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 environment
  - 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 environment
  - The adversary observes access patterns, but cannot see the trusted environment’s internal state

Access patterns *leak* sensitive information
Access Pattern Leakage: Example

The private memory size is 2

consider merging two sorted sub-arrays
Access Pattern Leakage: Example

First records of $S_1$ and $S_2$ are retrieved
Access Pattern Leakage: Example

One record is written out

PETS 2017
Access Pattern Leakage: Example

The 2\textsuperscript{nd} record of $S_1$ is retrieved.
Access Pattern Leakage: Example

$S_1$ contains the smallest record
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 (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 (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 parameters for a particular application at hand

Simplicity ✔
**STC - Scope**

STC supports a permutation-invariant\(^\#\) algorithm \(P\) if there exists an imitator \(\langle T, P^* \rangle\) of \(P\)

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

\(^\#\) outputs the same \(Y\) for any permutation of \(X\)
**STC - Scope**

STC supports a permutation-invariant algorithm \( P \) if there exists an imitator \( \langle T, P^* \rangle \) of \( P \)

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

Expressiveness ✔

# outputs the same \( Y \) for any permutation of \( X \)
**STC - A Closer Look**

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

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

Given $P$ operating on input $X$, $STC$ derives a privacy-preserving algorithm $A_P$:

1. $X' \leftarrow \text{Pre-Process}\ (X)\ \text{(if required)}$
   - ensure permutation-invariant requirement

2. $S \leftarrow \text{Scramble}\ (X')$
   - based on Melbourne Shuffle Algorithm

3. $Y' \leftarrow P(S)$

4. $Y \leftarrow \text{Post-Process}\ (Y)\ \text{(if required)}$
   - reverse effect of step 1

- Data Oblivious
- Requires private memory of size $O(\sqrt{n})$
- Runtime $O(n)$
**STC - A Closer Look**

Given $P$ operating on input $X$, STC derives a privacy-preserving algorithm $A_P$:

1. $X' \leftarrow \text{Pre-Process} (X)$ (if required)
   - ensure permutation-invariant requirement

2. $S \leftarrow \text{Scramble} (X')$
   - based on Melbourne Shuffle Algorithm

3. $Y' \leftarrow P(S)$

4. $Y \leftarrow \text{Post-Process} (Y)$ (if required)
   - reverse effect of step 1

---

**Low overhead ✔**

- Data Oblivious
- Requires private memory of size $O(\sqrt{n})$
- Runtime $O(n)$
**$STC$ - A Closer Look**

Given $P$ operating on input $X$, $STC$ derives a privacy-preserving algorithm $A_P$:

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

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

$X = 1 \; 3 \; 1 \; 4 \; 2 \; 4$
$STC$ - A Closer Look

Given $P$ operating on input $X$, $STC$ derives a privacy-preserving algorithm $A_P$:

1. $X' \leftarrow$ Pre-Process ($X$) (if required)

2. $S \leftarrow$ Scramble ($X'$)

3. $Y' \leftarrow P(S)$

4. $Y \leftarrow$ Post-Process ($Y$) (if required)

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

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

STC - A Closer Look

Privately scramble the input

The scrambling hide correspondences between records of $\mathcal{X}'$ and those of $\mathcal{S}$
**STC - A Closer Look**

Given $P$ operating on input $X$, STC derives a privacy-preserving algorithm $A_P$:

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

**Sort the scrambled input by external merge sort**

Observation made on $S$ cannot be linked back to that of $X'$. 
**STC - A Closer Look**

Given $P$ operating on input $X$, STC derives a privacy-preserving algorithm $A_P$:

1. $X' \leftarrow \text{Pre-Process } (X)$ (if required)
2. $S \leftarrow \text{Scramble } (X')$
3. $Y' \leftarrow P(S)$
4. $Y \leftarrow \text{Post-Process } (Y)$ (if required)
## Comparison with Alternative Solutions

<table>
<thead>
<tr>
<th></th>
<th>ORAM</th>
<th>STC</th>
<th>Tailor-made Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance Overhead</strong></td>
<td>$\Omega(\log n)$ amortized overhead <em>per each access</em></td>
<td>$O(n)$ additive overhead <em>per execution</em></td>
<td>less efficient than STC counterpart</td>
</tr>
<tr>
<td><strong>Expressiveness</strong></td>
<td>all applications</td>
<td>Spark and many data processing operations</td>
<td>application-specific</td>
</tr>
<tr>
<td><strong>Design and Implement Effort</strong></td>
<td>moderate - complicated</td>
<td>simple</td>
<td>complicated</td>
</tr>
</tbody>
</table>

PETS 2017
### Performance - Running time (s)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Baseline</th>
<th>(STC)</th>
<th>Tailor-made Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sort</strong></td>
<td>7,961</td>
<td>14,330</td>
<td>59,628 (7.49x)</td>
</tr>
<tr>
<td><strong>Compaction</strong></td>
<td>1,678</td>
<td>82,53</td>
<td>25,012 (14.89x)</td>
</tr>
<tr>
<td><strong>Select</strong></td>
<td>2,758</td>
<td>9,451</td>
<td>29,365 (16.65x)</td>
</tr>
<tr>
<td><strong>Aggregation</strong></td>
<td>10,593</td>
<td>24,578</td>
<td>63,477 (5.99x)</td>
</tr>
<tr>
<td><strong>Join</strong></td>
<td>12,400</td>
<td>59,610</td>
<td>105,235 (8.49x)</td>
</tr>
</tbody>
</table>

Input size: 32GB (i.e., \(2^{28}\) records)
Privacy-Preserving Computation with Trusted Computing via Scramble-then-Compute

Upto 4.1x speedups
Performance - Scalability

- **pSORT**
- **pCOMPACT**
- **pSELECT**
- **pAGGR**
- **pJOIN**

Support parallelism
Recaps

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 $Q_P(X)$ be the access patterns (i.e., sequence of read/write) the adversary observe during the execution of an algorithm $P$ on input $X$.

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

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 \textit{STC}

\textit{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.

PETS 2017
Melbourne Shuffle - Cleanup phase

Private Memory M

Cloud Storage

Read bucket

$\sqrt{n}$ buckets

$p \log n \sqrt{n}$

Remove dummies
Sort bucket

$O$

$\sqrt{n}$ buckets

$\sqrt{n}$

courtesy of Ohrimenko et al.