M^2R: Enabling Stronger Privacy in MapReduce Computation

Anh Dinh, Prateek Saxena,
Ee-Chien Chang, Beng Chin Ooi, Chunwang Zhang

School of Computing
National University of Singapore
1. Motivation

- Distributed computation (MapReduce) on large dataset with Trusted computing.
- Integrity + Confidentiality.
- Applicable in private or public cloud setting.
Challenge 1: Keep Trusted Code Base small

- Application Frameworks
- Operating Systems
- Hypervisor

CVEs in Linux [CVE-DB]

Affected many hypervisors (e.g. Xen / KVM) [CS Report]
Baseline System

- All data outside trusted environment is encrypted
- Software only attack.
- Previous system: VC³ (IEEE S&P 2015)
Background: MapReduce

- Computation & “shuffling” of <key, value> tuples.
- Phases: Map $\rightarrow$ Shuffle $\rightarrow$ Reduce.
- “map” outputs a set of tuples.
- During Shuffling, tuples are grouped according to their key.
- Each “reduce” instance corresponds to an unique key $k$. It takes all tuples with the key $k$ and output a set of tuples.
Background: Hadoop

• Hadoop: software framework written in Java
• ≈ 190K LOC  (Hadoop 0.21.0)
• Consists of MapReduce modules, Hadoop Distributed File System (HDFS), etc.
Is the Baseline System sufficient?

Identify small essential components of MapReduce to be included in the TCB.
Challenge 2: Interactions Leaks Info

M2R: Enabling Stronger Privacy in MapReduce Computation
Example of leakage: wordcount

- Map Phase: each mapT generates the tuples.
**Example of leakage: wordcount**

- **Shuffling Phase:** The tuples are grouped w.r.t the “words”.

  ![Diagram showing shuffling process]

- **Reduce Phase:** reduceT counts and outputs the number of tuples it received.
Example of leakage: word counts

- By observing the flow of tuples, one can infer relationships among the input files.

```
F1
  w1
  w2

mapT

shuffling

reduceT (key: w1)
```

```
F2
  w1
  w2
  w4

mapT

reduceT (key: w2)
reduceT (key: w3)
reduceT (key: w4)
```

```
F3
  w3
  w4

mapT

reduceT (key:w4)
```

F3 contains a unique word
Example of leakage: word counts

- By observing the flow of tuples, one can infer relationships among the input files.

Goal: hide these relationships.
Possible solution: Oblivious RAM

• Very high overhead.
2. Our solution

- Randomly permutes the tuples.
- Group the tuples according to their keys.
2. Our solution

- For execution integrity, addition step of verification is required.
2. Our solution

Randomly permutes the tuples

Original untrusted Shuffling process

Verifies grouping is correctly done

Secure Mixing (MixT)

Grouping

Verify groupings (GroupT)
Cascaded Mixing

A *cascaded mixing* is employed to randomly permute the tuples distributedly.
Remarks

• Key management, handling of the random nonce and initial value is not straightforward.

• In Hadoop, multiple reduce instances are carried out by a single *reducer*. Likewise *mapper*.
ORAM vs Our solution

- M$^2$R exploits the fact that, reads and writes can be “batched” into 2 phases, whereas ORAM caters for single read/write and thus incurs higher overhead.
- Many constructions of ORAM need to permute or o-sort the data.
3. Security Model

Adversary can observe the following:
• Input/output size of each trusted instance.
• Source/destination of the input/output.
• Time of invocation/return of each trusted instance.

Active adversary can:
• Arbitrary Invoke trusted instances.
• Halt instances.
• Drop/duplicate ciphertext (encrypted tuples).
• Add delays.
Modulo-$\Psi$ private

Based on formulation by Canetti (FOCS 01).

Let $\Psi$ be the permissible data that can be revealed during honest execution.

A provisioning protocol is modulo-$\Psi$ private if, for any adversary $A$ executing the protocol, there is an algorithm $B$ with access only to $\Psi$, such that the output of $A$ and $B$ are indistinguishable.
The permissible $\Psi$:

- size of input/output, time of revocation/return of $\text{mapT}$ and $\text{reduceT}$ under honest execution.

Goal: hide these relationships.
$M^2R$ is $\Psi$-modulo private.
4. Implementations & Experiments

• Use Xen-4.3.3 as the trusted hypervisor, and its Verifiable Dynamic Function Executor to load and execute trusted codes. (The design of M²R can be implemented differently depending on the underlying architecture, e.g. on Intel SGX).

• Ported 7 MapReduce benchmark applications.
  – KMeans : Iterative, Compute intensive
  – Grep : Compute intensive
  – Pagerank : Iterative, Compute intensive
  – WordCount : Shuffle intensive
  – Index : Shuffle intensive
  – Join : database queries
  – Aggregate : database queries

• 8 compute nodes, each quad-core Intel CPU 1.8 GHz, 8GB RAM, 1GB Ethernet cards.
Trusted Code Base

4 trusted computation units:
mixT, GroupT, mapT, reduceT.

• Platform related: (mixT, GroupT)
  Lines Of Code: ≈ 300
• Applications: (mapT, ReduceT)
  Lines Of Code ≈ 200 for our examples.
## Performance

<table>
<thead>
<tr>
<th>Job</th>
<th>Input size (bytes) (vs plaintext size)</th>
<th>Shuffled bytes</th>
<th>#Applications hyper calls</th>
<th>#platform hyper calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wordcount</td>
<td>2.1G (1.1×)</td>
<td>4.2G</td>
<td>3.3 × 10^6</td>
<td>35</td>
</tr>
<tr>
<td>Index</td>
<td>2.5G (1.2×)</td>
<td>8G</td>
<td>3.3 × 10^6</td>
<td>59</td>
</tr>
<tr>
<td>Grep</td>
<td>2.1G (1.1×)</td>
<td>75M</td>
<td>3.3 × 10^6</td>
<td>10</td>
</tr>
<tr>
<td>Aggregate</td>
<td>2.0G (1.2×)</td>
<td>289M</td>
<td>18.0 × 10^6</td>
<td>12</td>
</tr>
<tr>
<td>Join</td>
<td>2.0G (1.2×)</td>
<td>450M</td>
<td>11.0 × 10^6</td>
<td>14</td>
</tr>
<tr>
<td>Pagerank</td>
<td>2.5G (4.0×)</td>
<td>2.6G</td>
<td>1.7 × 10^6</td>
<td>21</td>
</tr>
<tr>
<td>KMeans</td>
<td>1.0G (1.1×)</td>
<td>11K</td>
<td>12.0 × 10^6</td>
<td>8</td>
</tr>
</tbody>
</table>
## Running time (s)

<table>
<thead>
<tr>
<th>Job</th>
<th>Baseline (vs no encryption)</th>
<th>M²R (vs baseline)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wordcount</td>
<td>570 (2.6 ×)</td>
<td>1156 (2.0 ×)</td>
</tr>
<tr>
<td>Index</td>
<td>666 (1.6 ×)</td>
<td>1549 (2.3 ×)</td>
</tr>
<tr>
<td>Grep</td>
<td>70 (1.5 ×)</td>
<td>106 (1.5 ×)</td>
</tr>
<tr>
<td>Aggregate</td>
<td>125 (1.6 ×)</td>
<td>205 (1.6 ×)</td>
</tr>
<tr>
<td>Join</td>
<td>422 (2 ×)</td>
<td>510 (1.2 ×)</td>
</tr>
<tr>
<td>Pagerank</td>
<td>521 (1.6 ×)</td>
<td>755 (1.4 ×)</td>
</tr>
<tr>
<td>KMeans</td>
<td>123 (1.7 ×)</td>
<td>145 (1.2 ×)</td>
</tr>
</tbody>
</table>
Conclusions

• Privacy-preserving distributed computation of MapReduce with trusted computing.

• Security:
  – Execution integrity + Data Confidentiality
  – Observation that simply running the map/reduce in trusted environment is not sufficient: interactions leak sensitive info.
  – Small TCB

• Exploit the algorithmic structure to outperform a solution that employs generic ORAM.

• Future works: other distributed dataflow systems.