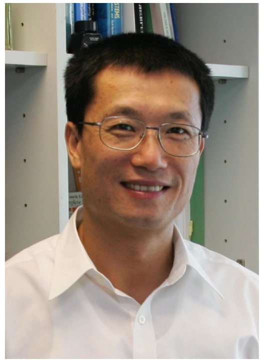
33rd USENIX Security Symposium

GraphGuard: Private Time-Constrained Pattern Detection Over Streaming Graphs in the Cloud



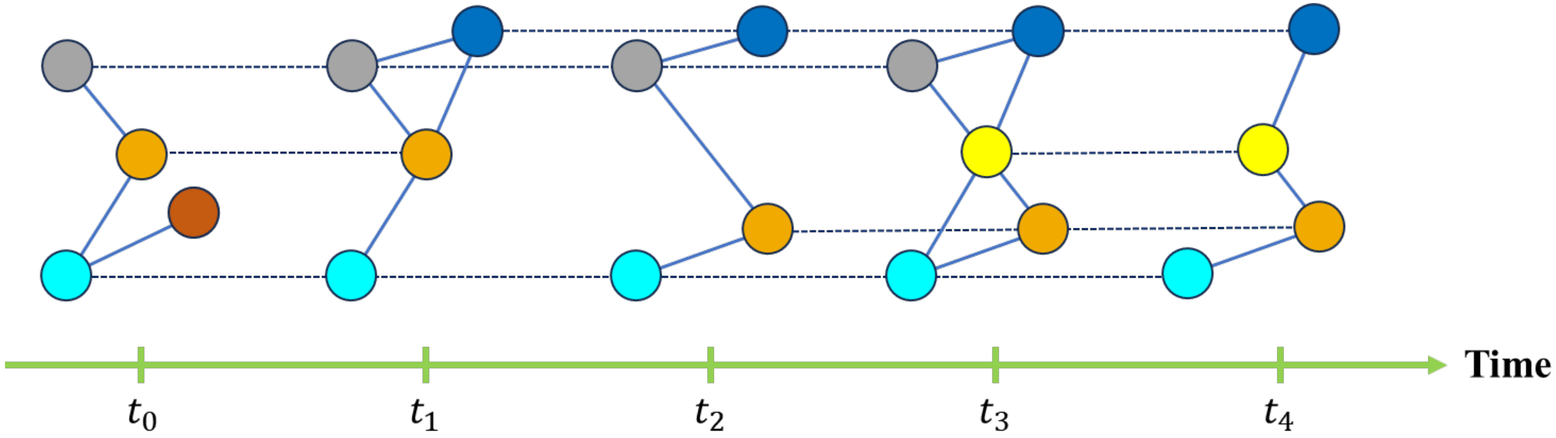
Songlei Wang Yifeng Zheng Xiaohua Jia

哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

CityU
香港城市大學
City University of Hong Kong

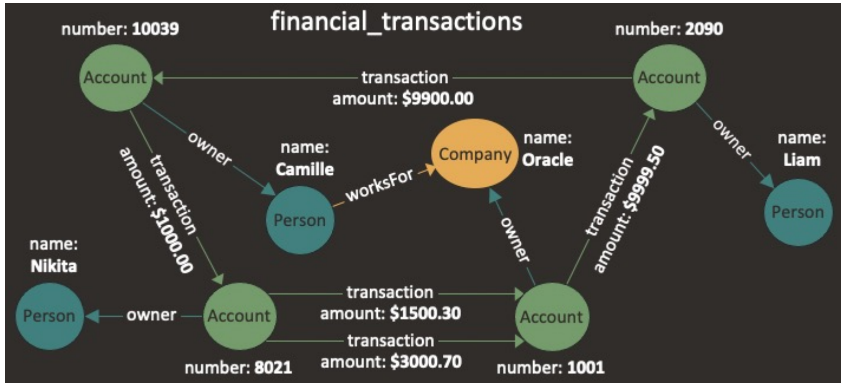
Streaming graphs

- Streaming graphs: vertices and edges change over time



- Widely seen in many practical applications

Social media

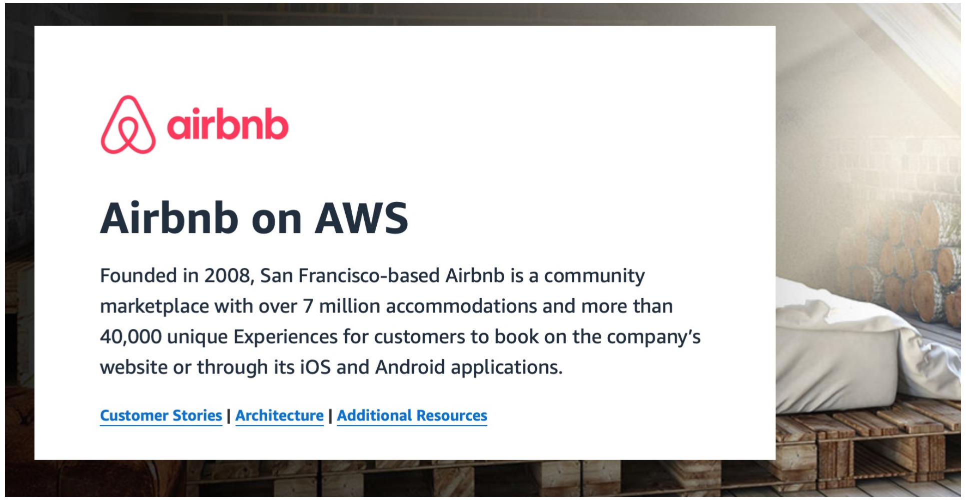


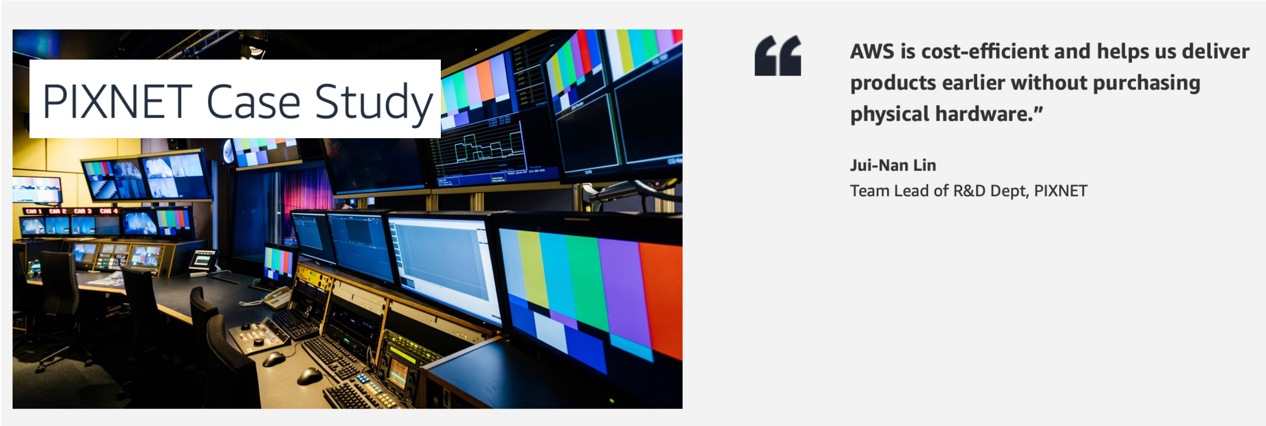
Financial networks

Computer networks

Storing and querying graphs in the cloud is popular

- Harness the benefits of cloud computing like cost efficiency, scalability, ubiqiutous access, etc.

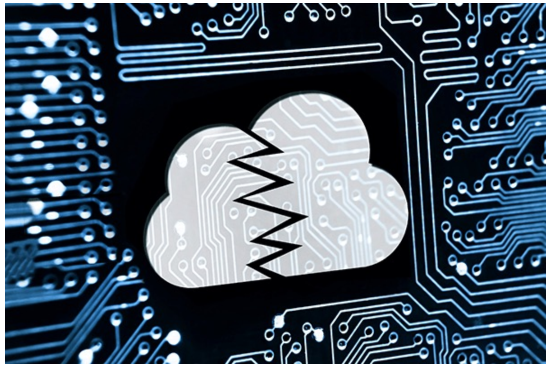




Concerns on data privacy

- Streaming graphs contain rich information
  - Might be privacy-sensitive (e.g., personal connections) or proprietary to the graph owner

- Cloud data breaches happen from time to time
  - E.g., 39% of businesses faced a cloud environment data breach last year [2023 Thales Cloud Security Report]

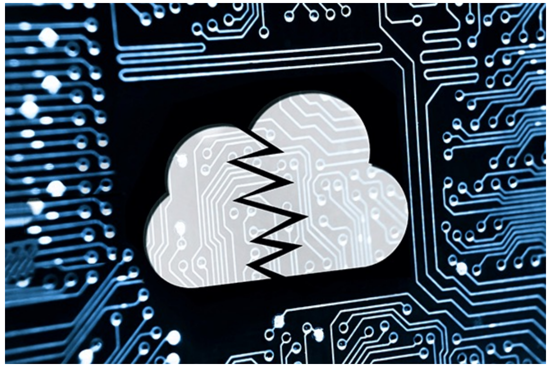


[2023 Thales Cloud Security Report]https://cpl.thalesgroup.com/about-us/newsroom/2023-cloud-security-cyberattacks-data-breaches-press-release

Concerns on data privacy

- Streaming graphs contain rich information
  - Might be privacy-sensitive (e.g., personal connections) or proprietary to the graph owner

- Cloud data breaches happen from time to time
  - E.g., 39% of businesses faced a cloud environment data breach last year
    [2023 Thales Cloud Security Report]



Essential to secure the outsourced streaming graphs and queries!

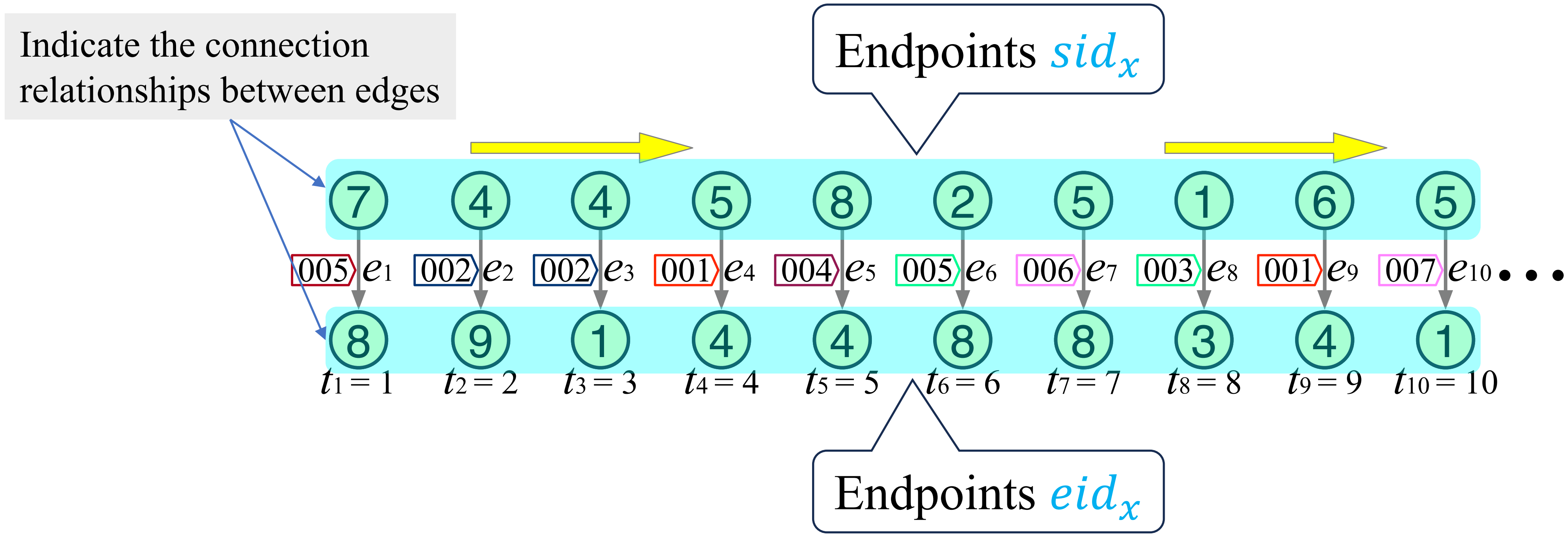
[2023 Thales Cloud Security Report]https://cpl.thalesgroup.com/about-us/newsroom/2023-cloud-security-cyberattacks-data-breaches-press-release

Our focus: time-constrained graph pattern detection

- Aim: continuously detect subgraphs that match a given query pattern
  - Important for applications like credit card fraud detection [Qiu et al., VLDB'18] and cyber-attack detection [Choudhury et al., EDBT'15]

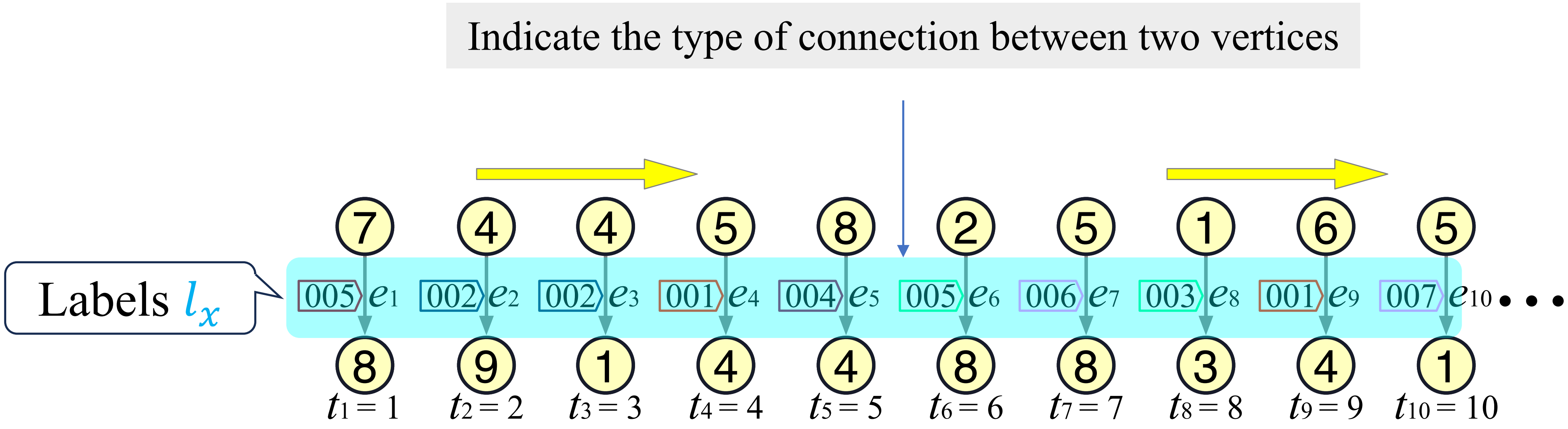
- What makes a "match"?
  - Structure is matched, i.e., isomorphism
  - The labels of edges are matched
  - Edge timing order matching, i.e., edge occurrence orders adhere to the timing order constraints specified by the query pattern

Example of time-constrained graph pattern detection



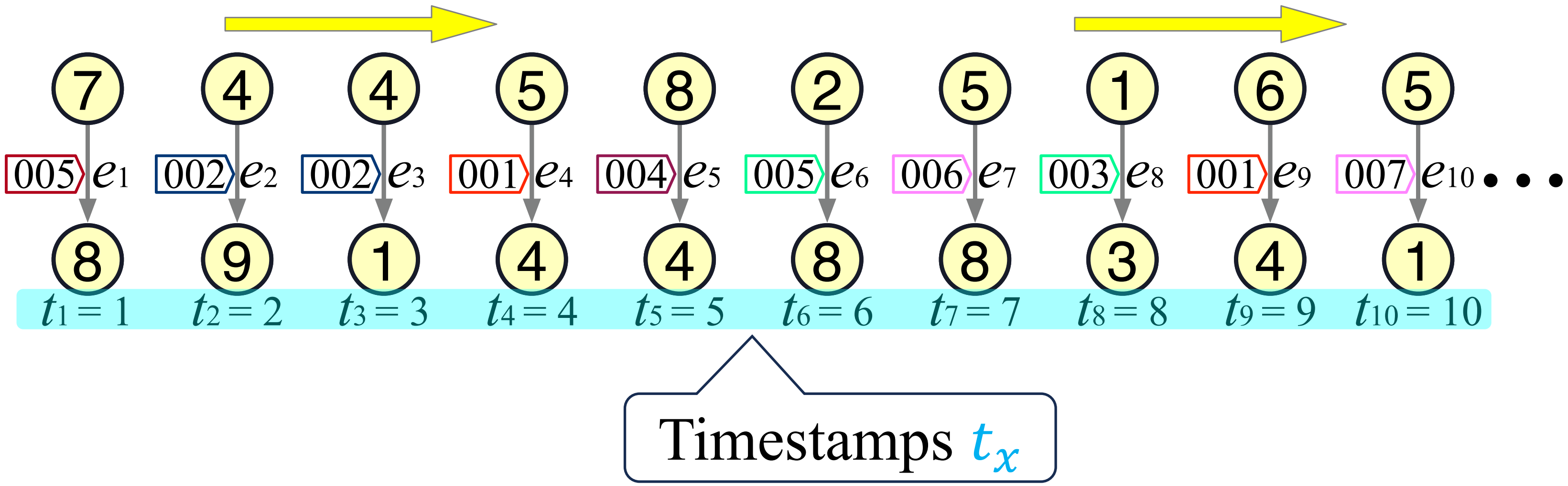
A streaming graph $\mathbb{G} = \{e_x\}_{x \in [X]}$, where $e_x = (sid_x, eid_x, l_x, t_x)$

Example of time-constrained graph pattern detection



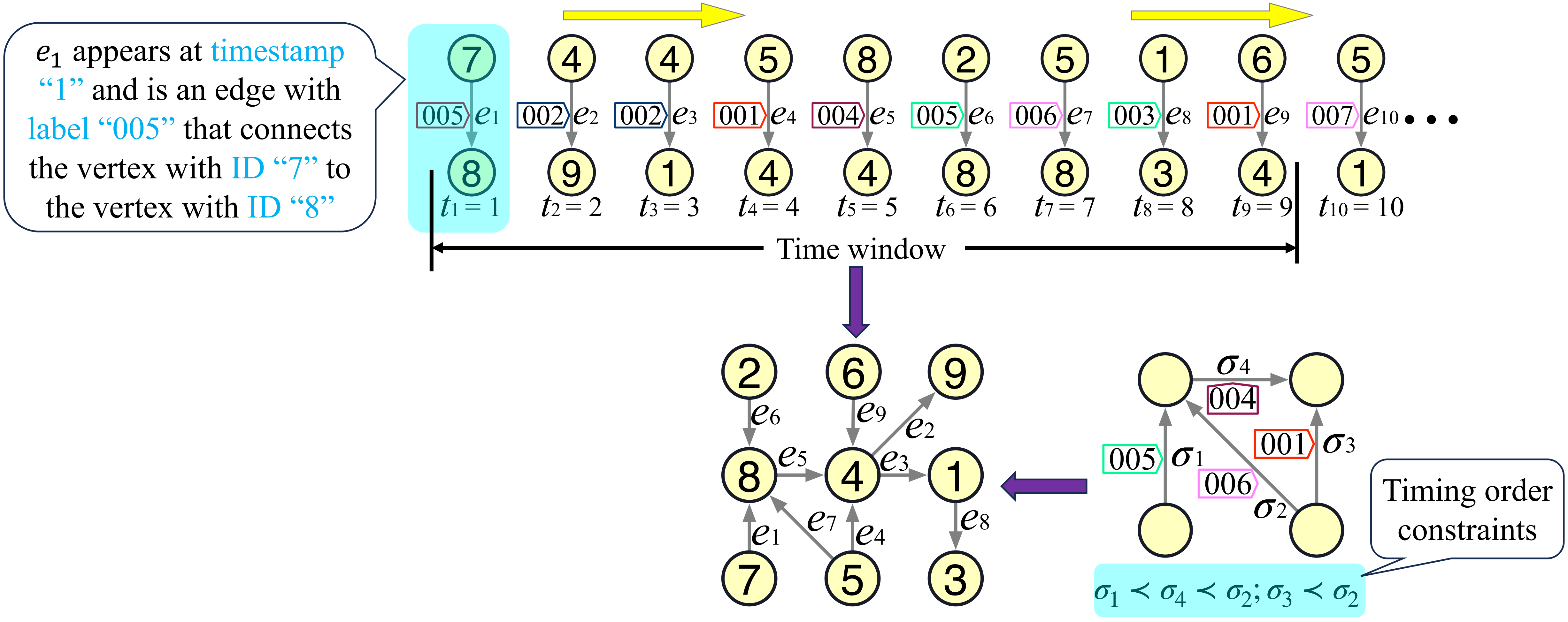
A streaming graph $\mathbb{G} = \{e_x\}_{x \in [X]}$, where $e_x = (sid_x, eid_x, l_x, t_x)$

Example of time-constrained graph pattern detection



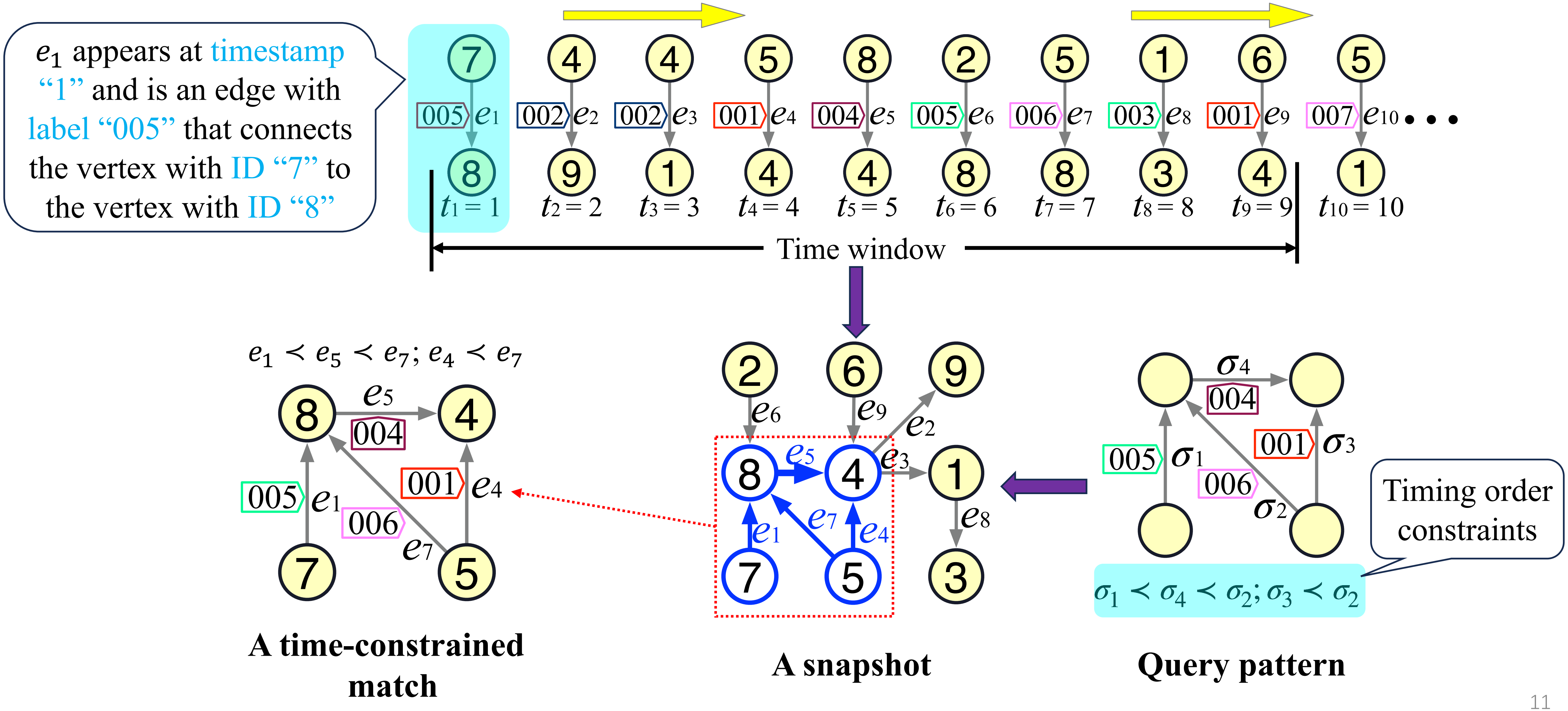
A streaming graph $\mathbb{G} = \{e_x\}_{x\in[X]}$, where $e_x = (sid_x, eid_x, l_x, t_x)$

Example of time-constrained graph pattern detection



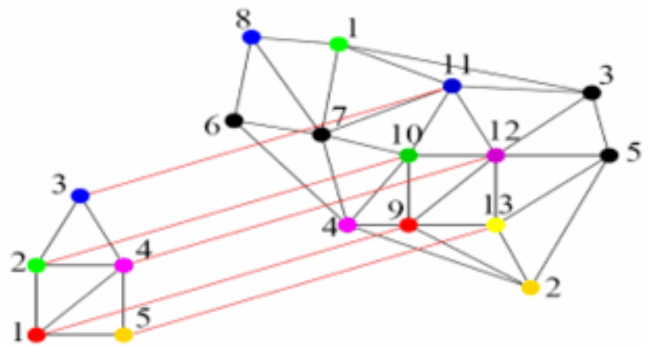
A snapshot **Query pattern**

Example of time-constrained graph pattern detection

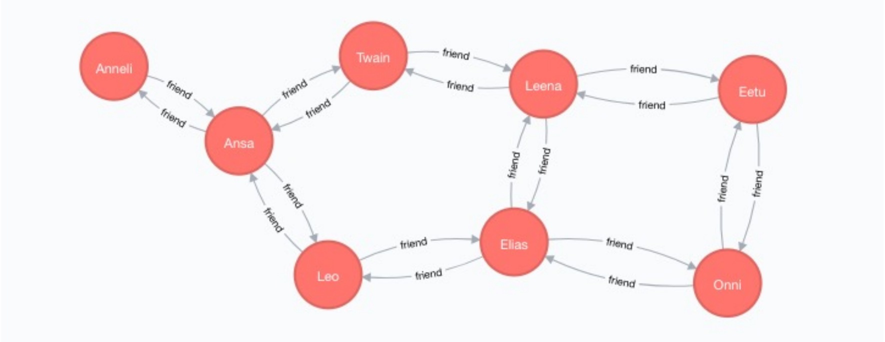


Related works on privacy-aware graph query processing

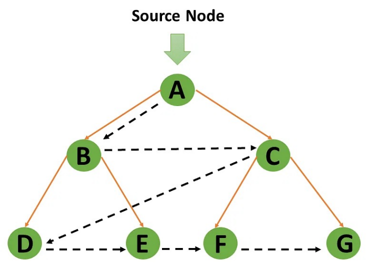
- Mainly focus on privately querying static graphs
  - Private subgraph matching (without timing order constraints) [Xu et al., SIGMOD'23]
  - Private shortest path search [Ghosh et al., AsiaCCS'21]
  - Private breadth-first search [Araki et al., CCS'21]



Subgraph matching



Shortest path search

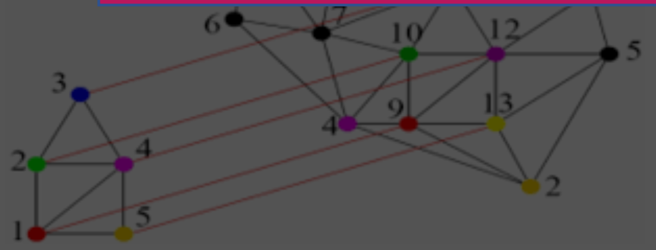


Breadth-first search

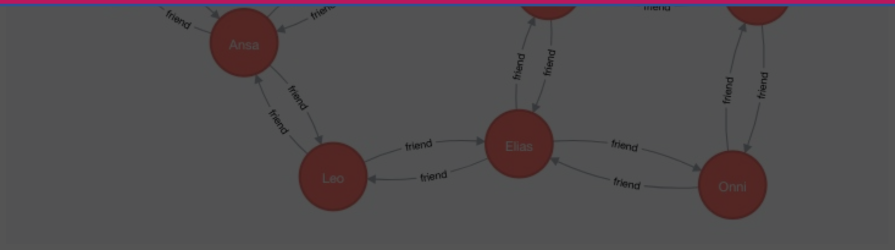
Related works on privacy-aware graph query processing

- Mainly focus on privately querying static graphs
  - Private subgraph matching (without timing order constraints) [Xu et al., SIGMOD'23]
  - Private shortest path search [Ghosh et al., AsiaCCS'21]

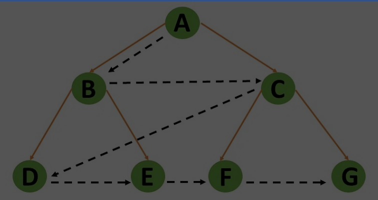
No prior work on privacy-preserving time-constrained pattern detection over streaming graphs.



Subgraph matching



Shortest path search

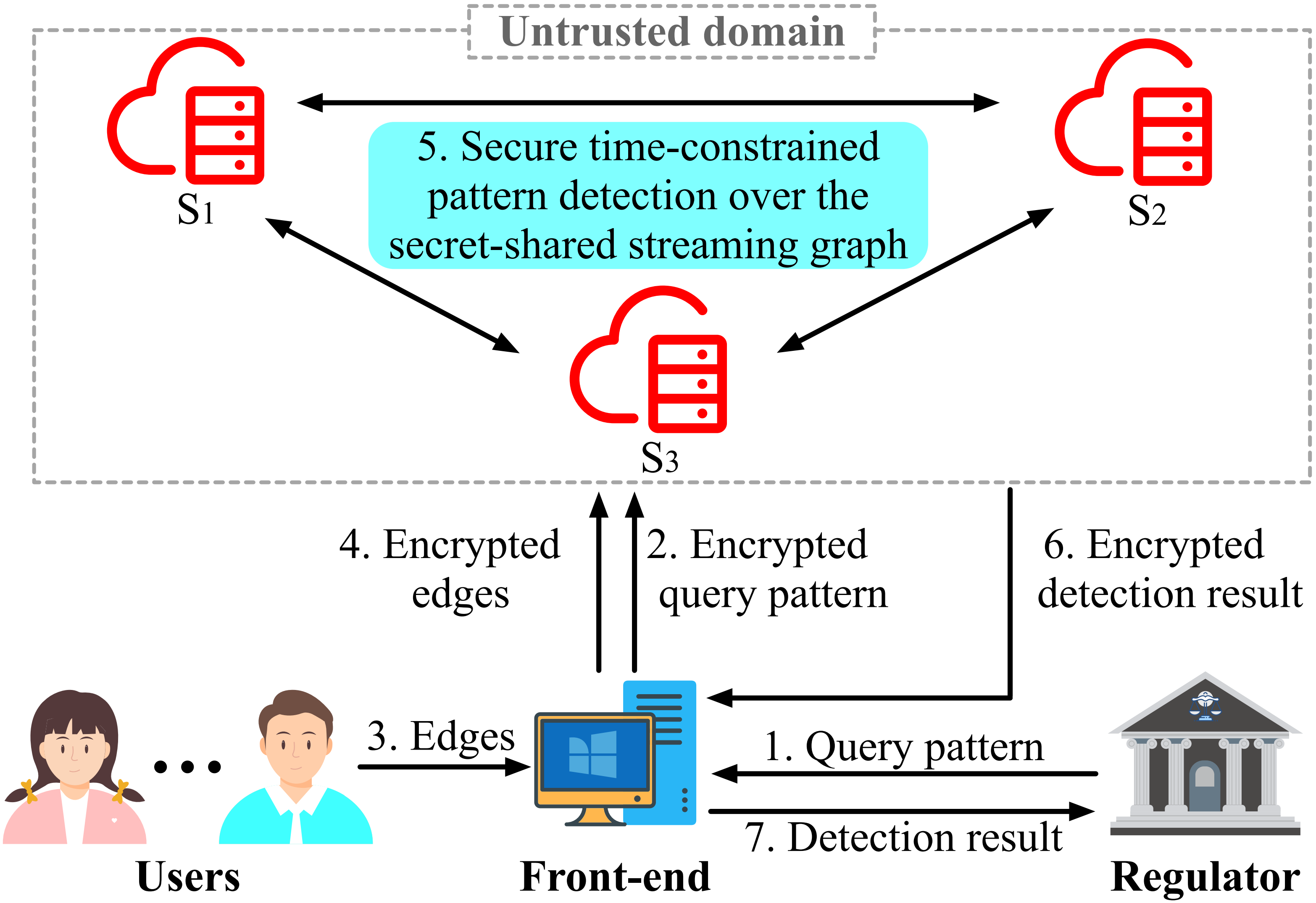


Breadth-first search

Our research effort: GraphGuard

- The first framework for privacy-preserving outsourcing of time-constrained pattern detection over streaming graphs
  - Protect the confidentiality of edge/vertex labels and the connections between vertices in the streaming graph and query patterns

System architecture of GraphGuard

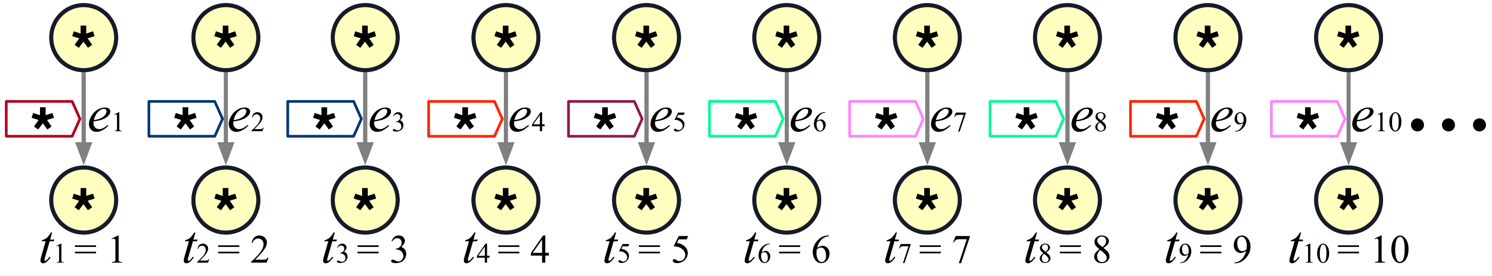


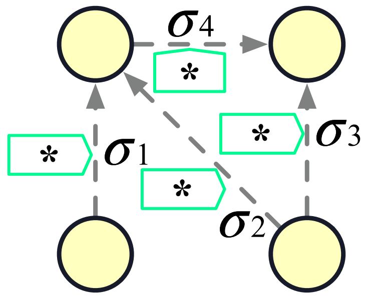
Untrusted domain

Assumption: semi-honest and non-colluding cloud servers (same as prior security designs [Bell et al., CCS'22], [Tan et al., S&P'21], [Wang et al., VLDB'22])

Trusted domain

Security guarantees





$\sigma_1 * \sigma_4 * \sigma_2; \sigma_3 * \sigma_2$

Protected streaming graph

Protected query pattern

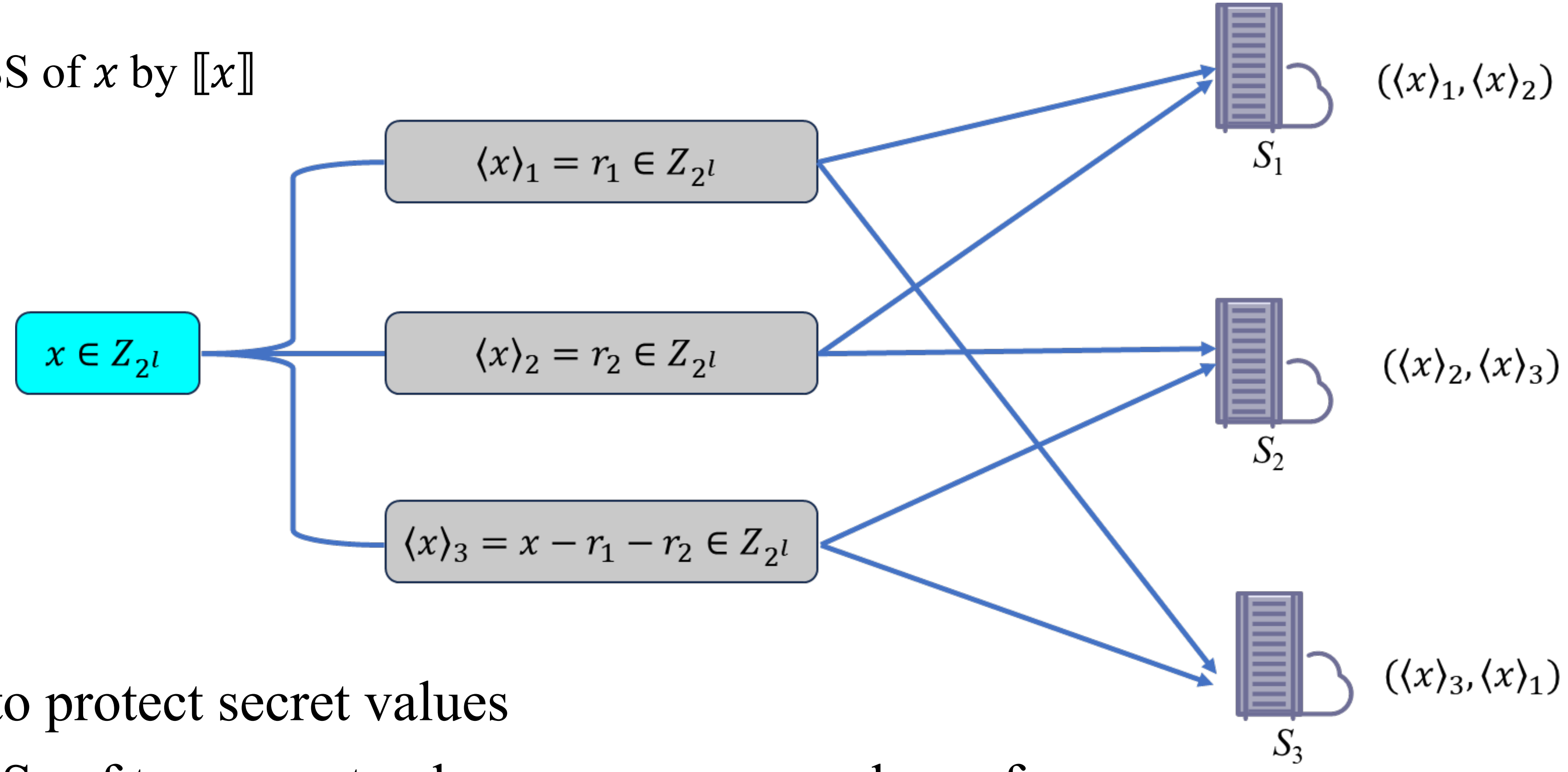
- Protect each edge's **label**
- Hide the **connections** between the vertices
- Public info: timestamps

- Protect each edge's **label**
- Hide the **connections** between the vertices
- Hide the **timing order constraints between each pair of edges**

During the online detection process: Hide the search access patterns

Cryptographic tool: Replicated Secret Sharing

Note: Denote the RSS of x by $[\![x]\!]$



- Can be used to protect secret values
- Given the RSSs of two secret values, we can securely perform:
  - ✓ Addition/subtraction (only local processing needed)
  - ✓ Multiplication (need one communication round)

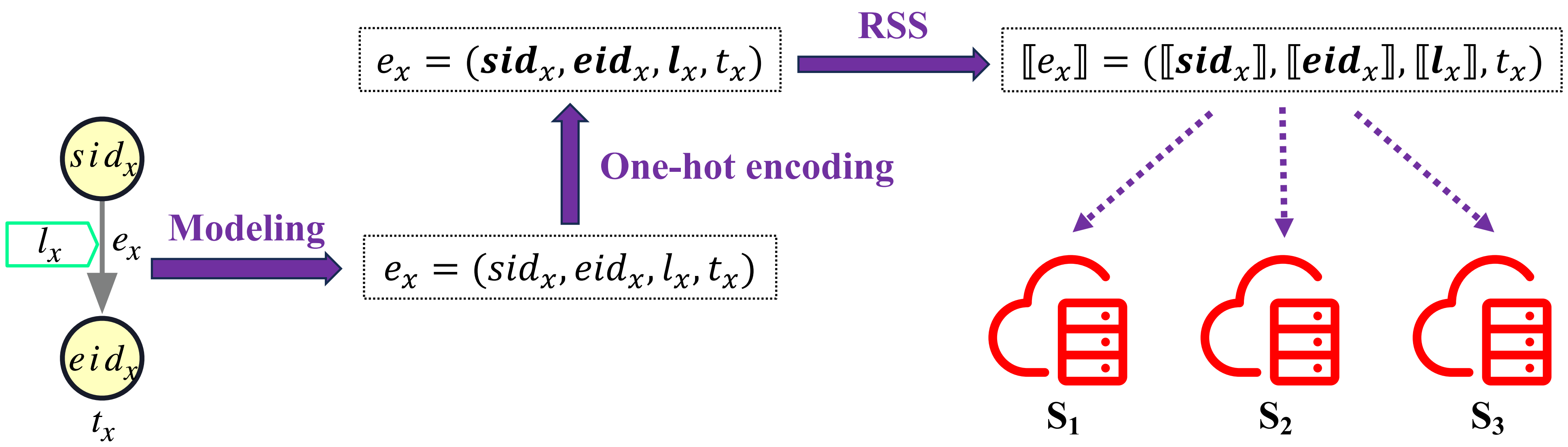
Our technical design

How to protect the streaming graph?

Streaming graph encryption

- GraphGuard processes each edge independently, facilitating subsequent dynamic updates
  - Each edge is modeled as a tuple $e_x = (sid_x, eid_x, l_x, t_x)$
- GraphGuard uses RSS to protect the private values, including sid_x, eid_x, l_x
  - GraphGuard encodes each private value into a one-hot vector, and encrypts each bit via RSS
    - For ensuring efficient equality test in the secret sharing domain

- sid_x: The ID of the source of edge e_x
- eid_x: The ID of the target of edge e_x
- l_x: e_x's label
- t_x: e_x's timestamp



A new generated edge

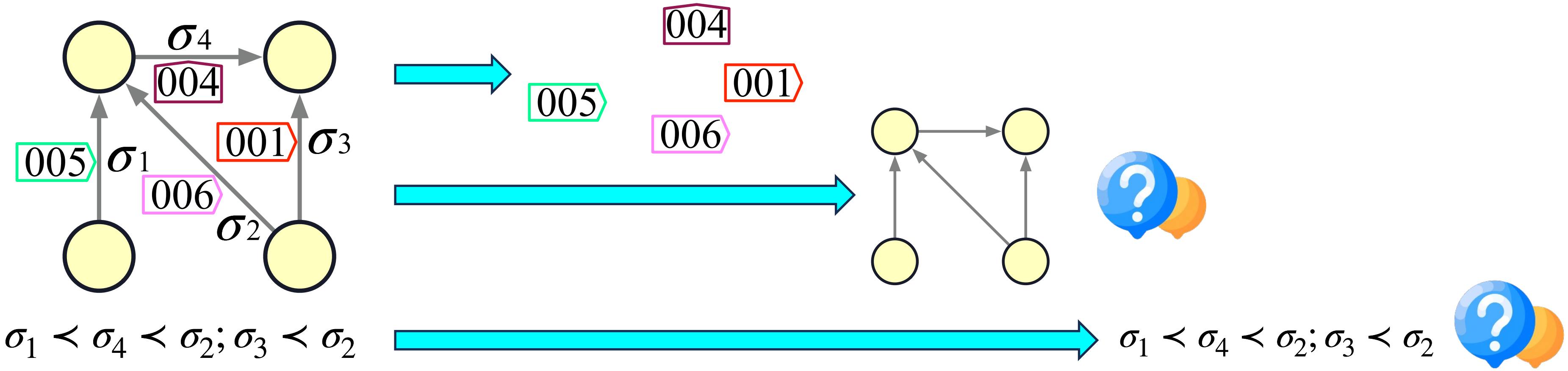
The secret-shared streaming graph $[\![\mathbb{G}]\!] = \{[\![e_x]\!]\}_{x \in [X]}$

Our technical design

How to protect the query pattern?

Query pattern modeling

- It is easy to model the labels of the query pattern
  - $\mathcal{L} = \{l_1, l_2, \cdots\}$
- How to model the structure?
  - **Goal:** Facilitate efficient graph isomorphism checking in the secret sharing domain
- How to model timing order constraints?
  - **Goal:** Facilitate edge temporal consistency checking in the secret sharing domain



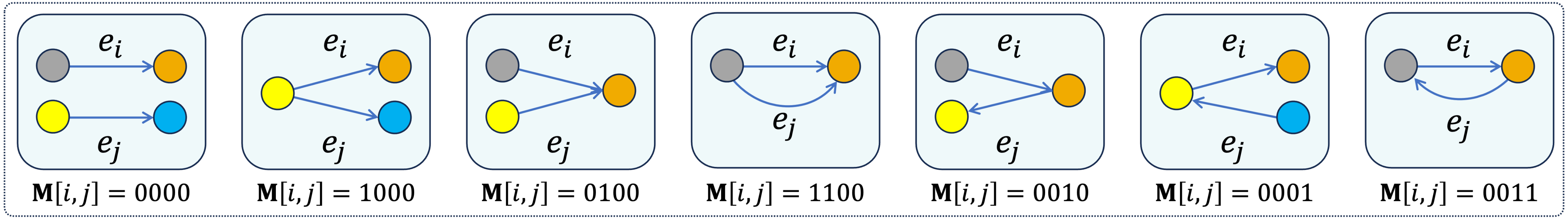
Modeling the structure

- To check graph isomorphism, a common strategy is to find the bijective match function by constructing the search tree along the connections between vertices
  - Difficult to realize in the secret sharing domain

- Therefore, we propose a new data structure - endpoint adjacency matrix (EAM) - to model vertex connections
  - With EAM, checking graph isomorphism can be simplified as the comparison between their EAMs, consisting of only basic "$\oplus$" and "$\otimes$" operations

Modeling the structure

- We enumerate all possible cases of connection relationships, considering the edge directions, between two edges and assign a 4-bit element to each case:



Rules:

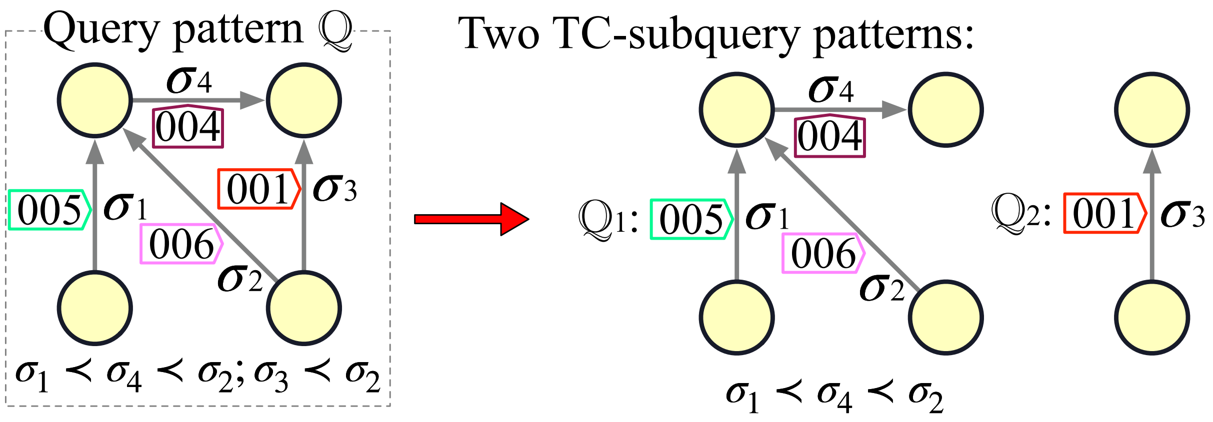
- If the sources of e_i and e_j are connected: The first bit of $\mathbf{M}[i,j]$ is equal to 1
- If the targets of e_i and e_j are connected: The second bit of $\mathbf{M}[i,j]$ is equal to 1
- If the target of e_i is connected to the source of e_j: The third bit of $\mathbf{M}[i,j]$ is equal to 1
- If the source of e_i is connected to the target of e_j: The fourth bit of $\mathbf{M}[i,j]$ is equal to 1

Modeling the timing order constraints

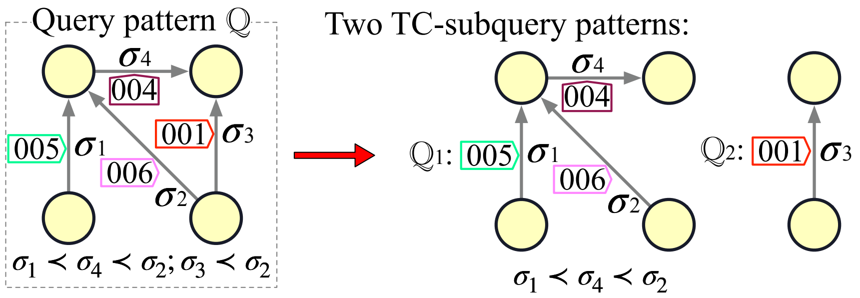
- Decompose the query into timing-connected subquery patterns (TC-subquery patterns) inspired by the plaintext method [Li et al., TKDE'22]
  - To simplify the representation and efficient evaluation of timing order constraints

There is a strict sequential timing order relationship among all the edges in each TC-subquery pattern, i.e., $\sigma_1 \prec \cdots \prec \sigma_\kappa$

Example:



Query pattern modeling and encryption



Modeling

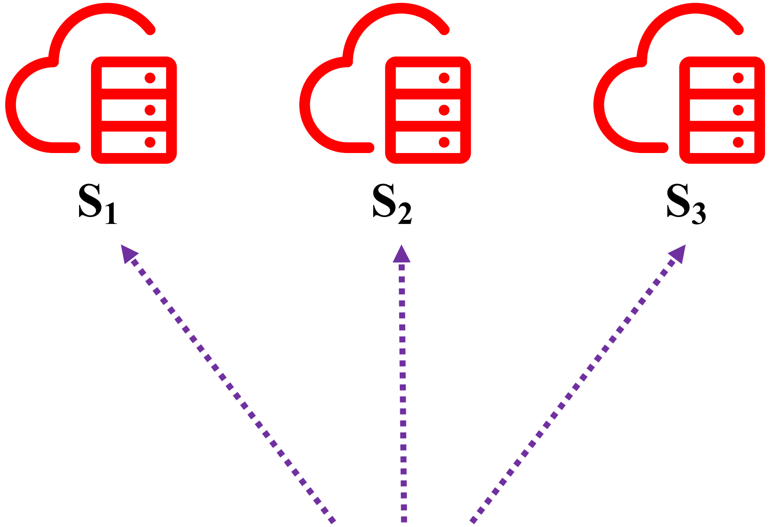
$$\text{EAM } \mathbf{M} = \begin{bmatrix} 0000 & 0100 & 0000 & 0010 \\ 0100 & 0000 & 1000 & 0010 \\ 0000 & 1000 & 0000 & 0100 \\ 0001 & 0001 & 0100 & 0000 \end{bmatrix}$$

Label set $\mathcal{L} = \{005, 006, 001, 004\}$

Decompositions $\mathcal{Y} = \{\boldsymbol{y}_1 = [1,4,2], \boldsymbol{y}_2 = [3]\}$

RSS

$$[\![\mathbb{Q}]\!] = ([\![\mathbf{M}]\!], [\![\mathcal{L}]\!], \mathcal{Y})$$



Our technical design

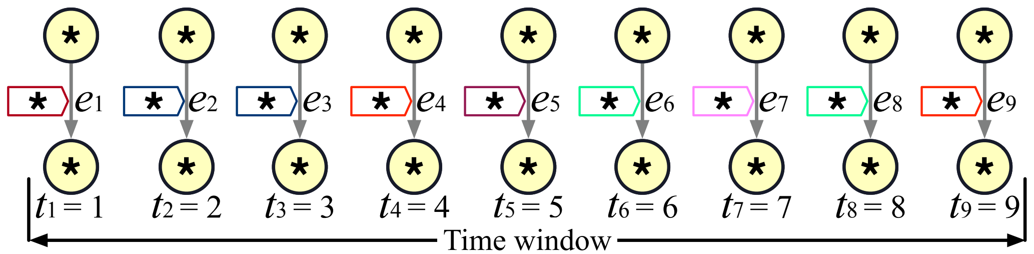
How to securely detect time-constrained matches over each secret-shared snapshot?

Workflow

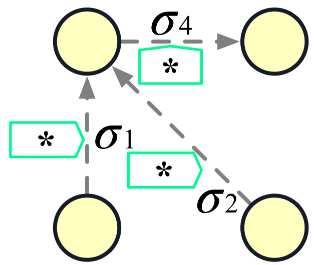
1. Secure matched edges fetching
   - Securely fetch the matched edges for each edge in each TC-subquery pattern
     - Matched edges: The edges whose labels are identical to those in the query pattern
2. Construct candidate partial matches
   - Construct candidate partial matches by the edges from different matched edge sets that obey the timing order constraints of TC-subquery pattern
3. Secure candidate partial matches filtering
   - Securely filter out candidate partial matches whose structures are inconsistent with the corresponding TC-subquery patterns, to obtain the partial matches
4. Secure partial matches compatibility checking
   - Securely check the timing orders and structural compatibility among partial matches to produce the detection result

Secure matched edges fetching

Secret-shared snapshot $[\![\mathbb{G}_9]\!]$



Secret-shared TC-subquery pattern $[\![\mathbb{Q}_1]\!]$



Oblivious dummy edges padding on the snapshot ⇒ **Secure equality test** ⇒ **Reveal the results**

Directly reveal the test results?

No, leak the access pattern

Secret-shared matched edges

Oblivious dummy edges padding

Challenge

How to appropriately set the number of dummy edges to balance the trade-off between efficiency and privacy?

Solution

Draw the number from discrete Laplace distribution $Lap(\varepsilon, \delta, \Delta)$ to make the leakage about the frequency of edge labels differentially private.

Issue

The drawn number cloud be negative

Solution

Truncate it to 0: $\max\{Lap(\varepsilon, \delta, \Delta), 0\}$

Parameters

- $\Delta = 1$
- $\mu = -\frac{\ln[(e^{\varepsilon}+1)\cdot(1-\delta)]}{\varepsilon}$

Refer to Section 5 of our paper for the proof of the DP guarantee

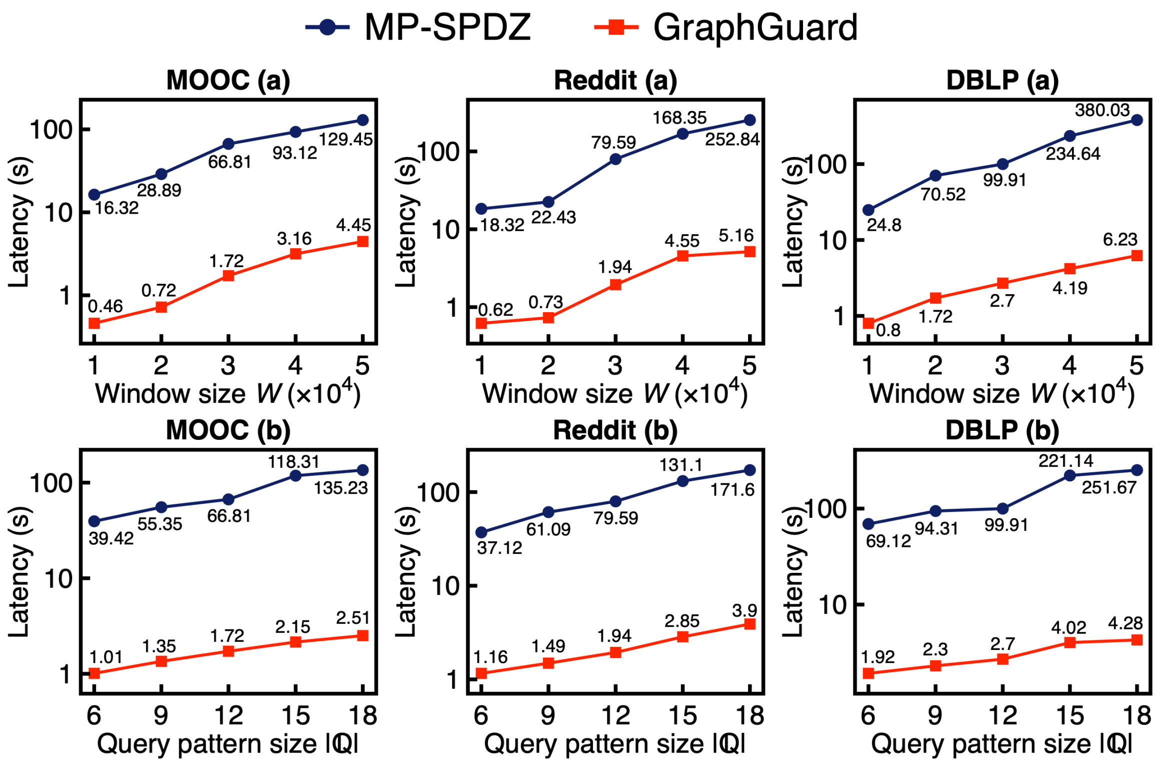
Evaluation setup

- Implementation: Python and C++
- Dataset: three real-world graph datasets:
  - MOOC user action (MOOC)[1]: 7,143 vertices and 411,749 temporal edges
  - Reddit hyperlink network (Reddit)[2]: 55,863 vertices and 858,490 temporal edges
  - com-DBLP (DBLP)[3]: 317,080 vertices and 1,049,866 edges
- Deployment
  - Cloud servers: A workstation with 24 Intel Xeon Gold 6240R CPU cores and 128 GB RAM running Ubuntu 20.04.3 LTS (latency: 10 ms)
  - Front-end: a MacBook Air with 8 GB of RAM
- Baseline: using the generic and popular framework MP-SPDZ [Keller et al., CCS'20]

1. https://snap.stanford.edu/data/act-mooc.html
2. https://snap.stanford.edu/data/soc- RedditHyperlinks.html
3. https://snap.stanford.edu/data/com-DBLP.html

Evaluation on query latency

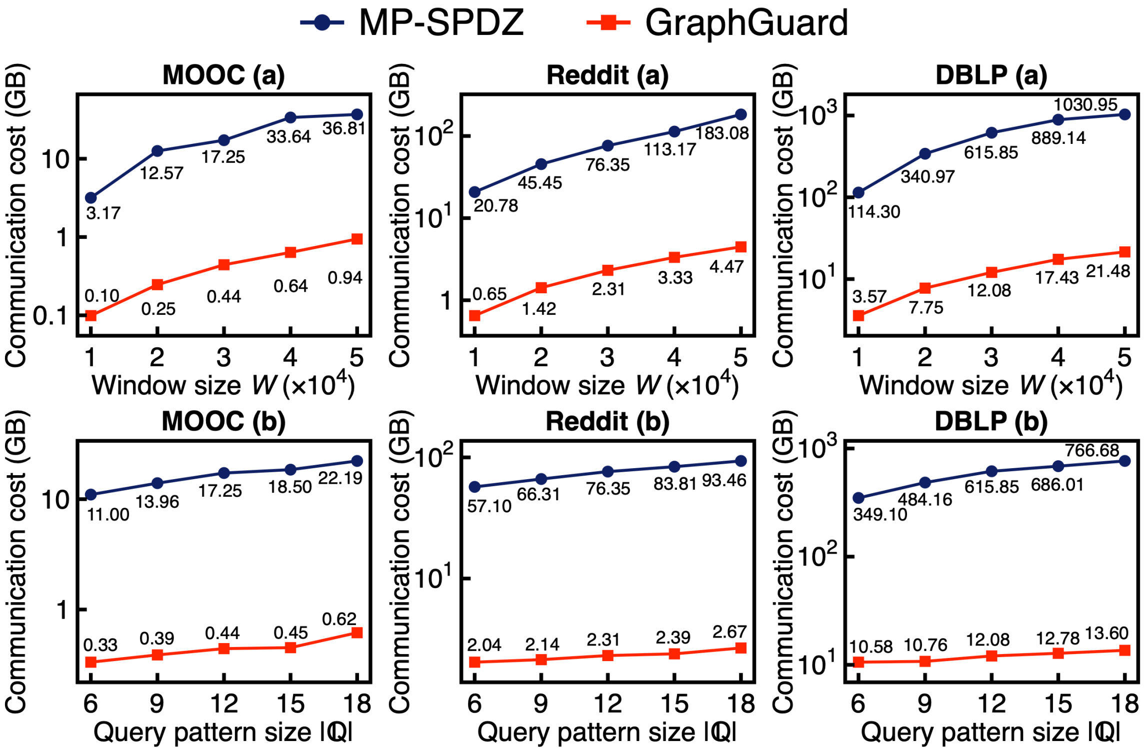
(a) $|\mathbb{Q}| = 12,\ W \in [1\times10^4, 5\times10^4]$
(b) $W = 3\times10^4, |\mathbb{Q}| \in [6,9,12,15,18]$



- The query latency gap between GraphGuard and the baseline increases significantly as the values of window size W and query pattern size $|\mathbb{Q}|$ increase
- The results clearly demonstrate that GraphGuard consistently outperforms the baseline, achieving a substantial speedup ranging from 29× to 60×

Evaluation on the server-side communication cost

(a) $|\mathbb{Q}| = 12,\ W \in [1\times10^4, 5\times10^4]$
(b) $W = 3\times10^4, |\mathbb{Q}| \in [6,9,12,15,18]$



- Communication cost savings of GraphGuard compared to the baseline increase significantly as the values of W and query pattern size $|\mathbb{Q}|$ increase
- GraphGuard consistently outperforms the baseline, achieving substantial communication cost savings ranging from 96% to 98%

Summary

- The first framework for privacy-preserving outsourcing of time-constrained pattern detection over streaming graphs
  - Bridge insights on graph processing and lightweight cryptography
  - Achieve secure subgraph isomorphism search on dynamic graphs
- GraphGuard substantially outperforms the baseline constructed by the generic MPC framework
  - 60× improvement in query latency and up to 98% savings in communication
- Directions for future work:
  - The support for malicious security
  - The support for vertex/edge deletion

Thank You!
Q&A?

songlei.wang@outlook.com