PartialGC: A server-aided system for saving GC state

Benjamin Mood, Debayan Gupta, Kevin Butler, and Joan Feigenbaum

February 21, 2014
Privacy Preserving Data Analysis
Data Analysis

Privacy Preserving Data Analysis

Data Initialization
Data Analysis

Privacy Preserving Data Analysis

Data Initialization
Receive Report
Data Analysis

Privacy Preserving Data Analysis

Data Initialization
Receive Report
Update Information
Privacy Preserving Data Analysis

Data Analysis

Data Initialization  Receive Report  Update Information  Receive Report
Data Analysis

Privacy Preserving Data Analysis

Data Initialization  Receive Report  Update Information  Receive Report  Remove Data
Privacy Preserving Data Analysis
Data Analysis

Privacy Preserving Data Analysis

- Data Initialization
- Receive Report
- Update Information
- Receive Report
- Remove Data
- Modify Data
- Receive Report
Overall Idea:
Overall Idea:

Garbled Circuit 1

AND  XOR  NOT

Saved Values
Overall Idea:

- **Garbled Circuit 1**
  - AND
  - XOR
  - OR
  - NOT

- Transformation Protocol
- Saved Values
Overall Idea:

Garbled Circuit 1

Transformation Protocol

Saved Values

Garbled Circuit 2
Outline

- Transformation
- Checking Transformation
- Server Aided Protocol
- Results
Transforming Wires

• Generator creates garbled gates that transform the wires that work in one garbled circuit to wires that work in another garbled circuit.

Transformation Protocol
Transformation Details

* = once per circuit
Transformation Details

Encrypted Output Wires

Encrypted Input Wires

* = once per circuit
Transformation Details

* = once per circuit
Transformation Details

Generator

AND

AND

OR

* = once per circuit
Transformation Details

Generator

* = once per circuit
Transformation Details

Generator

AND  Output-0
     Output-1

AND

OR

* = once per circuit
Transformation Details

Generator

- AND
  - Output-0
  - Output-1
- AND
- OR

Random-0
Random-1

* = once per circuit
Transformation Details

Generator

nonce = PRNG.rand() *
Transform-0 = hash(Output-0 ⊕ nonce) ⊕ Random-0
Transform-1 = hash(Output-1 ⊕ nonce) ⊕ Random-1

* = once per circuit
nonce = PRNG.rand() *

\[
\text{Transform-0} = \text{hash}(\text{Output-0} \oplus \text{nonce}) \oplus \text{Random-0} \\
\text{Transform-1} = \text{hash}(\text{Output-1} \oplus \text{nonce}) \oplus \text{Random-1}
\]

* = once per circuit
Transformation Details

* = once per circuit
Evaluator

AND

AND

OR

* = once per circuit
Transformation Details

Evaluator

* = once per circuit
Evaluator

- AND
- Output-0

- AND
- nonce *
- Transform-0
- Transform-1

* = once per circuit
Evaluator

\[
\text{Random-0} = \text{hash}((\text{Output-0} \oplus \text{nonce}) \oplus \text{Transform-0})
\]

* = once per circuit
Evaluator

\[
\text{Random-0} = \text{hash(Output-0 } \oplus \text{ nonce) } \oplus \text{ Transform-0}
\]

\* = once per circuit
How to check?

- Evaluator can save the possible out values for a check circuit and upon receiving the next iteration of that check circuit can verify the transformation is correct.
How to check? cont.

• **Problem:**
  In our base protocol both parties know the check and evaluation split allowing the generator to only disrupt evaluation gates unless we commit.

• If we commit ahead of time we introduce other problems of longevity of the values.
• If the generator does not know the evaluation circuits from the check circuits, then he has to send correct transformation gates for all circuits.

• This also means the generator, for the entirety of the computation, can never learn the split.
Multiple Cut and Choose

Garbled Circuit 1

AND

XOR

OR

NOT

Check Circuit

Evaluation Circuit

Check Circuit

Evaluation Circuit

Check Circuit

Evaluation Circuit

Garbled Circuit 2

AND

XOR

OR

NOT

Evaluation Circuit

Check Circuit

Check Circuit

Check Circuit

Evaluation Circuit
Multiple Cut and Choose

Garbled Circuit 1

AND → XOR → OR → NOT

Check Circuit
Evaluation Circuit
Check Circuit
Evaluation Circuit
Check Circuit

Garbled Circuit 2

AND → XOR → OR → NOT

Evaluation Circuit
Check Circuit
Check Circuit
Check Circuit
Check Circuit
Evaluation Circuit
Single Cut and Choose

Garbled Circuit 1

AND
XOR
OR
NOT

Check Circuit
Evaluation Circuit
Check Circuit
Evaluation Circuit
Check Circuit

Garbled Circuit 2

AND
XOR
OR
NOT

Check Circuit
Evaluation Circuit
Check Circuit
Evaluation Circuit
Check Circuit

• For the cut and choose, we use OT to select encryption keys as implemented in [SS13].

• In the first computation perform the cut and choose.

• In any subsequent computation use the encryption keys to generate new encryption keys.
3 Circuit Example

Cut and Choose via OT
3 Circuit Example

Cut and Choose via OT
Checking Transformations

• Generator never learns the check/evaluation circuits
• Evaluator can check how the generator transforms values from one garbled circuit computation to another garbled circuit computation.
Implementation

- Server-aided setting
- [CMTB13] system: Outsources the evaluation of a garbled circuit from a mobile device to a high performance server (cloud) with security guarantees.
- Based on [KSS12]
CMTB

Generator

Cloud

Evaluator

Circuit Commit
Outsourced Oblivious transfer
CMTB

Generator

Cloud

Evaluator

Generator’s Input

Consistency Check
Cloud

Generator

Evaluator

Circuit Evaluation
CMTB

Generator

Cloud

Evaluator

Output and Output check
Outsourced PartialGC

Generator → Saved Values → Generator

Computation → Phone → Saved Values → Cloud

Phone → Computation → Phone

Cloud → Saved Values → Cloud
<table>
<thead>
<tr>
<th>Generator</th>
<th>Cloud</th>
<th>Phone</th>
</tr>
</thead>
</table>

Oregon Systems Infrastructure Research and Information Security (OSIRIS) Lab
Protocol

Generator | Cloud | Phone

Cut and Choose
Protocol

Generator | Cloud | Phone

Cut and Choose

OOT
Protocol

Generator | Cloud | Phone

Cut and Choose

OOT

Generator's input Consistency Check
Protocol

Generator -> Cloud -> Phone

- Cut and Choose
- OOT
- Generator's input Consistency Check
- Partial Generator -> Partial Evaluation
- Generation / Evaluation
- Output
 Protocol

Generator | Cloud | Phone

Cut and Choose

OOT

Generator's input Consistency Check

Partial Generator | Partial Evaluation

Generation / Evaluation

Output Check
Output Check

- Output \( (x || \text{MAC}(x)) \)

- Slower for circuit evaluation. Proof of concept implementation has \( \sim 14,000 \) non-XOR gates per 128 bits.

- Extremely fast for our outsourcing party [bits/128 MAC operations instead of the output proof with homomorphic XOR commitments].
Wrong Saved Values

- Generator gets caught through circuit check
- Cloud gets caught, assuming he continues in the computation, when the output check fails
Incorrect Check Circuits

• Aborting on incorrect check circuits gives away information about what circuits are check or evaluation.

• If check is found to be incorrect, then the remaining computation and any saved values must be abandoned.

• Cloud informs the Generator and Phone of the incorrect circuit and what it should have been.

Check Circuit
# Preliminary Test Results

<table>
<thead>
<tr>
<th></th>
<th>64 Circuits</th>
<th>256 Circuits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CMTB</td>
<td>Partial</td>
</tr>
<tr>
<td>KeyedDB 64</td>
<td>72 ± 2%</td>
<td>8.3 ± 5%</td>
</tr>
<tr>
<td>KeyedDB 128</td>
<td>140 ± 2%</td>
<td>9.5 ± 4%</td>
</tr>
<tr>
<td>KeyedDB 256</td>
<td>270 ± 1%</td>
<td>12 ± 6%</td>
</tr>
</tbody>
</table>

* both evaluated on same hardware, security parameters, and setup
Work in progress

<table>
<thead>
<tr>
<th></th>
<th>64 Circuits</th>
<th>256 Circuits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CMTB</td>
<td>Partial</td>
</tr>
<tr>
<td>Largest Substring 128</td>
<td>190 ± 4%</td>
<td>20 ± 9%</td>
</tr>
<tr>
<td>Largest Substring 256</td>
<td>370 ± 4%</td>
<td>40 ± 10%</td>
</tr>
<tr>
<td>Largest Substring 512</td>
<td>730 ± 4%</td>
<td>70 ± 10%</td>
</tr>
</tbody>
</table>

- For comparison

-- In [CMTB] output and input values under a 1-time pad with MACs.
Conclusion

• Saving wire labels
• Transform and check values
• Discussed our protocol and preliminary results
• Work in progress ...
Questions?

Benjamin Mood, bmood@cs.uoregon.edu

http://osiris.cs.uoregon.edu/
Questions?

Benjamin Mood, bmood@cs.uoregon.edu

http://osiris.cs.uoregon.edu/