Scalable Secure Multi-Party Network Vulnerability Analysis via Symbolic Optimization

Privacy Preserving Measurements Aggregation and Analysis

Kinan Dak Albab, Rawane Issa, Andrei Lapets, Azer Bestavors, Nikolaj Volgushev.
Why Do We Need Privacy Preserving Measurement Aggregation and Analysis?
Example
Simple “In The Open” Algorithm

Algorithm 1 Network Distance

1: $i \leftarrow 0$
2: while $i < \text{iterations}$ do
3: for $n \in V$ do
4: $D_{i+1}[n] \leftarrow \min(D_i[n], \min_{v \in \text{nb}(n)} (D_i[v] + 1))$
5: $i \leftarrow i + 1$

$O(\text{diameter} \times n \times \text{degree})$
Security, Measurements, And Privacy

- Measurements are network-whole: often require access to or express information about the entirety of the network.
- Security is network-whole: security analysis requires analysis of the entire network.
Security, Measurements, And Privacy

- Measurements and security may conflict with privacy:
  - May require access to private information.
  - May reveal private information in the process of measuring.

- Parties are interested in keeping their private information private.

- Parties are interested in collecting and analysing measurements and networks to increase security.

- Parties have a need for privacy-preserving approaches to measurements and security analysis.

You shall have both!
What is Secure Multi-Party Computation?
Secure Multi-party Computation

- A Cryptographic framework introduced in the 80s [Yao82]
- Allows computation and aggregation of private information.
- Provable guarantees that the inputs remain secret.
- Only reveals the final output, which may be different for every party.
- Only leaks information deducible from the output.
Shamir’s Secret Sharing

- A scheme for sharing a secret/input [Shamir79].
- A secret/input is divided into n parts (shares) - one part for every party.
- Some or all shares are required to reconstruct the secret.
Secure Multi-party Computation

- Share -> operate on received share -> combine resulting shares.
- Operations that are constant “in the open” are not constant in MPC.
- Addition: relatively fast, requires two communication rounds (sharing and reconstruction).
- Multiplication: expensive, number of rounds depends on the number of parties.
- Multiplication is much more expensive than addition!
MPC in Practice

- Recent work on making MPC better:
  - Efficient protocols for specific problems.
  - High-level Frameworks/libraries [VIFF][ShareMind][Sepia].
  - Deployments and applications [Best17] [Lapets16].

- (Out of the box) MPC remains very slow!
## MPC in Practice

<table>
<thead>
<tr>
<th>P</th>
<th>Node</th>
<th>Edge</th>
<th>MPC(^1)</th>
<th>Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>32378</td>
<td>67218</td>
<td>&gt; 24hrs</td>
<td>1.7s</td>
</tr>
<tr>
<td>3</td>
<td>32378</td>
<td>67218</td>
<td>&gt; 24hrs</td>
<td>2s</td>
</tr>
<tr>
<td>4</td>
<td>43510</td>
<td>89783</td>
<td>&gt; 24hrs</td>
<td>2.8s</td>
</tr>
<tr>
<td>4</td>
<td>43510</td>
<td>89783</td>
<td>&gt; 24hrs</td>
<td>2.8s</td>
</tr>
<tr>
<td>5</td>
<td>55093</td>
<td>156773</td>
<td>Rec. Limit(^2)</td>
<td>5.8s</td>
</tr>
<tr>
<td>5</td>
<td>55093</td>
<td>156773</td>
<td>Rec. Limit(^2)</td>
<td>5.9s</td>
</tr>
<tr>
<td>10</td>
<td>108788</td>
<td>250800</td>
<td>-</td>
<td>19.4s</td>
</tr>
</tbody>
</table>

\(^1\) Direct implementation using VIFF and python.

\(^2\) Recursion Limit (10000) was reached.
Our Approach
Approach

- Reduce the size of inputs into MPC.
  - Three stages:
    - Local -> MPC -> Local
  - Perform MPC on the public gateways network.
- Reduce the number of expensive operations in MPC (multiplication and comparisons).
  - Unroll the MPC stage into symbolic expressions.
  - Optimize the expressions to reduce expensive operations.
Input
Global View
Local Network
Company 1
Local Computation Output
Company 1
Gateways Graph
Global View
Gateways Graph
Company 1
MPC Output
Company 1
Final Output
Company 1
Final Output
Global View
Optimizing The MPC Stage
**MPC Stage**

- The MPC stage runs on the public gateway network.
  - Public Connections between gateways of different parties with weight 1.
  - Private Connections between gateways of the same party with private weights (from first local stage).
- We must minimize distances over connections with weight 1 and connections with private weights.
Example Run

\[
\text{Min}(A, B+1, C+1)
\]

\[
\text{Min}( \text{Min}(A, B+1, C+1), \\
\text{Min}(B, D+1, C+1, A+1) + 1, \\
\text{Min}(C, D+1, B+1, A+1) + 1 \\
) 
\]

\[
\text{Min}( \text{Min}( \text{...}), \\
\text{Min}( \text{Min}(\text{...}), \text{Min}(D, B+1, C+1) +1, \text{...}) +1, \\
\text{Min}( \text{Min}(\text{...}), \\
\text{Min}(C, B+1, D+1, F+1) +1, \text{...})+1, \\
\text{...} \\
) 
\]
Symbolic Expressions

- Expressions contain a lot of repetitions and redundant terms.
- Expressions contain terms that cannot be the minimum.
- Running the algorithm in MPC directly will compute these terms.
- Optimize the expression and reduce its size before evaluating it in MPC.
**Plus-Min Reduction**
Min-Min Reduction
Early-Min Reduction
Optimizing The MPC Stage

- The optimized expressions are guaranteed to be a single Min.
- This Min has at most as many terms as gateways.
- Evaluating these expressions is equivalent to evaluate the portion of the algorithm that runs on the public connections.
Optimizing The MPC Stage (2)

- Alternate between:
  - Evaluating the expression for every gateway to propagate distances across parties.
  - Computing the minimum distance for every gateway over gateways of the same party to propagate distances within the same party.
- Repeat for as many parties as we have
  - May need less iterations if parties are highly connected and diameter is small.
Implementation
Implementation

- We implemented our approach in Python using VIFF.
- The implementation includes a library for expression manipulation and MPC.
- Expression Optimizers and Evaluators are implemented as separate classes.
- Any other MPC framework can also be used by implementing an Evaluator class for it.
- Client code does not need to reveal details about expressions or MPC.
- Doing certain reductions explicitly can improve performance.
- https://github.com/hicsail/ExpressionMPC
## Benchmarks

<table>
<thead>
<tr>
<th>P</th>
<th>Node</th>
<th>Edge</th>
<th>Gateway</th>
<th>Pub. Edg.</th>
<th>Our Method</th>
<th>MPC¹</th>
<th>Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>32378</td>
<td>67218</td>
<td>34</td>
<td>86</td>
<td>0.72min</td>
<td>&gt; 24hrs</td>
<td>1.7s</td>
</tr>
<tr>
<td>3</td>
<td>32378</td>
<td>67218</td>
<td>220</td>
<td>579</td>
<td>62min</td>
<td>&gt; 24hrs</td>
<td>2s</td>
</tr>
<tr>
<td>4</td>
<td>43510</td>
<td>89783</td>
<td>43</td>
<td>105</td>
<td>2.75min</td>
<td>&gt; 24hrs</td>
<td>2.8s</td>
</tr>
<tr>
<td>4</td>
<td>43510</td>
<td>89783</td>
<td>301</td>
<td>850</td>
<td>72min</td>
<td>&gt; 24hrs</td>
<td>2.8s</td>
</tr>
<tr>
<td>5</td>
<td>55093</td>
<td>156773</td>
<td>45</td>
<td>105</td>
<td>2min</td>
<td>Rec. Limit²</td>
<td>5.8s</td>
</tr>
<tr>
<td>5</td>
<td>55093</td>
<td>156773</td>
<td>393</td>
<td>981</td>
<td>154min</td>
<td>Rec. Limit²</td>
<td>5.9s</td>
</tr>
<tr>
<td>10</td>
<td>108788</td>
<td>250800</td>
<td>44</td>
<td>124</td>
<td>3min</td>
<td>-</td>
<td>19.4s</td>
</tr>
</tbody>
</table>

Networks representing autonomous systems peering information from the Stanford large network dataset collection.
Future Work

- Apply the techniques to more problems (Flow-networks) (Geospatial algorithms).
- Include conditionals, loops, and other python statements.
- Speed-up evaluation by exploiting similar terms (memoization).
Summary

- MPC gives us privacy-preserving aggregation and analysis.
- We can perform a global vulnerability analysis without knowing/revealing information about any private sub-network.
- Two techniques for optimizing MPC:
  - Do as much local computation as possible before doing MPC.
  - Reduce the number of expensive operations in MPC.
References


Acknowledgments

- Co-authors: Rawane Issa, Andrei Lapets, Azer Bestavors, Nikolaj Volgushev.
- Mayank Varia and Ran Canetti for feedback and discussions.