On the Performance of Secure Graph Algorithms in Detecting Routing Loops

Sucheer Maddury[1†], Jaber Daneshamooz[2]
[1]Leland High School, 6677 Camden Ave, San Jose, CA 95120
[2]Department of Computer Science, University of California, Santa Barbara, CA 93106
[†]corresponding author: sumaddurycollege2024@gmail.com

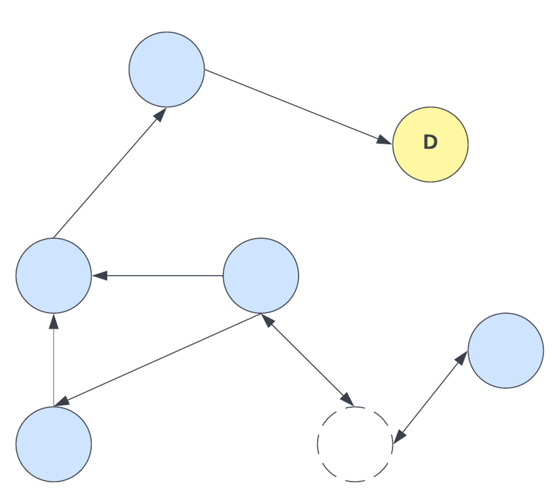
UC SANTA BARBARA
Research Mentorship Program

PROJECT DESCRIPTION

- **Network backbone cycles** cost billions of dollars each year and cause outages with humanitarian costs in authoritarian countries
- Detecting cycles requires graph algorithms, however they must be modified with **secret sharing** for **security of the node addresses**
- In this work, we review a variety of multi-party computation frameworks and choose the TNO-MPC framework due to its variety of arithmetic and comparison operators and high party support
- We also utilize TinySMPC for an educational and comparison purpose
- We then implement **MPC-powered breadth first search** (BFS) via additive secret sharing for **directed cycle detection** with destination node
- Afterwards, we gather **six internet topology datasets** from CAIDA's Internet Topology Data Kit for benchmarking of the BFS algorithm
- We **benchmark the BFS algorithm** with 10 runs per dataset and compare the execution time with graph size and party count
- We find that the BFS follows the expected $O(V+E)$ time complexity

MOTIVATION & BACKGROUND

NETWORK VERIFICATION

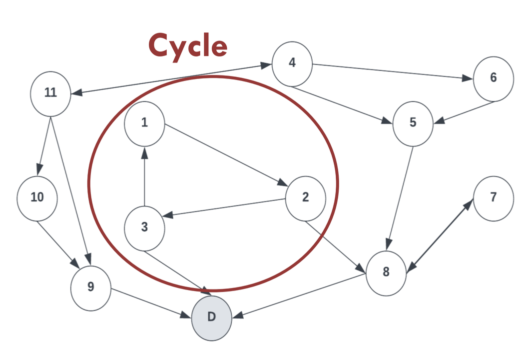


The Forwarding Graph

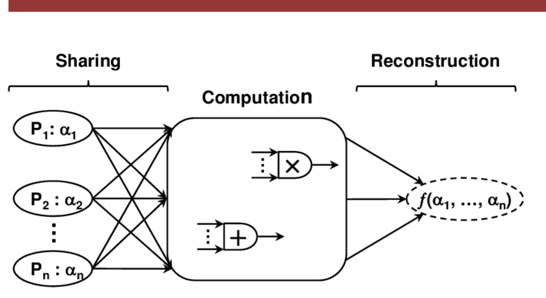
- Characterized by direct and indirect links
- Each node represents a separate computer (autonomous system)
- All nodes must have a path to the destination node
- Represented by edge-list

Routing cycles and other misconfigurations cause internet outages

- 504.4 million affected severely in 2021
- 73% of outages allowed human rights abuses



FOUNDATIONAL MPC METHODS



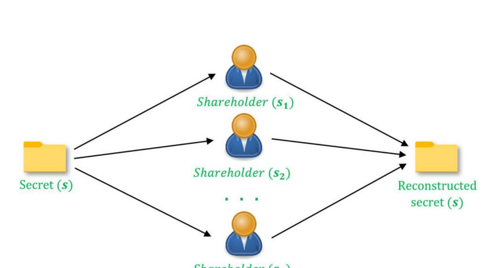
Nojoumian (2012)

Multi-Party Computation

- Allows several parties to jointly compute a function
- Enables security by ensuring no party has enough information independently for reconstruction

Shamir's Secret Sharing [1]

- One of the first secure MPC protocols with addition
- Functions by distributing summing shares that can be reconstructed to the original scalar
- Useful for securely sharing integers



GeeksforGeeks (2021)

$$[d]_i = [x]_i - [a]_i$$
$$[e]_i = [y]_i - [b]_i$$

$$[z]_i = de + d[b]_i + e[a]_i + [c]_i$$
$$z = de + db + ea + c = xy$$

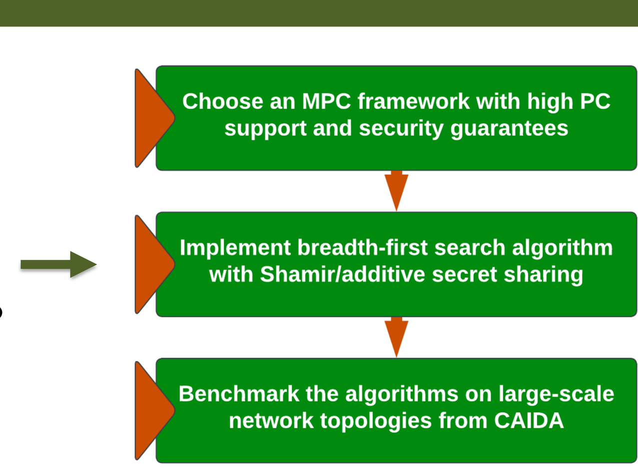
Beaver's Secret Sharing [2]

- Used multiplication triples for secure multiplication with n-parties
- Multiplication triples have become basis for modern MPC arithmetic

CHALLENGES & GOALS

Challenges

- Network administrators are reluctant to expose BGP configuration information
- Algorithms may function on smaller examples, but fail to scale to large networks
- Cycle detection may fail on directed cycles



BUILDING THE SECURE BREADTH-FIRST SEARCH ALGORITHM

CHOOSING AN MPC FRAMEWORK

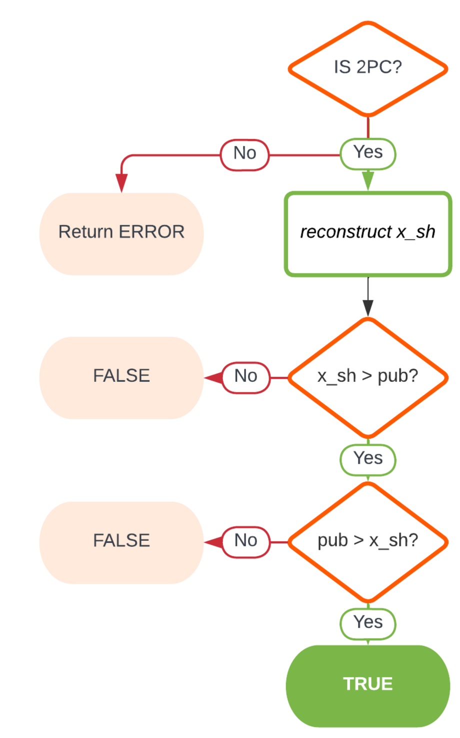
- We filtered all frameworks in Rotaru's awesome-MPC list
- We reviewed frameworks with >2PC support
- We noted their corrupted party type, PC, purpose, and operations offered
- On the side, we worked with TinySMPC [3] for small-scale simulations

TNO-MPC [4]

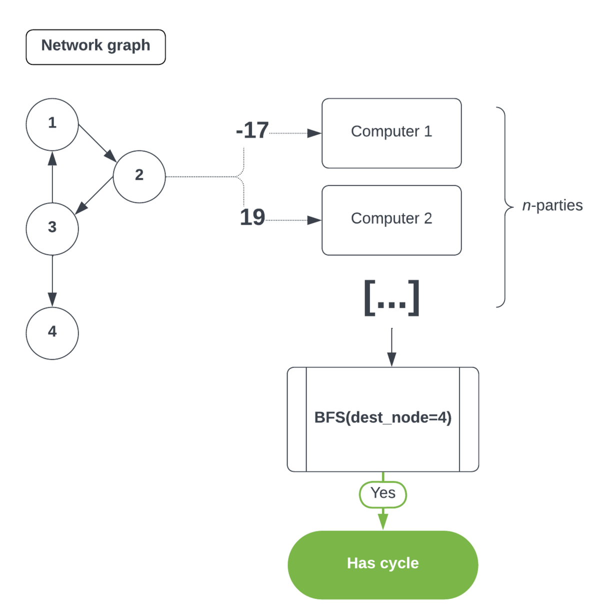
- ✓ Comprehensive documentation
- ✓ Wide array of arithmetic operators
- ✓ Extensive Shamir SS support
- ✓ Tested on 3PC, supports others
- ✓ Guarantees MPC privacy

EQUALITY CHECK

- Equality check is necessary to understand the identity of destination node
- TinySMPC only supports ">" out of the box
- Functions by checking whether the shared scalar is larger than the public integer and vice versa via *private compare*
- If both are false, the shared scalar = public
- Only works with 2PC due to limitation on TinySMPC's *private compare*



IMPLEMENTING SECURE BFS



- Create visited, in-progress, and to-be-visited sets
- Add dest_node to queue and in-progress set
- Continually look at front node in queue and search for a neighboring node that is in-progress → **cycle exists**
- If there are neighboring nodes not in the visited set, add them to the queue and in-processing
- Queue empties → **no cycle**

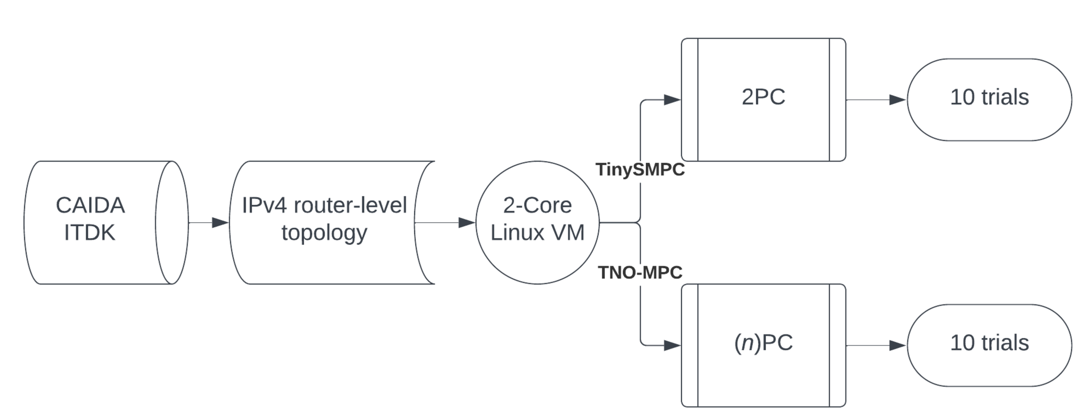
DATA ACQUISITION

- Data was taken from CAIDA's ITDK [5]
- All datasets used contained at least one directed cycle
- Only direct edges used
- Datasets had varying amounts of V+E

Dataset	Node count	Edge count
Team 1 2010	16695	33315
Team 1 2018	25194	54086
Team 1 2019	27895	55675
Team 1 2020	34796	77650
Team 2 2018	24868	53910
Team 3 2018	25059	54375

BENCHMARKING RESULTS

BENCHMARKING PROCEDURE

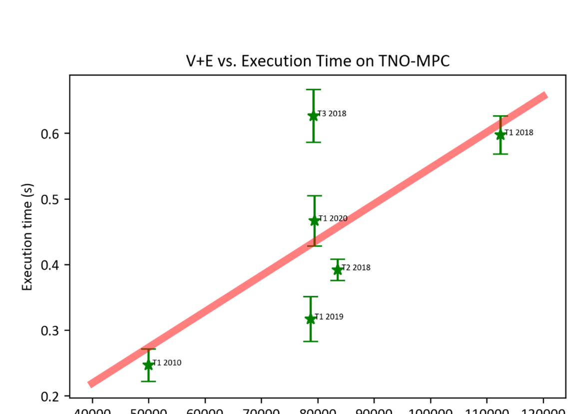


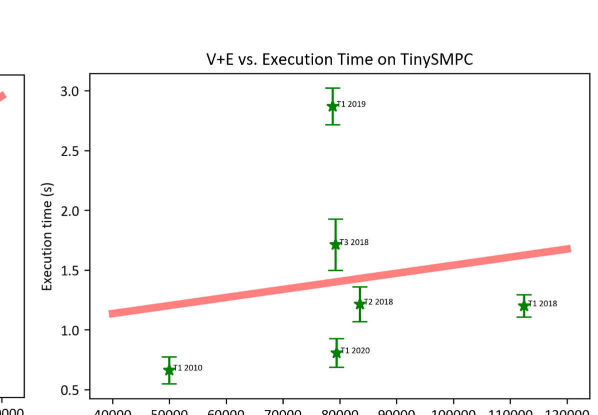
Datasets taken from CAIDA ITDK were preprocessed and fed in edge list format into the secure BFS algorithm for 10 trials on each dataset.

V+E VS. EXECUTION TIME

- In order to get some sense of scalability, we used datasets with a wide range of V+E, and ran BFS for 10 runs on each dataset
- The results averaged and had statistical analysis performed

Dataset	V+E	TNO-MPC time (s)	TinySMPC time (s)
Team 1 2010	50010	0.247 ± 0.0249	0.661 ± 0.113
Team 1 2018	83570	0.392 ± 0.0164	1.213 ± 0.145
Team 1 2019	112446	0.597 ± 0.0291	1.199 ± 0.0934
Team 1 2020	79280	0.626 ± 0.0402	1.711 ± 0.214
Team 2 2018	78778	0.317 ± 0.0343	2.868 ± 0.152
Team 3 2018	79434	0.466 ± 0.0383	0.806 ± 0.120





- All plots were created via Matplotlib [6]
- Via SciPy [7], we calculated statistical significance of the V+E vs execution time relationship with p-value:

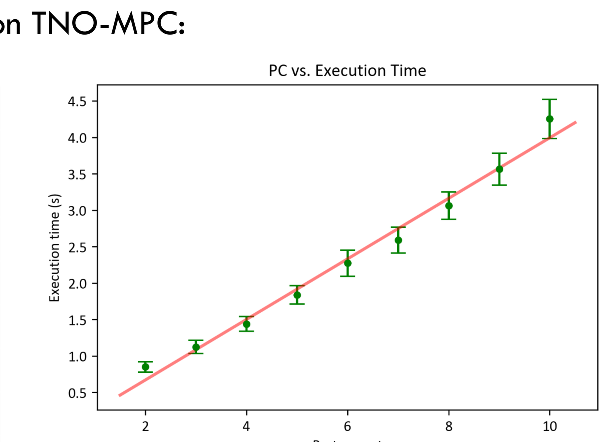
TNO-MPC p-value: 1.65383×10^-6 (****)

TinySMPC p-value: 1.65401×10^-6 (****)

V+E VS. EXECUTION TIME

To assess privacy potential, we also analyzed the relationship between PC and the execution time of the BFS on TNO-MPC:

PC	Time mean (s)	2 SE(s)
2	0.849	0.0691
3	1.124	0.0903
4	1.439	0.101
5	1.837	0.126
6	2.273	0.180
7	2.589	0.176
8	3.0606	0.187
9	3.564	0.219
10	4.252	0.267



p-value: 1.91515×10^-3 (**)

*Note: TinySMPC could have been additionally tested on secretly shared destination node, however the poor performance and erratic scalability caused us to disregard TinySMPC when testing PC vs. execution time.

DISCUSSION

- In terms of privacy, our BFS implementation is successful in keeping secret the true node addresses to the computing parties
- The node order will also remain secure during the MPC process
- However, a corrupted party may be able to gain some information about the general structure of the network backbone
- All relationships found are statistically significant, and the V+E vs. execution time relationship mostly follows $O(V+E)$
- The BFS execution time is fast enough to be used for new network backbones or large network changes
- However, it is not fast enough for usage in checking every routing change, as these happen at extremely high rates

FUTURE WORK

- In the future, our BFS implementation should be optimized via techniques such as next-hop traversal in order to only analyze the change area
- The algorithm should also be integrated with a larger network verification system, which takes an input of internet backbone
- It may also be useful to implement higher specificity in the BFS algorithm, i.e., specifying where the cycle exists in the network backbone

Acknowledgements: The first author would like to thank Jaber Daneshamooz for his crucial continued guidance and facilitation throughout this project. The first author would also like to thank Melody Yu for her contributions to the CAIDA data acquisition method. Finally, the first author would like to thank Dr. Lina Kim, Zheng Ke, and the Research Mentorship Program for their underlying support in enabling this research pursuit.

References:

1. A. Shamir, 'How to Share a Secret', Commun. ACM, vol. 22, no. 11, pp. 612–613, Nov. 1979.
2. D. Beaver, 'Efficient Multiparty Protocols Using Circuit Randomization', in Advances in Cryptology --- CRYPTO '91, 1992, pp. 420–432.
3. K. Song, "TinySMPC: A tiny, Educational Library for secure multi-party computation (in pure python).," GitHub, https://github.com/kennysong/tinysmpc.
4. T. Rooijakkers, B. Kamphorst, and M. S, "TNO-MPC Lab," GitHub, https://github.com/TNO-MPC/.
5. The CAIDA Macroscopic Internet Topology Data Kit, 2023, https://www.caida.org/catalog/datasets/internet-topology-data-kit
6. J. D. Hunter, 'Matplotlib: A 2D graphics environment', Computing in Science & Engineering, vol. 9, no. 3, pp. 90–95, 2007.
7. P. Virtanen et al., 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python', Nature Methods, vol. 17, pp. 261–272, 2020.