SECURE MULTIPARTY PAGE RANK ALGORITHM FOR FRAUD DETECTION

Alex Sangers, Maran van Heesch, Thomas Attema, Thijs Veugen, Mark Wiggerman, Jan Veldsink, Oscar Bloemen, Daniël Worm

Thomas.Attema@tno.nl
INTRODUCTION

- Dutch banks are obligated to take measures against criminal activities, such as
  - Fraud
  - Money laundering

- Relevant information to detect these activities is distributed amongst banks

- But also,
  - Privacy sensitive (e.g. GDPR) and confidential

- Can MPC solve this problem?
Conclusion:

Analysing transaction graphs reduces false positives of existing fraud detection algorithms
TRANSACTIONS REPRESENTED IN A GRAPH

Nodes: Bank accounts

Links: Transactions

Additional properties per link:
- Number of transactions
- Total amount of money transferred
- ...

For a given time interval
TRANSACTION GRAPHS PROVIDE VALUABLE INFORMATION FOR FRAUD DETECTION

Metrics that can be used for fraud detection

- **Shortest path**
  - Long path → fraudulent ++

- **Strongly connected components**
  - Different components → fraudulent ++

- **PageRank**
  - Low PageRank → fraudulent ++

Focus of this research: how to securely, collaboratively compute PageRank on coupled transaction graphs?
COLLABORATION IS REQUIRED TO ANALYSE THE COMBINED TRANSACTION NETWORK

Party A only sees the blue transactions:

Three different parties: A, B, C
PAGERANK IS A CENTRALITY MEASURE

- Developed by the founders of google Larry Page and Sergey Brin (1998)
  - To order webpages based on some measure of importance

- Imagine a dollar going from bank account to bank account, either
  - Following edges ($p$)
  - Teleporting ($1 - p$)

- The proportion of time spent in each bank account is the PageRank value

- Markov process
  - PageRank values are the steady state probabilities
PAGERANK IS A CENTRALITY MEASURE
COMPUTATION OF PAGERANK (1/2)

› Iterative algorithm

\[ x^j_k = \text{Probability the dollar is at node } j \text{ after iteration } k \]

› If \( p < 1 \)

› \( x^j_k \) converges to the PageRank values for \( j \to \infty \)

› Convergence rate is \( p \)

› Under mild conditions on the graph
PageRank is iteratively computed, for each node j:

\[ x_{k+1}^j = \frac{1 - p}{n} + p \cdot \sum_{i \in S(j)} \frac{x_k^i}{c_i} \]

where \( p \in [0.85, 0.99] \),

- \( n \) is the number of nodes,
- \( S(j) \) is set of nodes linking to node \( j \),
- \( c_i \) is the outdegree of node \( i \).

Example: red bank wants to update node 4 at iteration \( k + 1 \):

\[ S(4) = \{6, 2\}, \quad x_{k+1}^4 = \frac{1 - p}{n} + p \cdot \left( \frac{x_k^6}{c_6} + \frac{x_k^2}{c_2} \right) \]

Known by blue bank, but secret to red bank.
PAGERANK INPUT

› Public input:
  › Number of nodes $n$
  › PageRank probability $p$

› Private input per node $i$ (bank account)
  › Out degree $c_i$
  › Incoming nodes $S(i)$

› Output
  › PageRank values
In every multiplication at least one of the factors is known by at least one of the parties

- Only additively homomorphic encryption required

\[ x_{k+1} = \frac{1 - p}{n} + p \cdot \sum_{i \in S(j)} \frac{x_i}{c_i} \]
MPC SOLUTION EXPLOITING THE LINEARITY OF THE PAGERANK ALGORITHM

In every multiplication at least one of the factors is known by at least one of the parties
- Only additively homomorphic encryption required

Apply the Paillier encryption scheme

\[
\left[x_{k+1}^j\right] = \frac{1 - p}{n} + p \cdot \sum_{i \in S(j)} \frac{1}{c_i} \cdot [x_i^k]
\]
PROTOCOL: ONLY LITTLE DATA COMMUNICATION

Key generation:
- Centralized. Provide a public key and partial private keys.

Initialization:
- Collaboratively compute $n$ (number of nodes).

Secure PageRank:
- Only share encrypted PageRank contributions and values at each PageRank iteration.

Partial Decrypt & Combine:
- Collaboratively decrypt the PageRank values with partial decryptions.

Distributed KeyGen implemented
**RESULTS**

- Number of rounds is the number of PageRank iterations (~50)
- Communication per round
  - 4098 bit ciphertexts
  - $O((n - 1) \cdot \#nodes)$
  - 3 parties and 4000 nodes per party: ~12MB per round
- Computational complexity
  - 3 parties running on 1 machine (1 CPU p.p.)
  - Python implementation
NEXT STEPS

- Improve scalability
  - E.g. C-implementation

- Extend this approach to other graph algorithms
  - Shortest path problem
  - Max-flow
  - Connected components

- Active security
  - Now: passive security with honest majority
    - Currently honest majority required in the distributed key generation

- Publish distributed key generation implementation (github)
THANK YOU FOR YOUR ATTENTION

Take a look:
TNO.NL/TNO-INSIGHTS