Bloom Filter-Based Multi-Party Private Set Intersection (MPSI) 1st Stage Evaluation

Andra Alăzăroaie

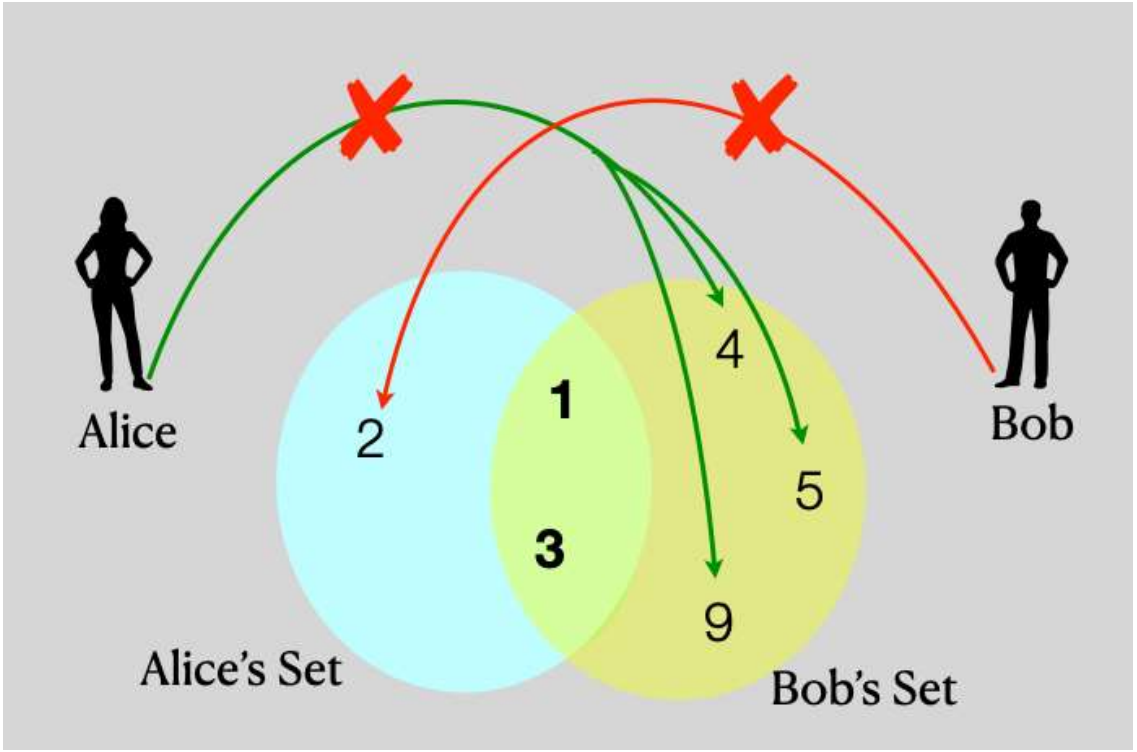
Supervisor: Lilika Markatou

Daily Supervisor: Tjitske Koster

09-02-2026

Introduction

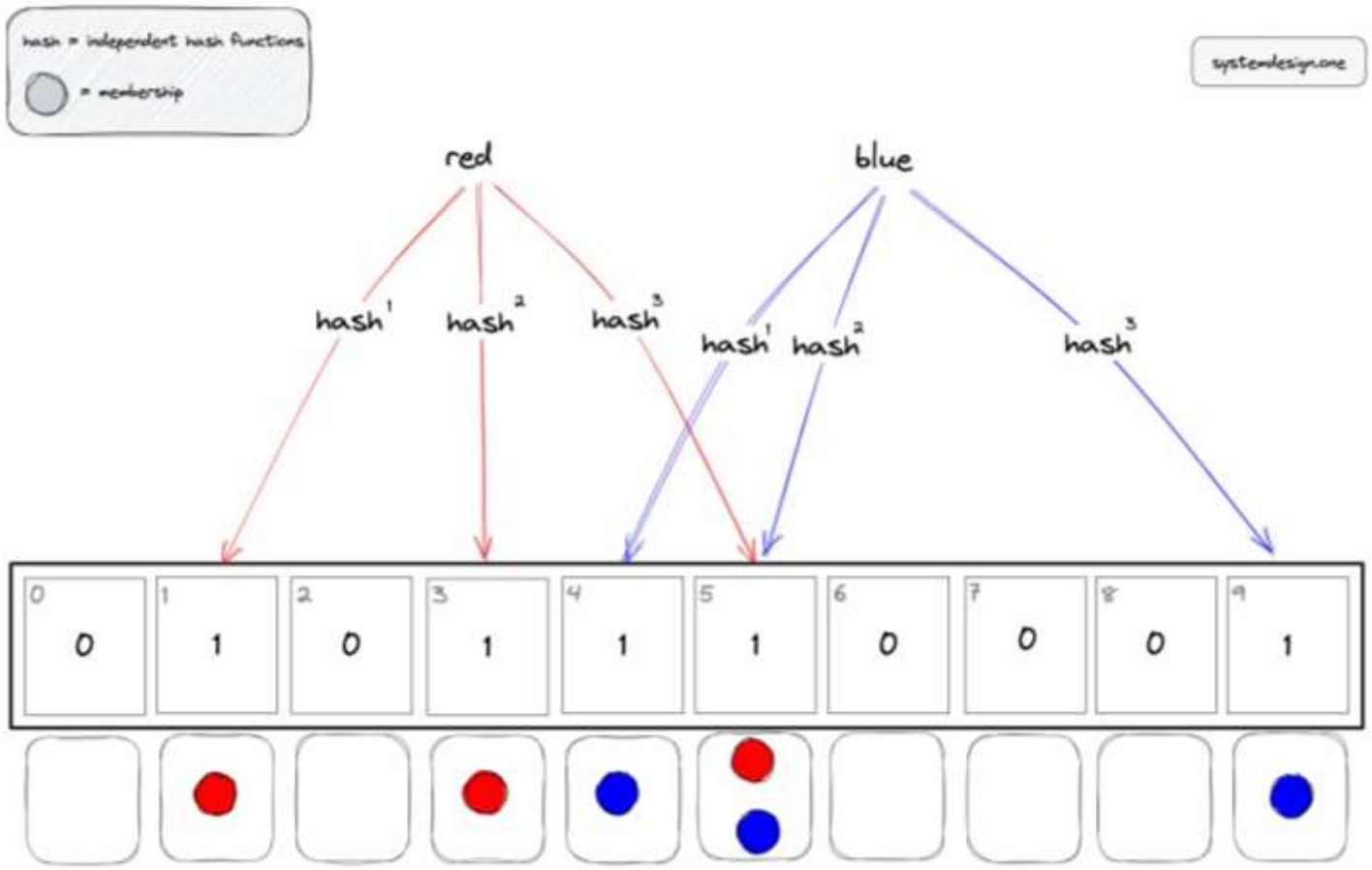
- **Private Set Intersection (PSI):**
  - two or more parties find the items they have in common
  - parties **ONLY** reveal the intersection
- **Multi-Party PSI (MPSI):**
  - PSI with n participants
- **Applications:**
  - contact medical analysis
  - anomaly detection
  - social networks
  - joint marketing campaigns [2].



https://en.wikipedia.org/wiki/Private_set_intersection

Background: Bloom Filters

- A common MPSI approach uses **Bloom Filters**
- **Insert:** hash it h times to get h positions in a bit array (size m). Set those bits to **1**.
- **Query:** hash it h times and check if all h bits are **1**.
- **Intersection: bitwise AND** operation on all parties' filters.
- No *false negatives*, but it can have *false positives.*



https://systemdesign.one/bloom-filters-explained/

The Research Gap

- Trade-off of Bloom filter-based MPSI: **high efficiency** in exchange for a **small, acceptable error rate** (false positives).

- **Vos et al. [3]** demonstrates that these protocols are **fundamentally insecure**.

- Simple fix: make the false positive rate negligible (e.g., $p \leq 2^{-30}$).
  - Requires a large filter size (m), so the protocol becomes **computationally inefficient**

Main Research Question

Can we design an efficient and secure Bloom filter-based multi-party PSI (MPSI) protocol?

The goal = find a new design that addresses the vulnerability from **Vos et al. [3]** *without* sacrificing the performance that made Bloom filters attractive in the first place.

Sub-Questions

RQ1. What is the **baseline performance** (in terms of computation, communication, and scalability) of an existing, unmitigated Bloom filter MPSI protocol?

RQ2. Under which conditions do Bloom filter-based MPSI protocols leak information and how does the attack by Vos et al. [3] exploit these conditions?

RQ3. To what extent do existing **mitigation techniques** (such as adjusting the parameters m, k, and h, OPRF-based blinding, garbled Bloom filters and input validation using a judge) reduce or prevent this leakage? How do these mitigation techniques impact performance?

RQ4. Can we design a **new Bloom filter-based MPSI protocol** that provides both improved **privacy guarantees** and improved **performance**?

RQ5. How does the proposed protocol compare to existing mitigations in terms of computation, communication, and scalability across different network communication topologies (star, full-mesh, wheel etc.)?

Methodology: Stage 1 – Familiarization

- **Implement Baseline Protocol - RQ1:**
  - We ~~will~~ implemented an existing Bloom filter MPSI protocol from the literature (e.g., Bay et al. [4], Vos et al. [5], **Ruan et al. [6]** or Ruan and Ai [7]).
  - This implementation of [6] ~~will~~ serves as our performance baseline for future comparisons.

- **Analyze the Attack - RQ2:**
  - We ~~will~~ study~~y~~ied the attack presented by **Vos et al. [3]**.
  - We ~~will~~ identify~~y~~ied the precise conditions that allow information leakage in our baseline implementation.

Methodology: Stage 1 – Familiarization

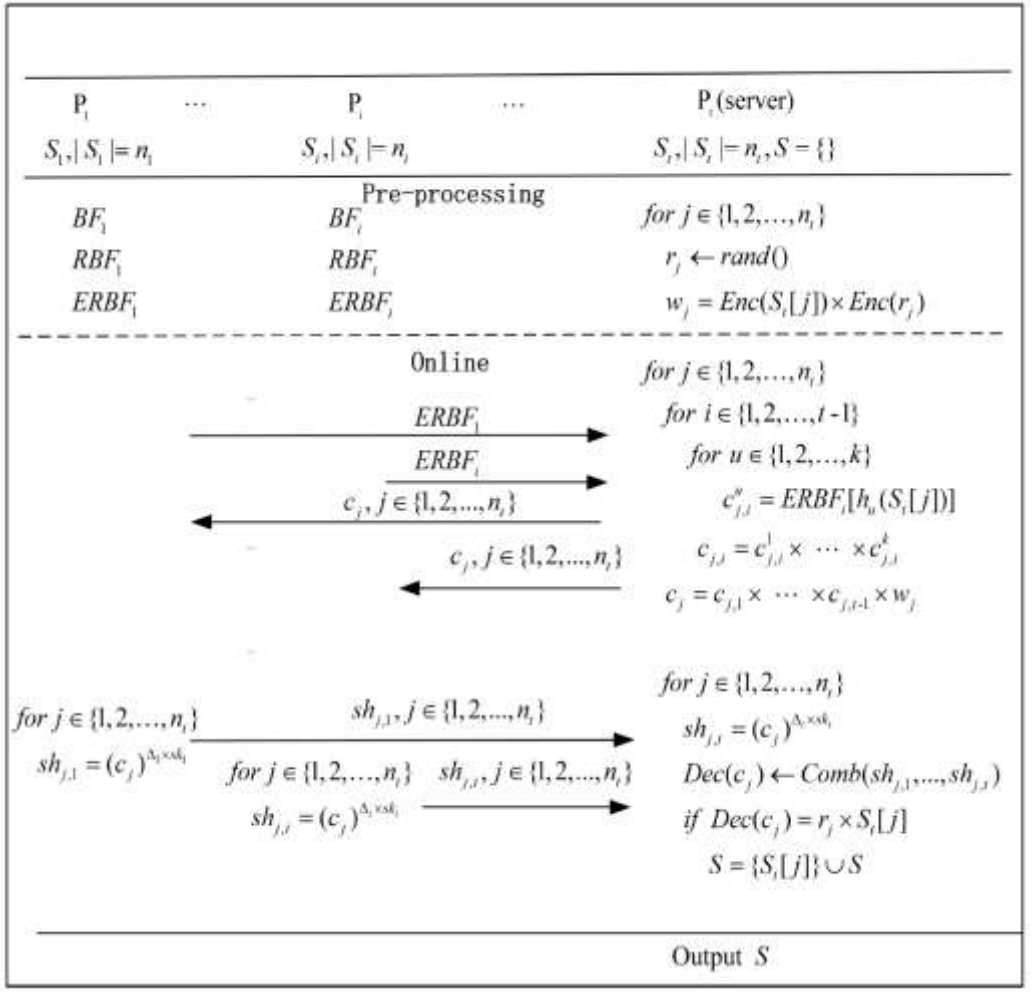
- **Implement Baseline Protocol:**

  Ruan et al. [6] (RQ1)

BF = Bloom filter

RBF = randomized Bloom filter

ERBF = encrypted randomized Bloom filter

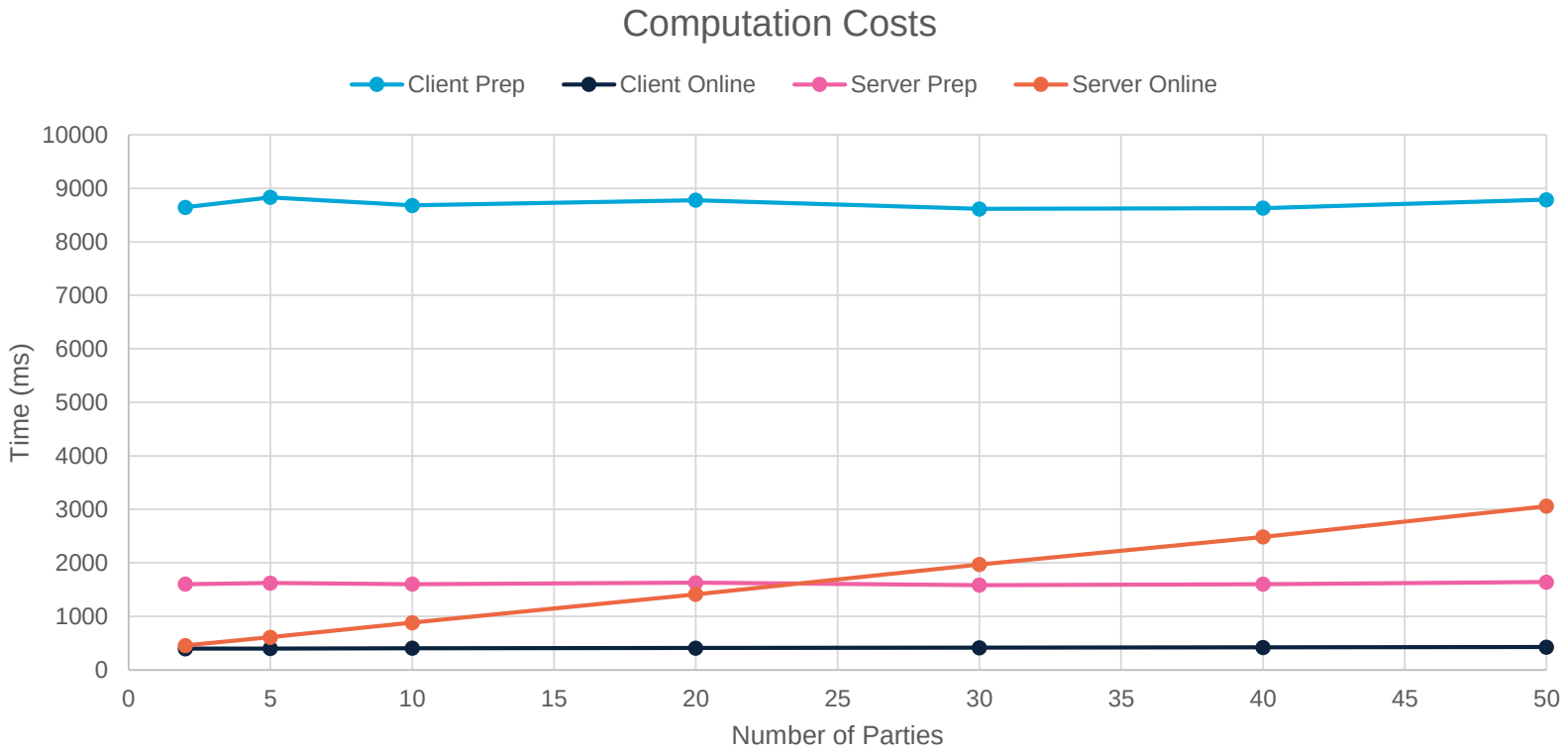


Methodology: Stage 1 – Familiarization

Ruan et al. [6] results **(RQ1)**:

Computation time (in ms) per party for a run of the protocol involving various number of parties (including the server)

Set size clients: 256 elements
Set size server: 1024 elements
False positive rate: 2^{-30}

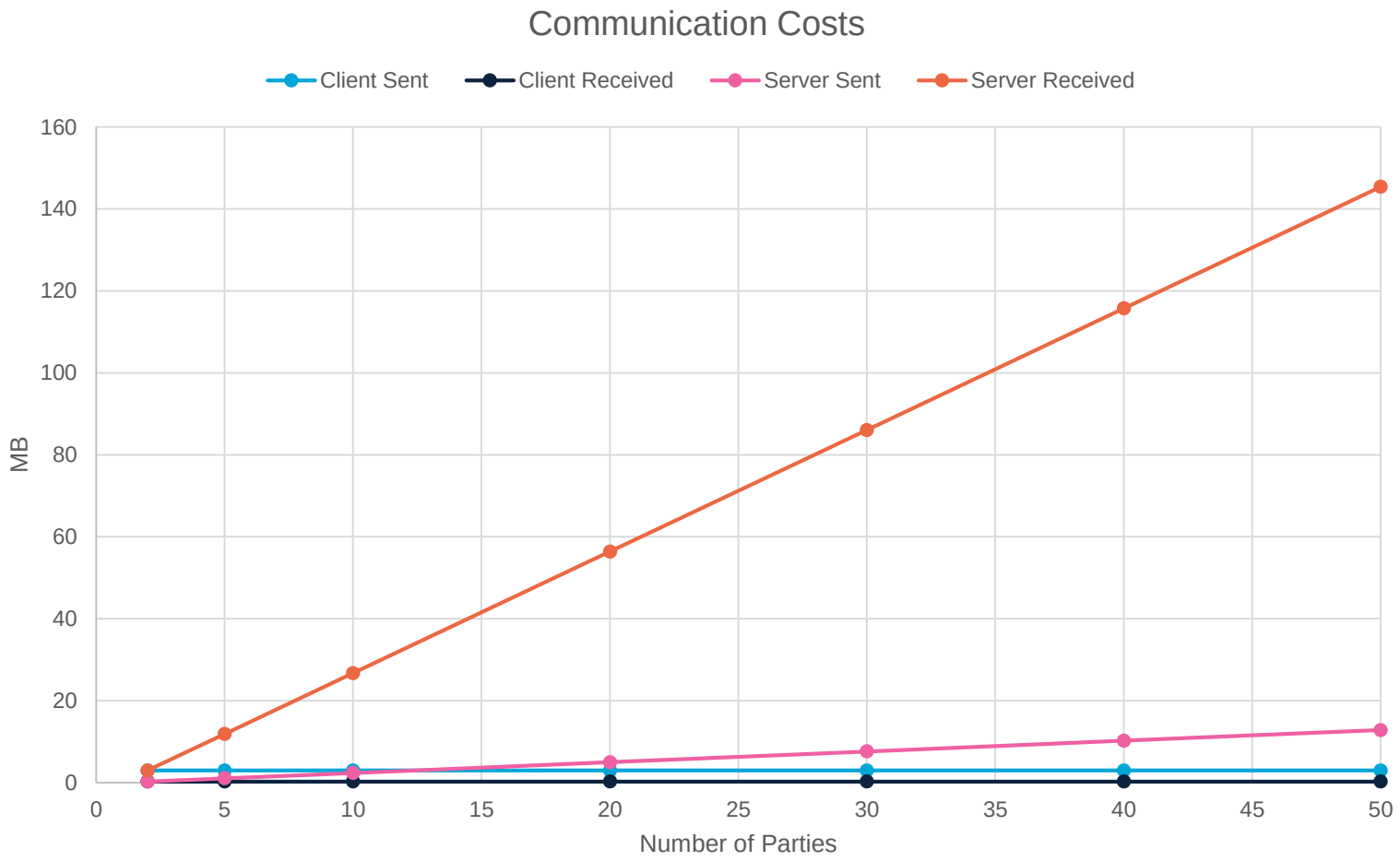


Methodology: Stage 1 – Familiarization

Ruan et al. [6] results **(RQ1)**:

Communication size (in MB) per party for a run of the protocol involving various number of parties (including the server)

Set size clients: 256 elements
Set size server: 1024 elements
False positive rate: 2^{-30}



Methodology: Stage 1 – Familiarization

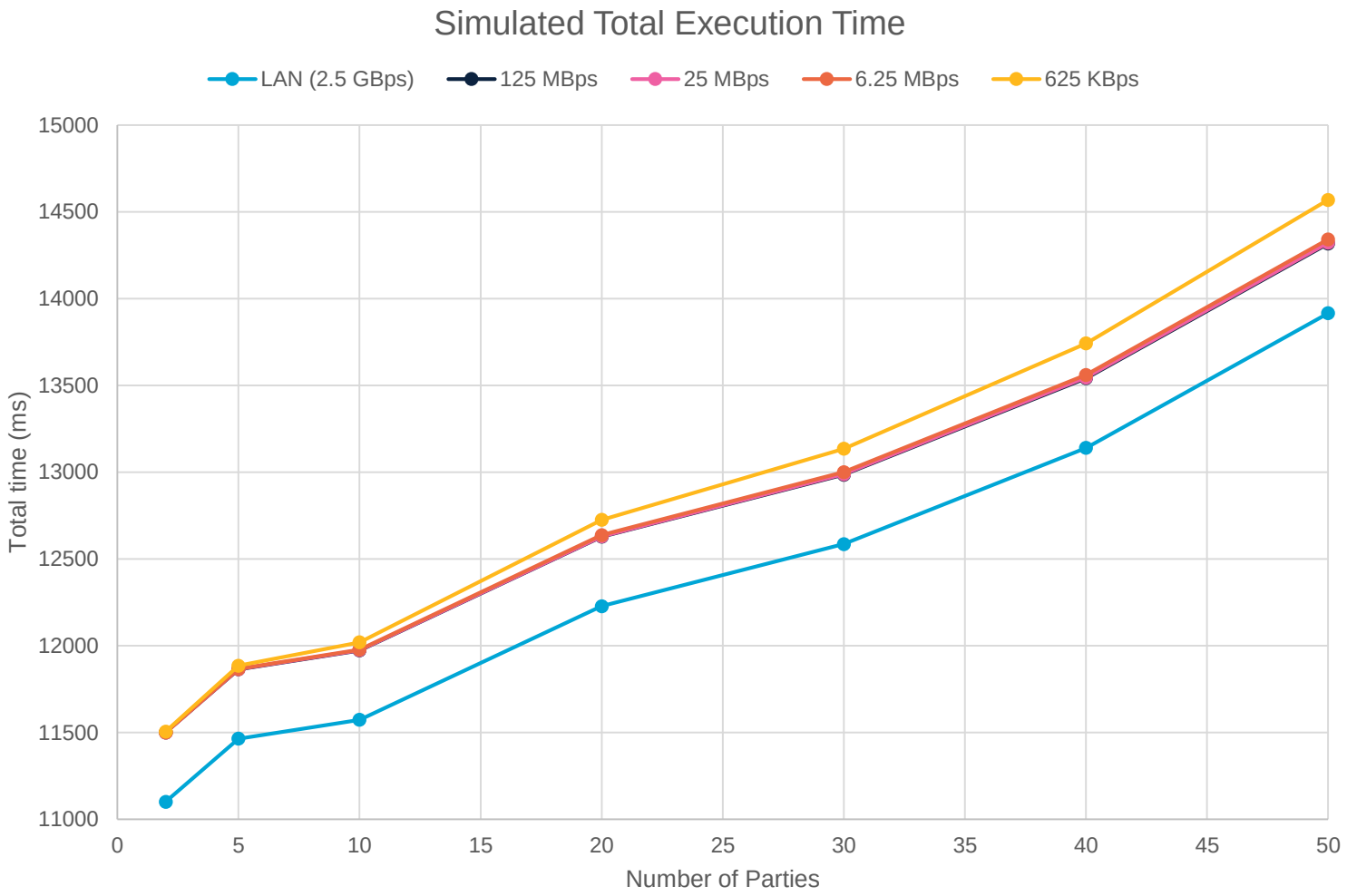
Ruan et al. [6] results **(RQ1)**:

Total time (in ms) for a run of the protocol involving various number of parties (including the server)

Set size clients: 256 elements
Set size server: 1024 elements
False positive rate: 2^{-30}

Total time calculated as:
Latency + Communication time + Computation Time

- **Latency:** *number of messages * hardcoded network latency*
- **Communication time:** *Bytes sent * hardcoded bandwidth*
- **Computation time:** plotted before



Methodology: Stage 1 – Familiarization

Why Ruan et al. [6] is vulnerable to the attack in Vos et al. [3] (RQ2):

- **Vos et al. [3]** constructs a distinguisher that can tell *the idealized behavior* π_{BF} *of a BF-based MPSI protocol* apart from *the idealized behavior* $\mathcal{F}_{waMPSI}$ *of an approximate MPSI protocol*
- **Ruan et al. [6]** reduces to the idealized behavior π_{BF} of a BF-based MPSI protocol
- The false positives of π_{BF} are not random, as those of $\mathcal{F}_{waMPSI}$
- A practical attack can be deployed on π_{BF}

Methodology: Stage 2 - Mitigations

Next, we will implement and evaluate four known mitigation strategies (**RQ3**):

1. Parameter Tuning:

- Make the false positive rate negligible ($p \leq 2^{-30}$) by significantly increasing the filter size m and recalculating the number of hash functions h.
- **Expected result:** Reduction in leakage, but increase in computation and communication costs.

2. OPRFs (Oblivious Pseudorandom Functions):

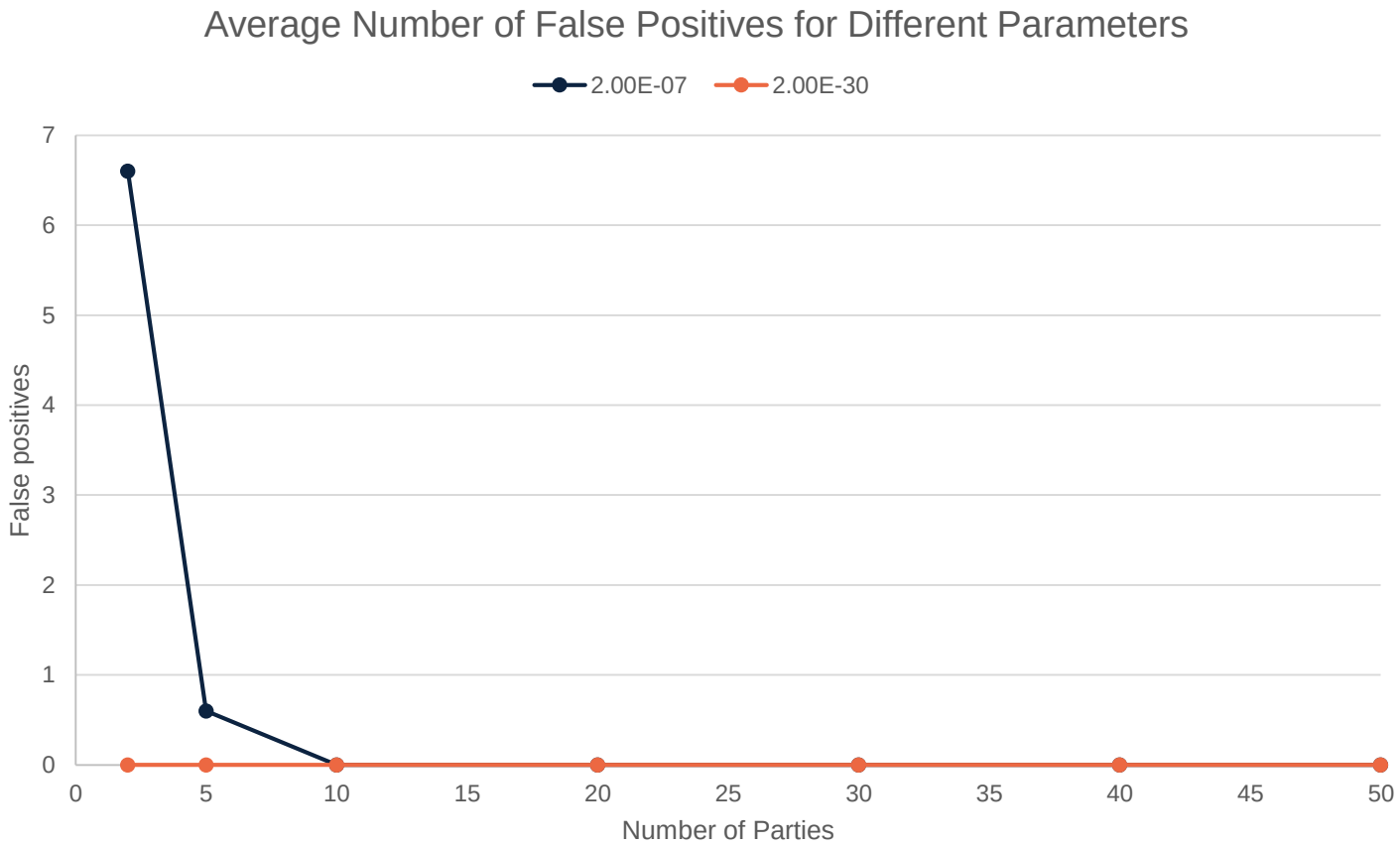
- Replace public hash functions with a secret-seeded OPRF. Parties must engage in a protocol to get hash positions.
- **Expected result:** Prevents the attacker from performing many offline queries, as they cannot compute hashes freely.

Methodology: Stage 1 – Familiarization

Ruan et al. [6] results, **Parameter Tuning (RQ3)**:

(Average) Number of False Positives for a run of the protocol involving various number of parties (including the server) **for different False Positive Rates**

Set size clients: 256 elements
Set size server: 1024 elements

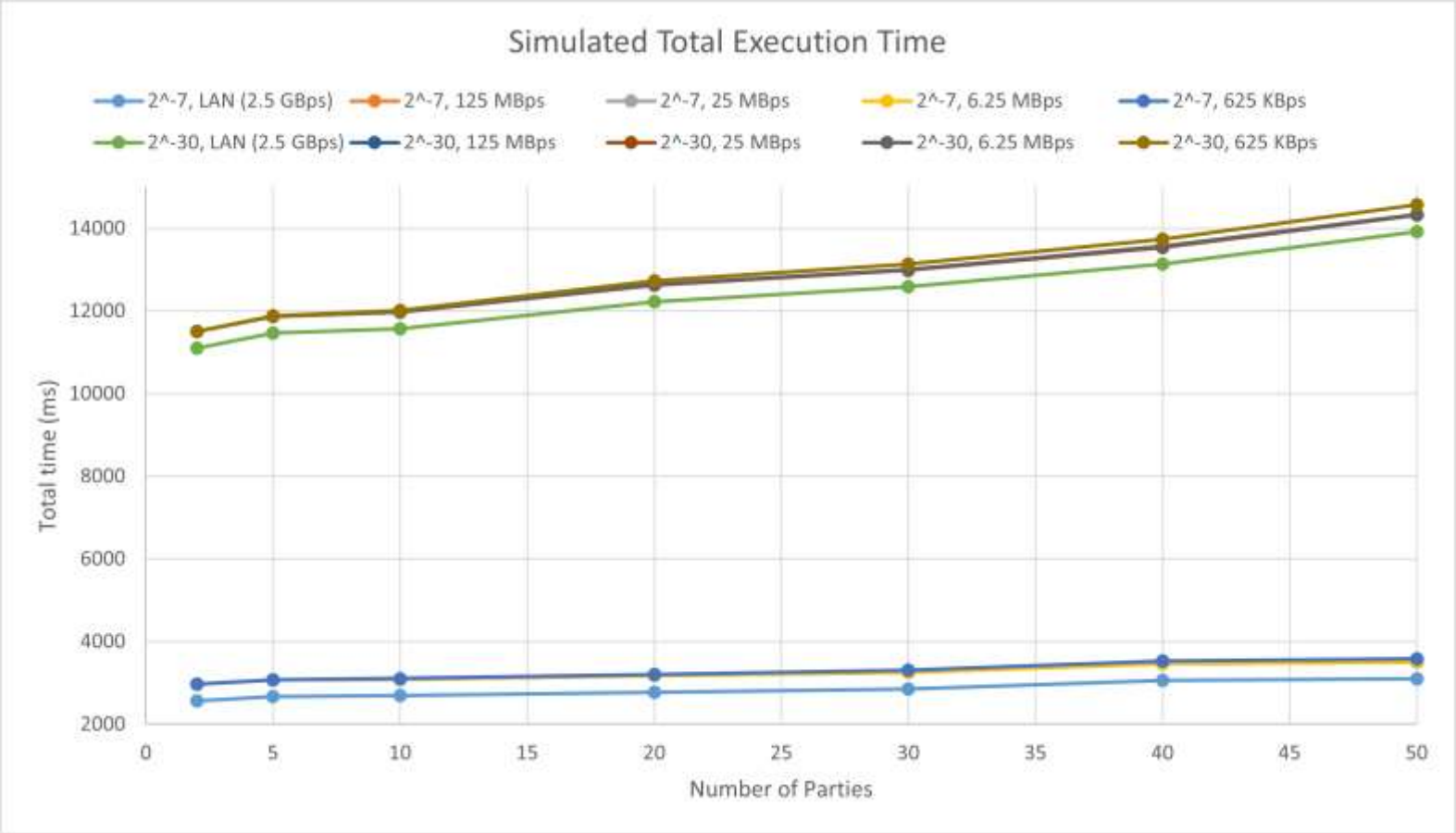


Methodology: Stage 1 – Familiarization

Ruan et al. [6] results, **Parameter Tuning (RQ3)**:

Total time (in ms) for a run of the protocol involving various number of parties (including the server) **for different False Positive Rates**

Set size clients: 256 elements
Set size server: 1024 elements



Methodology: Stage 2 - Mitigations

3. Garbled Bloom Filters (GBF):

- GBFs introduce additional randomness during element insertion. Parties construct a garbled data structure representing their filter, which is sent to other parties who can obliviously test their own elements against it.
- **Expected result:** Prevents the adversary from observing and exploiting the false positive behaviour.

4. Authorized PSI (Judge):

- We outsource trust to a semi-honest third party (judge) to pre-process and encrypt inputs *before* they are inserted into the filter. The judge signs the Bloom filter.
- **Expected result:** Prevents the adversary from constructing malicious inputs.

Methodology: Stage 3 – New Protocol

- **Design New Protocol** (RQ4)**:**
  - Using insights from Stages 1 and 2, we will design a **new Bloom filter-based MPSI protocol,** while the primary goal is to achieve an improved balance: **security** against the [1] attack while maintaining **efficiency**.
  - We will develop formal security arguments for the new protocol.
- **Evaluation** (RQ5)**:**
  - We will experimentally compare our new protocol against the baseline and mitigation strategies.
  - **Metrics:** Computation and communication cost, scalability.
  - **Network Topologies:** We will test how performance is affected by different communication topologies (Star, Full-Mesh, and Wheel).
  - **Tools:** All implementations will be in C++ using standard crypto libraries.

Timeline (Nov 2025 – Jun 2026)

1. **Baseline Implementation (November – January)**
   - Implement an existing Bloom filter based MPSI protocol
   - Analyze why the implementation is vulnerable based on [1]
2. **Mitigations (January – February)**
   - Implement and evaluate mitigations (larger Bloom Filters, OPRF, GBF, Authorized PSI)
3. **New Bloom Filter-based protocol (February – May)**
   - Design and implement a new, secure, Bloom-filter based MPSI protocol
   - Conduct experiments to compare the new protocol with the baseline and mitigations
4. **Writing (May – June)**
   - Finalize thesis writing and prepare presentation
   - Green light meeting mid-May
5. **Defense (end of June)**

Bibliography

[1] Jelle Vos, Mauro Conti, and Zekeriya Erkin. Sok: Collusion-resistant multi-party private set intersections in the semi-honest model. In 2024 IEEE Symposium on Security and Privacy (SP), pages 465–483. IEEE, 2024.

[2] Daniel Morales, Isaac Agudo, and Javier Lopez. Private set intersection: A systematic literature review. Computer Science Review, 49:100567, 2023.

[3] Jelle Vos, Jorrit van Assen, Tjitske Koster, Evangelia Anna Markatou, and Zekeriya Erkin. On the insecurity of bloom filter-based private set intersections. Cryptology ePrint Archive, 2024.

[4] Aslı Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos. Practical multi-party private set intersection protocols. IEEE Transactions on Information Forensics and Security, 17:1–15, 2021.

[5] Jelle Vos, Mauro Conti, and Zekeriya Erkin. Fast multi-party private set operations in the star topology from secure ands and ors. Cryptology ePrint Archive, 2022.

[6] Ou Ruan, Changwang Yan, Jing Zhou, and Chaohao Ai. A practical multiparty private set intersection protocol based on bloom filters for unbalanced scenarios. Applied Sciences, 13(24):13215, 2023.

[7] Ou Ruan and Chaohao Ai. An efficient multi-party private set intersection protocols based on bloom filter. In Second International Conference on Algorithms, Microchips, and Network Applications (AMNA 2023), volume 12635, pages 282–287. SPIE, 2023.

Thank you!

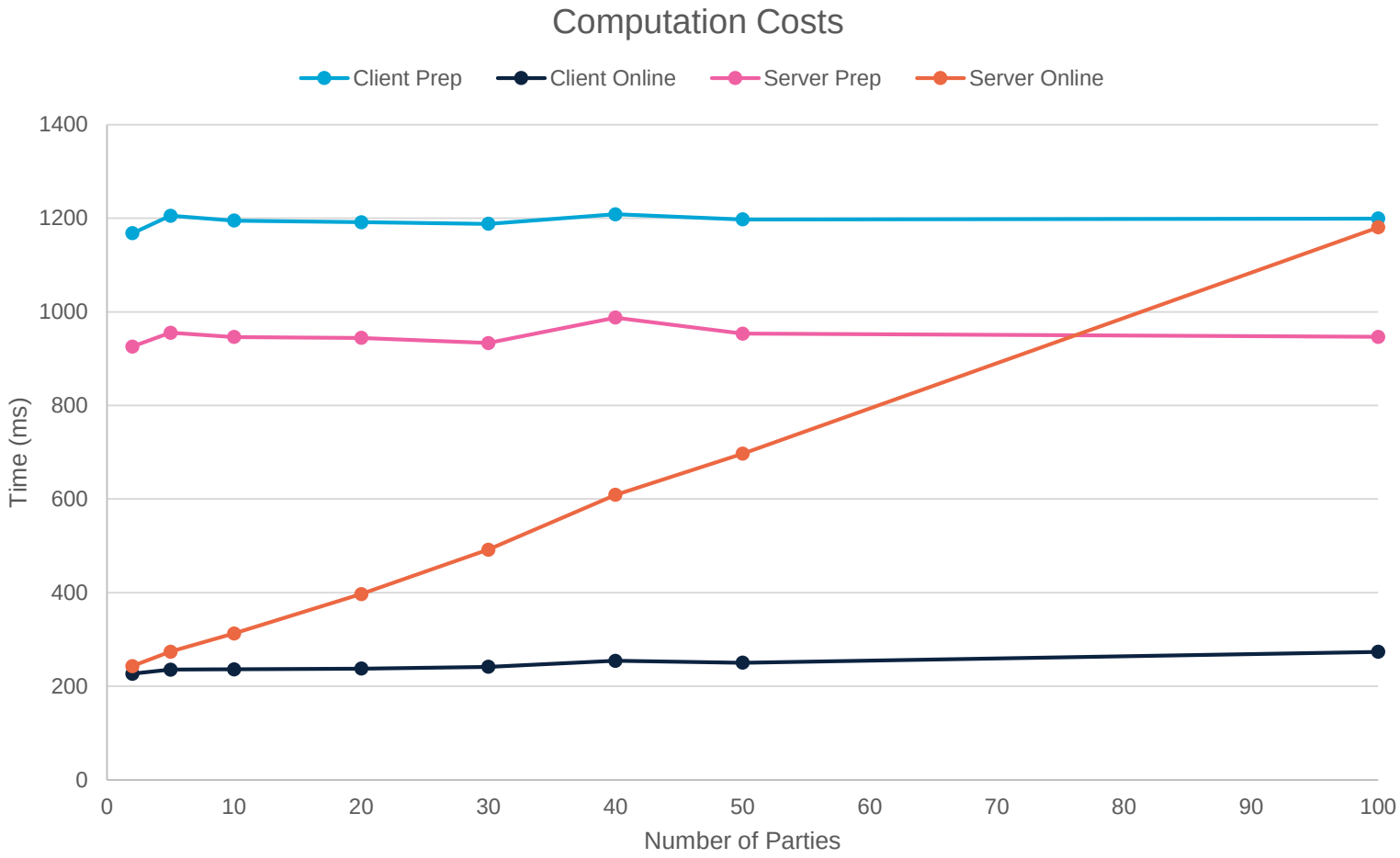
09-02-2026

Methodology: Stage 1 – Familiarization

Ruan et al. [6] results **(RQ1)**:

Computation time (in ms) per party for a run of the protocol involving various number of parties (including the server)

Set size clients: 256 elements
Set size server: 1024 elements
False positive rate: 2^{-7}

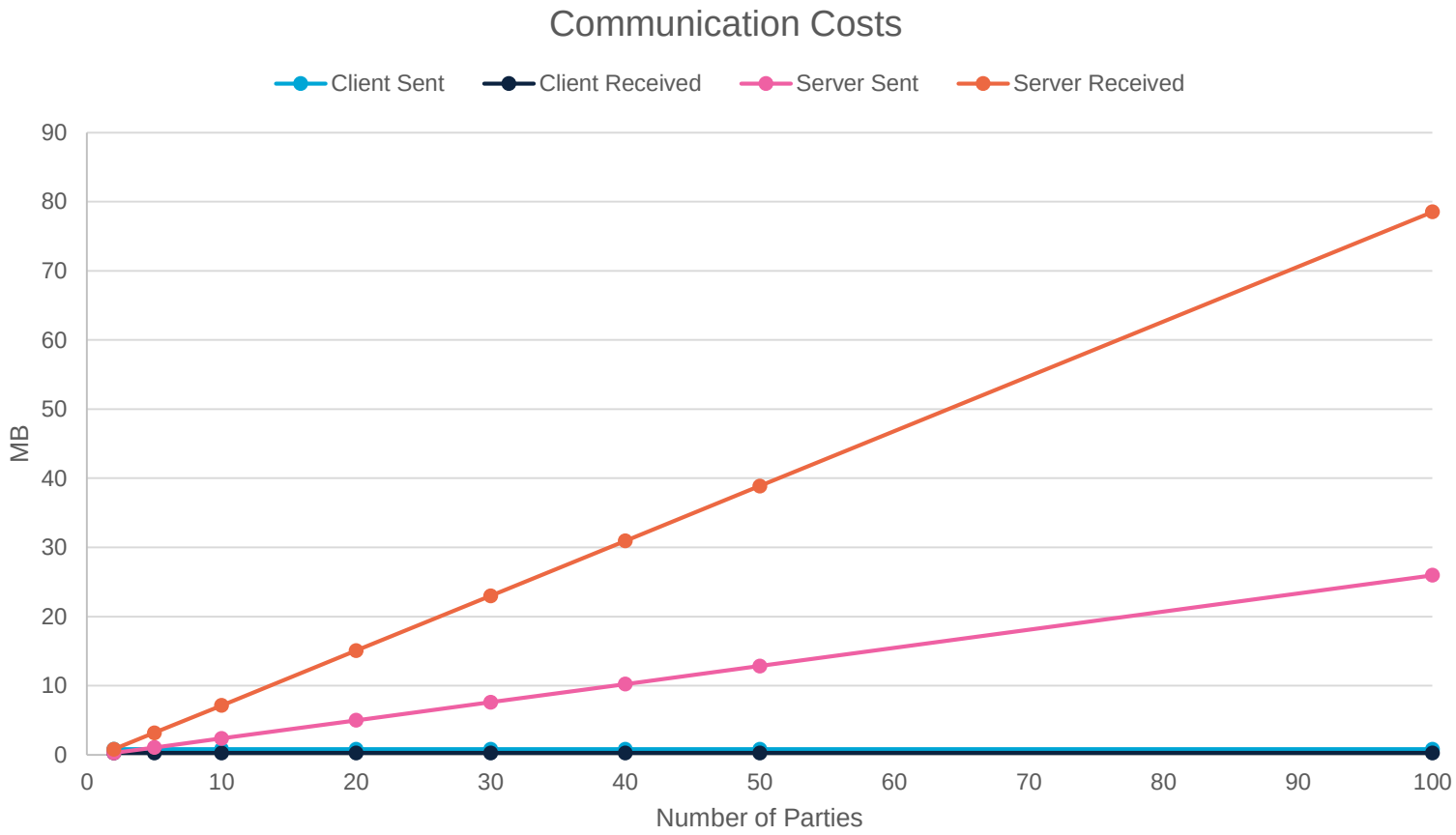


Methodology: Stage 1 – Familiarization

Ruan et al. [6] results **(RQ1)**:

Communication size (in MB) per party for a run of the protocol involving various number of parties (including the server)

Set size clients: 256 elements
Set size server: 1024 elements
False positive rate: 2^{-7}



Methodology: Stage 1 – Familiarization

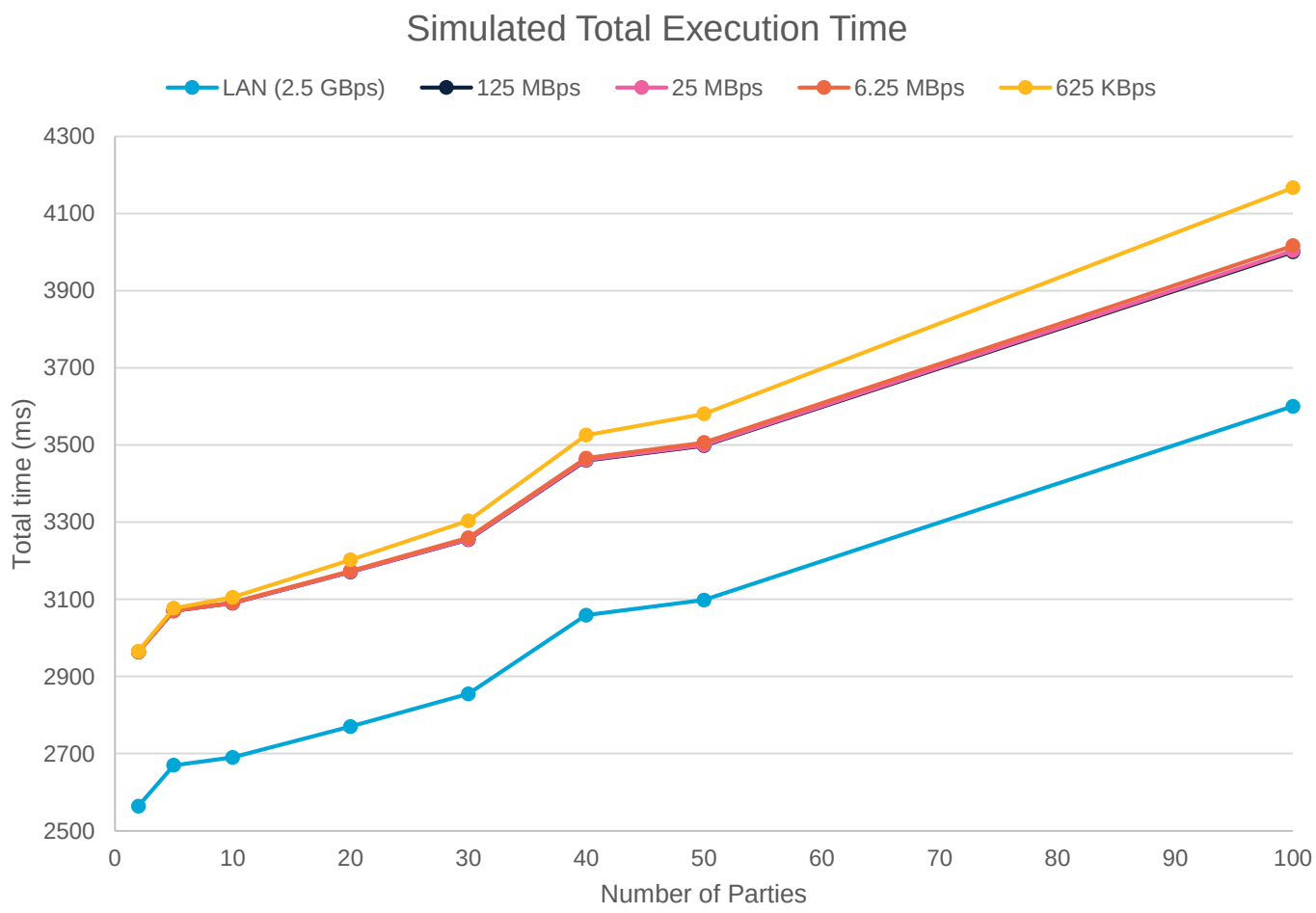
Ruan et al. [6] results **(RQ1)**:

Total time (in ms) for a run of the protocol involving various number of parties (including the server)

Set size clients: 256 elements
Set size server: 1024 elements
False positive rate: 2^{-7}

Total time calculated as:
Latency + Communication time + Computation Time

- **Latency:** *number of messages * hardcoded network latency*
- **Communication time:** *Bytes sent * hardcoded bandwidth*
- **Computation time:** plotted before



Methodology: Stage 1 – Familiarization

Why **Ruan et al. [6]** is vulnerable

to the attack in **Vos et al. [3]**:

In the UC framework, a protocol is considered secure if a "Real World" execution is computationally indistinguishable from an "Ideal World" execution where a trusted party computes the function. For approximate PSI, Vos et al. define the ideal functionality $\mathcal{F}_{waMPSI}$ (weakly approximate MPSI). In this ideal world, the trusted party returns the true intersection and adds false positives (elements not in the intersection) with a constant, random probability ϵ_{fp}. In the ideal world, whether a specific non-intersection element x is included in the output is a random event and does not depend on the specific value of x or its hash collisions.

In contrast, in the "Real World" execution of a Bloom filter-based protocol (abstracted as Π_{BF}), false positives are not random. They occur deterministically whenever an element x hashes to indices that are all set to 1 in the filter. Vos et al. construct a distinguisher, denoted as D_{FPs}, to exploit this discrepancy. The adversary selects an input set specifically designed to maximize false positives. Because the hash functions are public, the adversary can identify distinct elements $y \notin X_{client}$ such that all bins $h_1(y), \ldots, h_k(y)$ are set to 1 by the client's set.

When the adversary inputs such elements into the protocol, the behavior differs between the two worlds:

- **In the Real World (Π_{BF}):** The protocol checks the Bloom filter bins. Since the adversary chose the elements, the protocol returns a false positive with high probability.
- **In the Ideal World ($\mathcal{F}_{waMPSI}$):** The functionality checks if the element is in the intersection, which is not. It then includes the element only with the random probability ϵ_{fp}. Since ϵ_{fp} is usually negligible (e.g., 2^{-30}), the output is most likely empty.

The distinguisher observes the output. If the specific elements are present, it identifies the execution as the "Real World". This proves that Bloom filter-based protocols leak information about the set structure that the ideal functionality does not, making them insecure unless the parameters are so large that false positives almost never occur.

Methodology: Stage 1 – Familiarization

Why **Ruan et al. [6]** is vulnerable

to the attack in **Vos et al. [3]**:

Vos et al. classify the protocol by Ruan et al. as an instantiation of the abstract Bloom filter protocol Π_{BF}. While Ruan et al. use cryptographic primitives (threshold ElGamal encryption and Shamir secret sharing), the logic of the intersection operation remains the same as the unencrypted operations in Π_{BF}.

In the abstract functionality Π_{BF}, the output is the subset of the leader's elements that return "True" when queried against the combined Bloom filter of all other parties:

$$\text{Output}_{\Pi_{BF}} = \{x \in S_{\text{server}} \mid \text{contains}(\bigwedge_{i=1}^{t-1} \hat{X}_i, x)\} \tag{2.4}$$

Ruan's protocol implements this logic adding encryptions. The server computes a combined ciphertext c_j for an element x by homomorphically multiplying the responses from all clients:

$$c_j = \left(\prod_{i=1}^{t-1} c_{j,i}\right) \times w_j \tag{2.5}$$

where w_j is a blinded version of the server's element and $c_{j,i}$ is the encrypted value from client i at the hashed indices of x. The protocol considers an element in the intersection if and only if the decryption of c_j equals the blinded element w_j.

In Ruan's construction, the '0' bits in the Bloom filter are replaced by random values before encryption, while '1' bits are encrypted as they are. Due to the multiplicative homomorphic property of ElGamal, the product $\prod c_{j,i}$ will decrypt to 1 if and only if every component $c_{j,i}$ is an encryption of 1. If even one client has a randomized '0' at the queried index, the product becomes an encryption of a random value. Therefore, the cryptographic check in Ruan is equivalent to the boolean AND operation in Π_{BF}:

$$\text{Dec}(c_j) = w_j \iff \forall i : \text{contains}(\hat{X}_i, x) \tag{2.6}$$

Hence, Ruan's protocol outputs the same set of elements as Π_{BF}, including all false positives.