of $\langle 1,2,...,n \rangle$
- $\mathcal{P}^{\star}$ operates on $\mathcal{T}(\mathcal{X})$ *exactly the same* as $\mathcal{P}$ does on $\mathcal{X}$ (i.e., incur the same access pattern)

Expressiveness✔

outputs the same $\mathcal{Y}$ for any permutation of $\mathcal{X}$

$\mathcal{STC}$ - A Closer Look

Given $\mathcal{P}$ operating on input X, $\mathcal{STC}$ derives a privacy-preserving algorithm $\mathcal{A}_{\mathcal{P}}$:

1. *$X' \leftarrow$ Pre-Process (X) (if required)*
2. *$S \leftarrow$ Scramble (X')*
3. *$Y' \leftarrow \mathcal{P}(S)$*
4. *$Y \leftarrow$ Post-Process (Y') (if required)*

$\mathcal{STC}$ - A Closer Look

Given $\mathcal{P}$ operating on input $\mathcal{X}$, $\mathcal{STC}$ derives a privacy-preserving algorithm $\mathcal{A}_{\mathcal{P}}$:

1. $\mathcal{X}' \leftarrow$ *Pre-Process* $(\mathcal{X})$ *(if required)*
   - ➢ ensure permutation-invariant requirement
2. $\mathcal{S} \leftarrow$ *Scramble* $(\mathcal{X}')$
   - ➢ based on Melbourne Shuffle Algorithm
3. $\mathcal{Y}' \leftarrow \mathcal{P}(\mathcal{S})$
4. $\mathcal{Y} \leftarrow$ *Post-Process* $(\mathcal{Y}')$ *(if required)*
   - ➢ reverse effect of step 1

➢ Data Oblivious
➢ Requires private memory of size O(√n)
➢ Runtime O(n)

$\mathcal{STC}$ - A Closer Look

Given $\mathcal{P}$ operating on input $\mathcal{X}$, $\mathcal{STC}$ derives a privacy-preserving algorithm $\mathcal{A}_{\mathcal{P}}$:

Low overhead ✔

1. *X' ← Pre-Process (X) (if required)*
   - ensure permutation-invariant requirement
2. *S ← Scramble (X')*
   - based on Melbourne Shuffle Algorithm
3. *Y' ← P(S)*
4. *Y ← Post-Process (Y') (if required)*
   - reverse effect of step 1

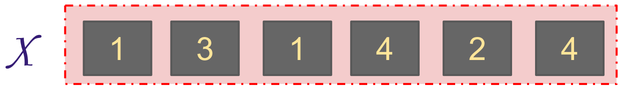
- Data Oblivious
- Requires private memory of size O(√n)
- Runtime O(n)

$\mathcal{STC}$ - A Closer Look

Given $\mathcal{P}$ operating on input $\mathcal{X}$, $\mathcal{STC}$ derives a privacy-preserving algorithm $\mathcal{A}_{\mathcal{P}}$:

1. $X' \leftarrow \textit{Pre-Process}(X)$ *(if required)*
2. $S \leftarrow \textit{Scramble}(X')$
3. $Y' \leftarrow P(S)$
4. $Y \leftarrow \textit{Post-Process}(Y')$ *(if required)*

E.g.,: Deriving a privacy-preserving sorting algorithm from external merge sort

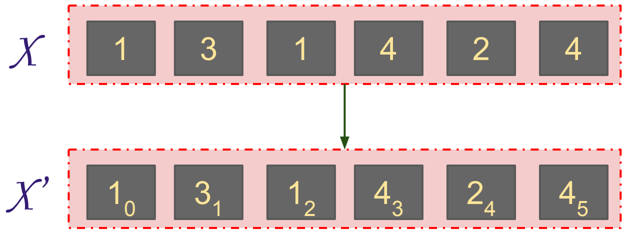


$\mathcal{STC}$ - A Closer Look

Given $\mathcal{P}$ operating on input $\mathcal{X}$, $\mathcal{STC}$ derives a privacy-preserving algorithm $\mathcal{A}_{\mathcal{P}}$:

1. *X' ← Pre-Process (X) (if required)*
2. *S ← Scramble (X')*
3. *Y' ← P(S)*
4. *Y ← Post-Process (Y') (if required)*

Add metadata to handle duplicates

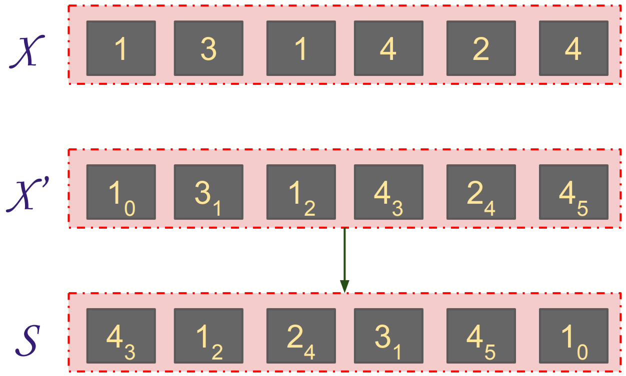


$\mathcal{STC}$ - A Closer Look

Given $\mathcal{P}$ operating on input $\mathcal{X}$, $\mathcal{STC}$ derives a privacy-preserving algorithm $\mathcal{A}_{\mathcal{P}}$:

1. $\mathcal{X}' \leftarrow$ *Pre-Process* $(\mathcal{X})$ *(if required)*
2. $\mathcal{S} \leftarrow$ *Scramble* $(\mathcal{X}')$
3. $\mathcal{Y}' \leftarrow \mathcal{P}(\mathcal{S})$
4. $\mathcal{Y} \leftarrow$ *Post-Process* $(\mathcal{Y}')$ *(if required)*

Privately scramble the input



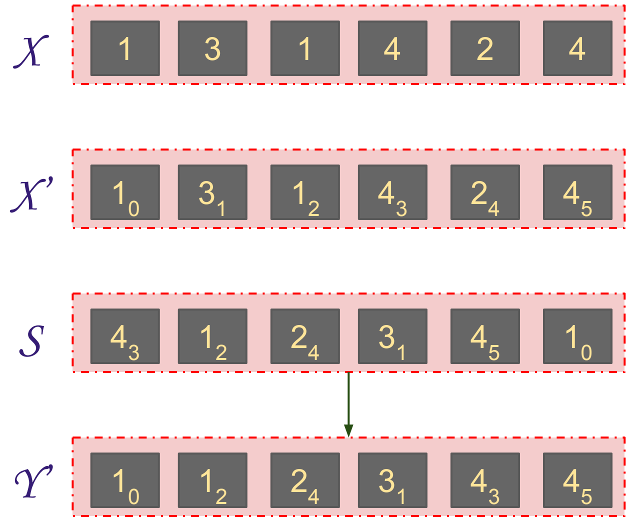
The scrambling hide correspondences between records of $\mathcal{X}'$ and those of $\mathcal{S}$

$\mathcal{STC}$ - A Closer Look

Given $\mathcal{P}$ operating on input $\mathcal{X}$, $\mathcal{STC}$ derives a privacy-preserving algorithm $\mathcal{A}_{\mathcal{P}}$:

1. $\mathcal{X}' \leftarrow$ *Pre-Process* $(\mathcal{X})$ *(if required)*
2. $\mathcal{S} \leftarrow$ *Scramble* $(\mathcal{X}')$
3. $\mathcal{Y}' \leftarrow \mathcal{P}(\mathcal{S})$ ←
4. $\mathcal{Y} \leftarrow$ *Post-Process* $(\mathcal{Y}')$ *(if required)*

Sort the scrambled input by external merge sort



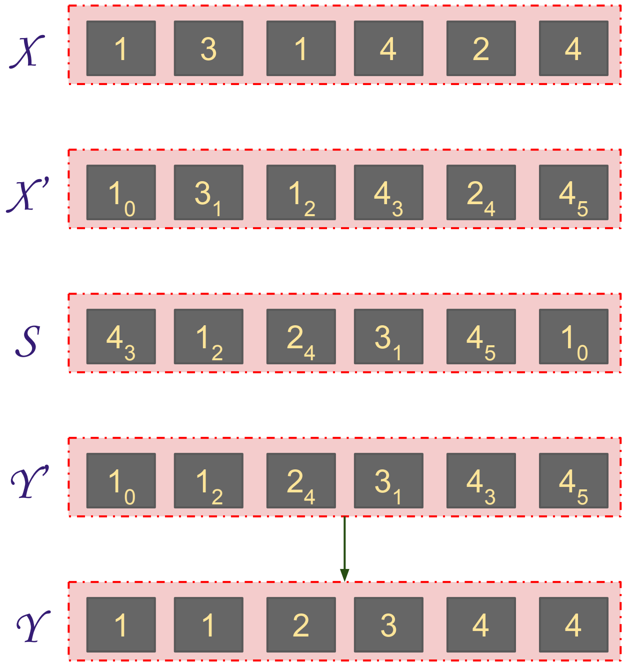
Observation maded on $\mathcal{S}$ cannot be linked back to that of $\mathcal{X}'$

$\mathcal{STC}$ - A Closer Look

Given $\mathcal{P}$ operating on input $\mathcal{X}$, $\mathcal{STC}$ derives a privacy-preserving algorithm $\mathcal{A}_{\mathcal{P}}$:

1. $\mathcal{X}' \leftarrow$ *Pre-Process* $(\mathcal{X})$ *(if required)*
2. $\mathcal{S} \leftarrow$ *Scramble* $(\mathcal{X}')$
3. $\mathcal{Y}' \leftarrow \mathcal{P}(\mathcal{S})$
4. $\mathcal{Y} \leftarrow$ *Post-Process* $(\mathcal{Y}')$ *(if required)*

Remove the metadata



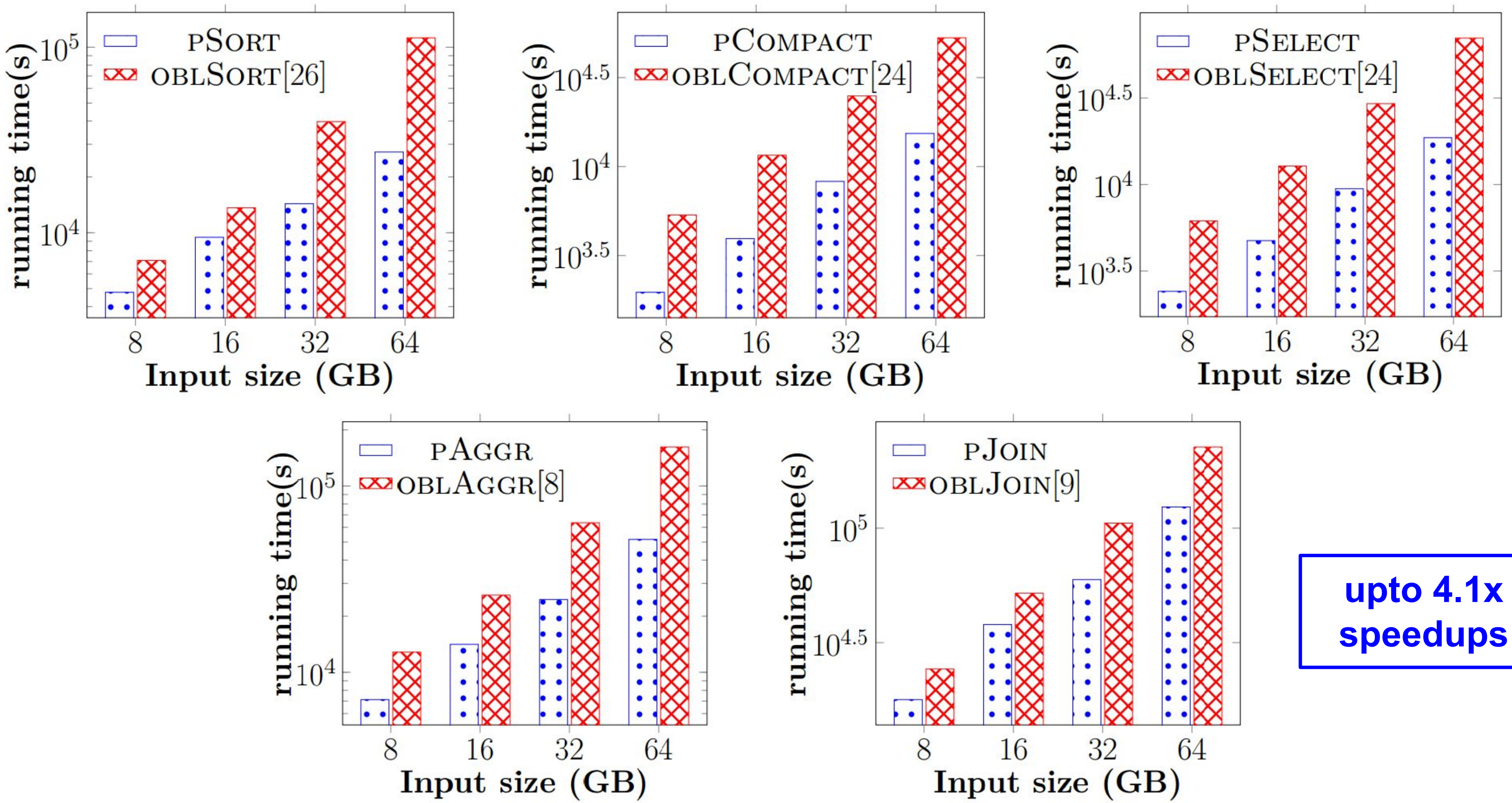
Comparison with Alternative Solutions

	ORAM	*STC*	Tailor-made Algorithm
Performance Overhead	$\Omega(\log n)$ amortized overhead *per each access*	$O(n)$ additive overhead *per execution*	less efficient than *STC* counterpart
Expressiveness	all applications	Spark and many data processing operations	application-specific
Design and Implement Effort	moderate - complicated	simple	complicated

Performance - Running time (s)

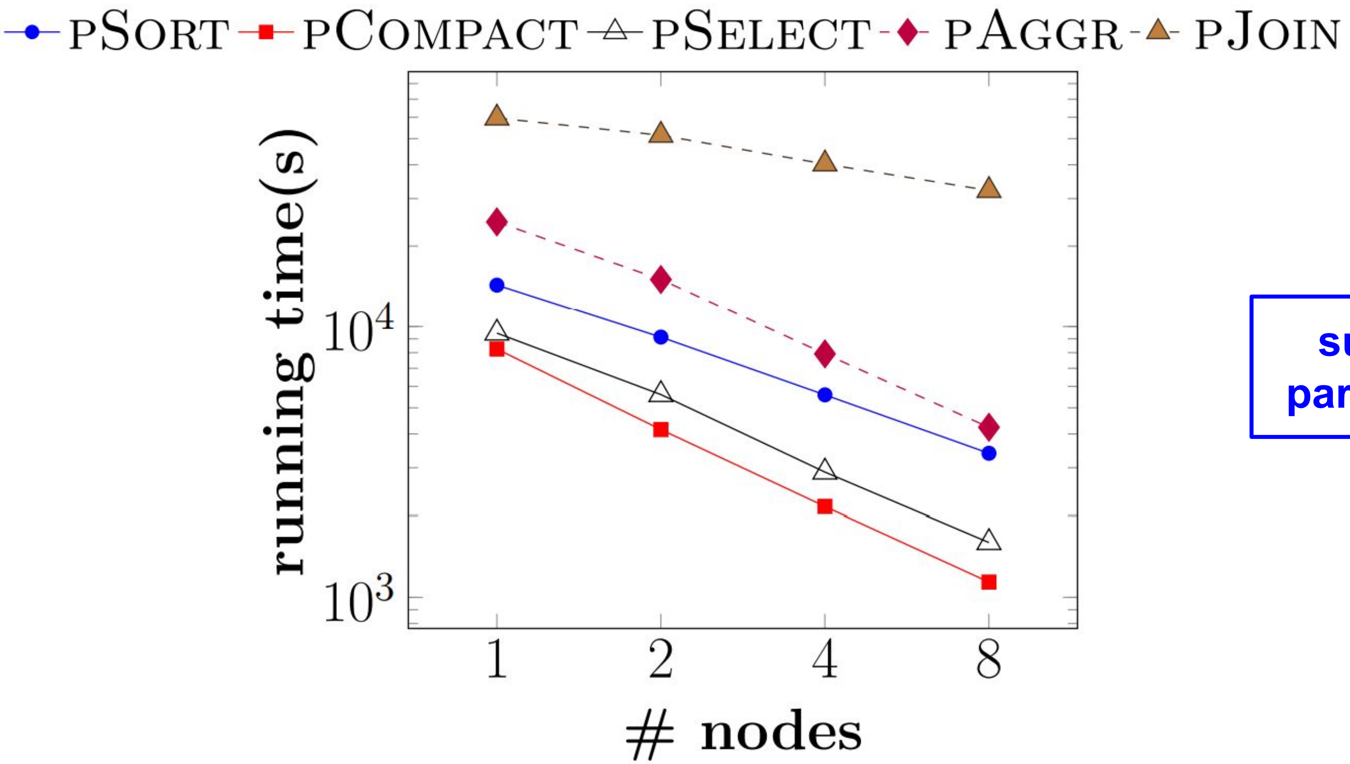
Operation	Baseline	*STC*	Tailor-made Algorithm
Sort	7,961	14,330 (1.79x)	59,628 (7.49x)
Compaction	1,678	82,53 (7.91x)	25,012 (14.89x)
Select	2,758	9,451 (3.42x)	29,365 (16.65x)
Aggregation	10,593	24,578 (2.32x)	63,477 (5.99x)
Join	12,400	59,610 (4.81x)	105,235 (8.49x)

Input size: 32GB (i.e., 2^{28} records)



upto 4.1x speedups

Performance - Scalability



support parallelism

Recaps

$\mathcal{STC}$ enables privacy-preserving computation at ease and at scale with trusted computing:

- Support an expressive class of computations
    - Enabling adoption of state-of-the-art external memory algorithms
- Low performance overhead
- Simple
    - Ease of design, implementation and TCB vetting

Thank you!

Hung Dang
hungdang@comp.nus.edu.sg

Privacy-Preserving Algorithm

Let $Q_{\mathcal{P}}(X)$ be the access patterns (i.e., sequence of read/write) the adversary observe during the execution of an algorithm $\mathcal{P}$ on input X

An algorithm $\mathcal{P}$ is privacy-preserving if for any two datasets X_1 and X_2 with the same number of records, $Q_{\mathcal{P}}(X_1)$ is computationally indistinguishable from $Q_{\mathcal{P}}(X_2)$

Intuition: access patterns do not reveal sensitive information of the input

Relationship to Data Obliviousness

- $\mathcal{P}$ is data-oblivious if $Q_{\mathcal{P}}(X_1) = Q_{\mathcal{P}}(X_2)$ for any X_1 and X_2 having the same number of records
- Data obliviousness implies *perfect zero leakage via access patterns*, while ours implies a *negligible leakage*
- However, since encryption is involved, the security of data oblivious algorithms essentially still rely on indistinguishability

Privacy-Preserving Computations with $\mathcal{STC}$

$\mathcal{STC}$ supports an expressive class of data processing operations including:

- ➢ Sort
- ➢ Compaction
- ➢ Selection
- ➢ Aggregation
- ➢ Join
- ➢ Spark operations

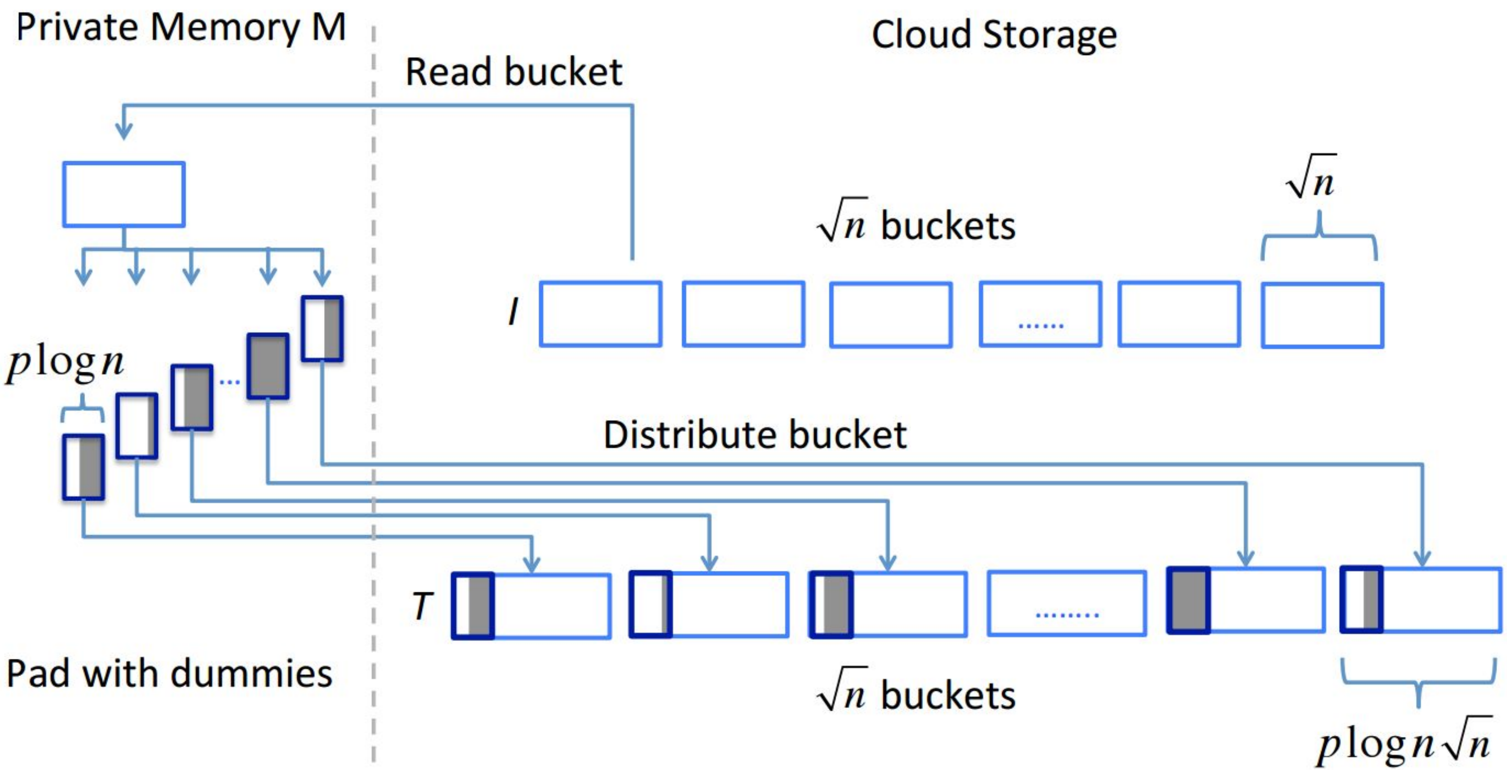
Potential Remedies

- Conventional Encryptions
  - Only protects data at rest
- Homomorphic Encryptions
  - Fully Homomorphic Encryption incurs prohibitive overheads
  - Partially Homomorphic Encryption supports limited operations
- Trusted Computing
  - Access pattern leaks sensitive information

Experiment Setups

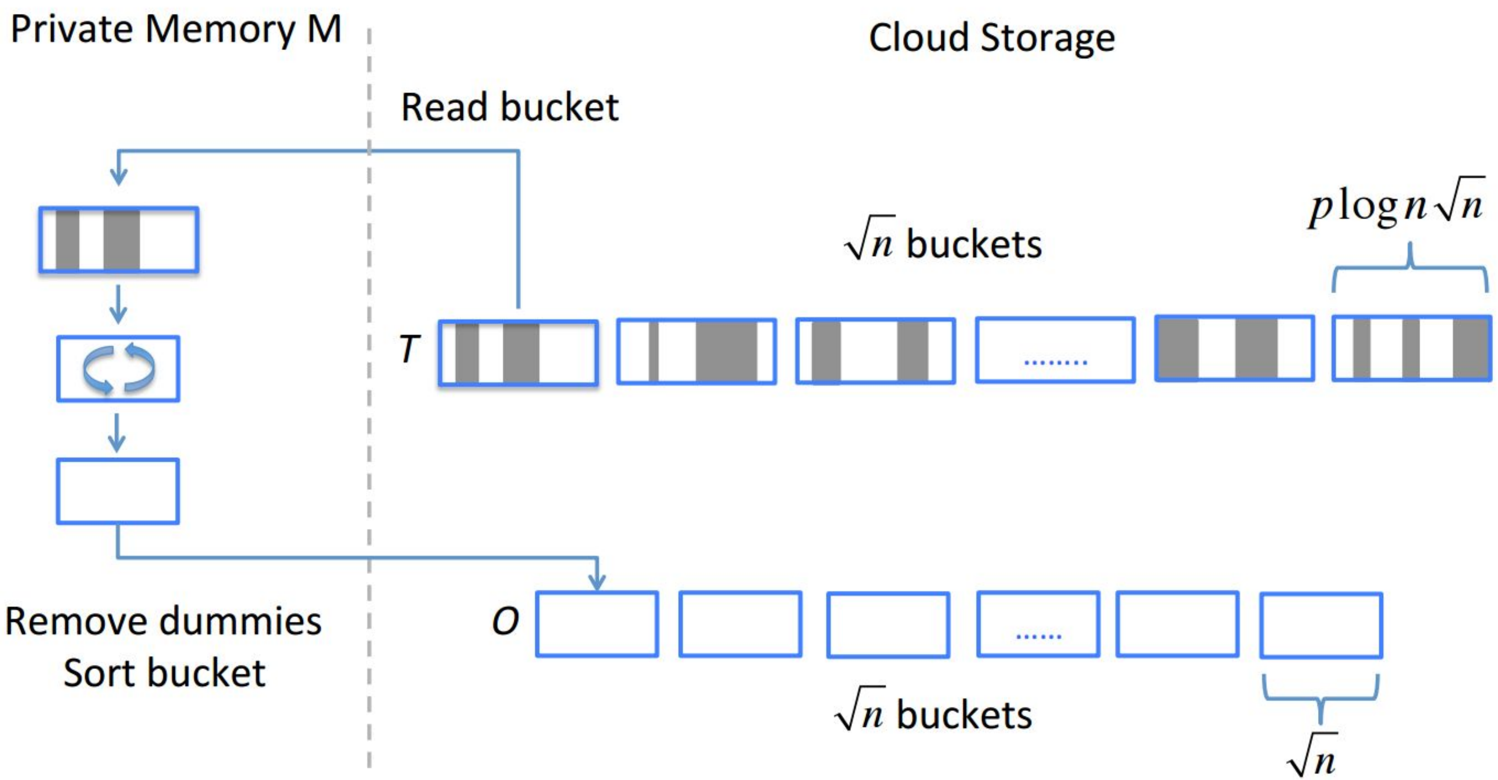
- Machines: Intel Xeon E5-2603 CPU, 8GB of RAM, two 500GB hard drives and two 1GB Ethernet cards
- Simulate trusted hardware (IBM 4767-002 PCIeCC2)
  - CPU clock: 233MHZ
  - Private memory: 64MB
- Input data: generated using Yahoo! TeraSort benchmark
  - Each record comprises 10-byte key and 90-byte value
  - 256-bit key AES encryption
  - Input size varies from 8 - 64 GB

Melbourne Shuffle - Distribution phase



courtesy of Ohrimenko et al.

Melbourne Shuffle - Cleanup phase



courtesy of Ohrimenko et al.