Learning from Functionality Outputs: Private Join and Compute in the Real World

Francesca Falzon
Joint work with Tianxin Tang
Appeared at USENIX 2025

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

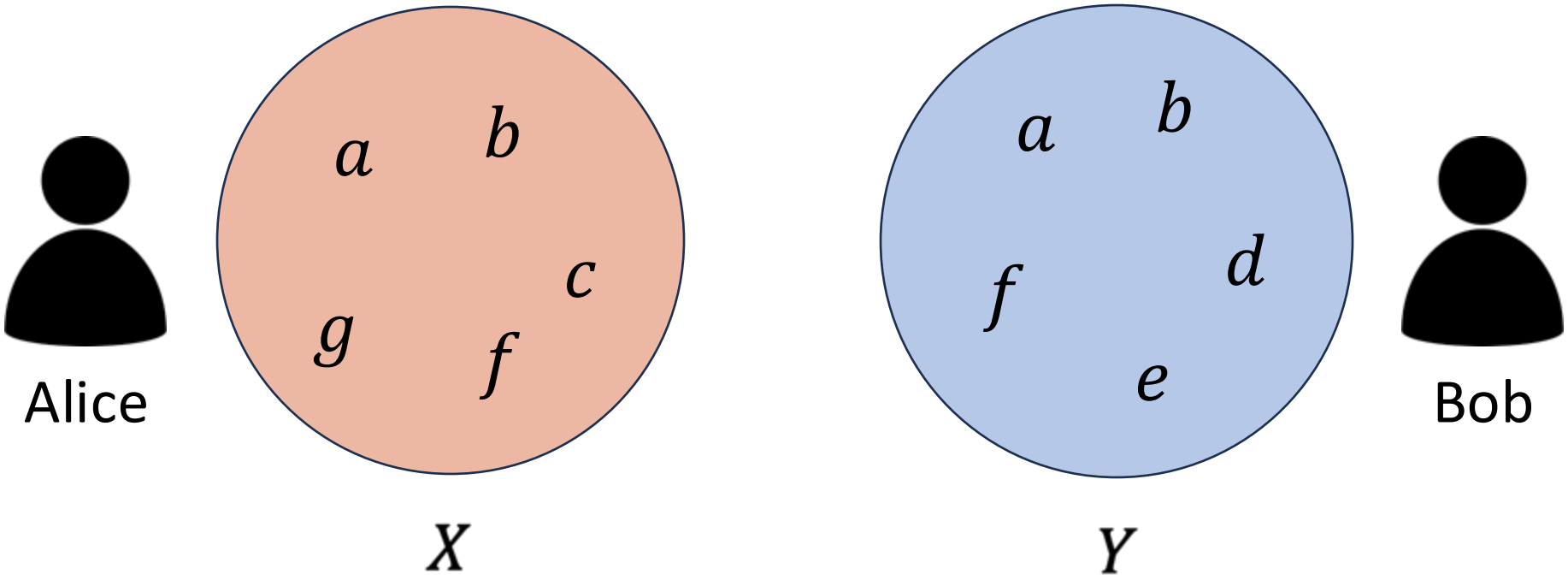
armasuisse Science and Technology

ETH zürich

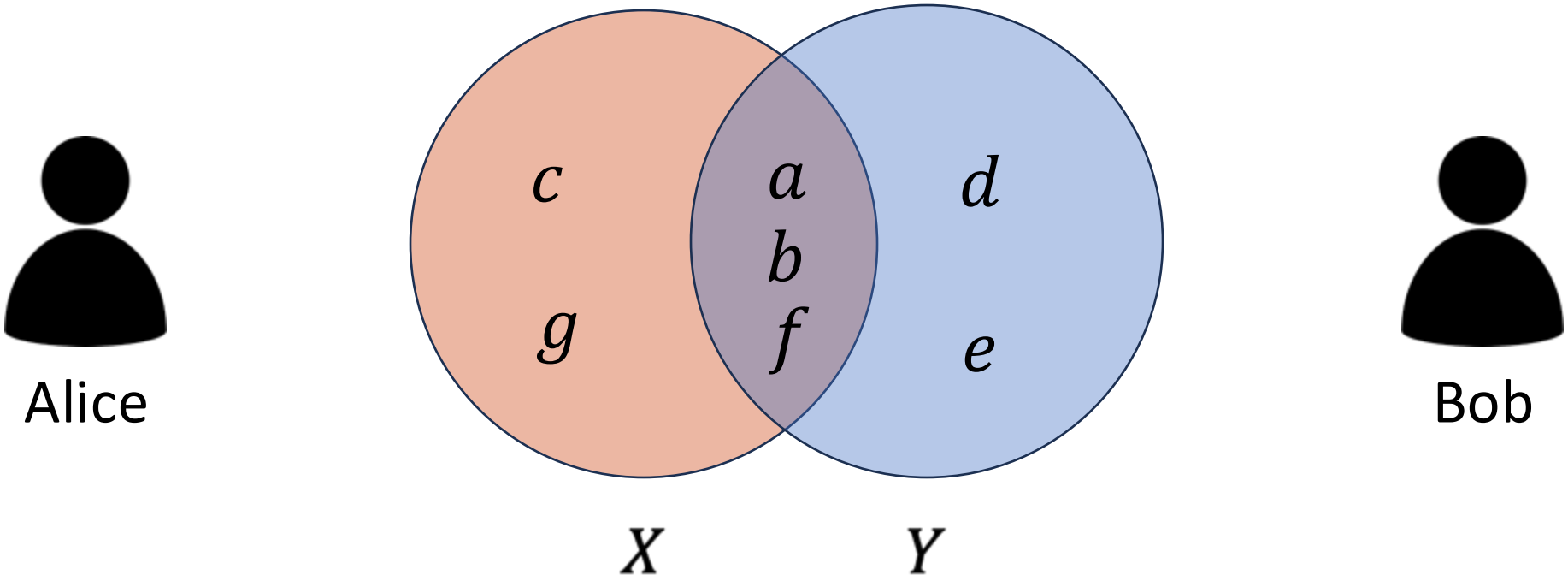
APPLIED CRYPTO GROUP

TU/e

Private Set Intersection (PSI)



Private Set Intersection (PSI)



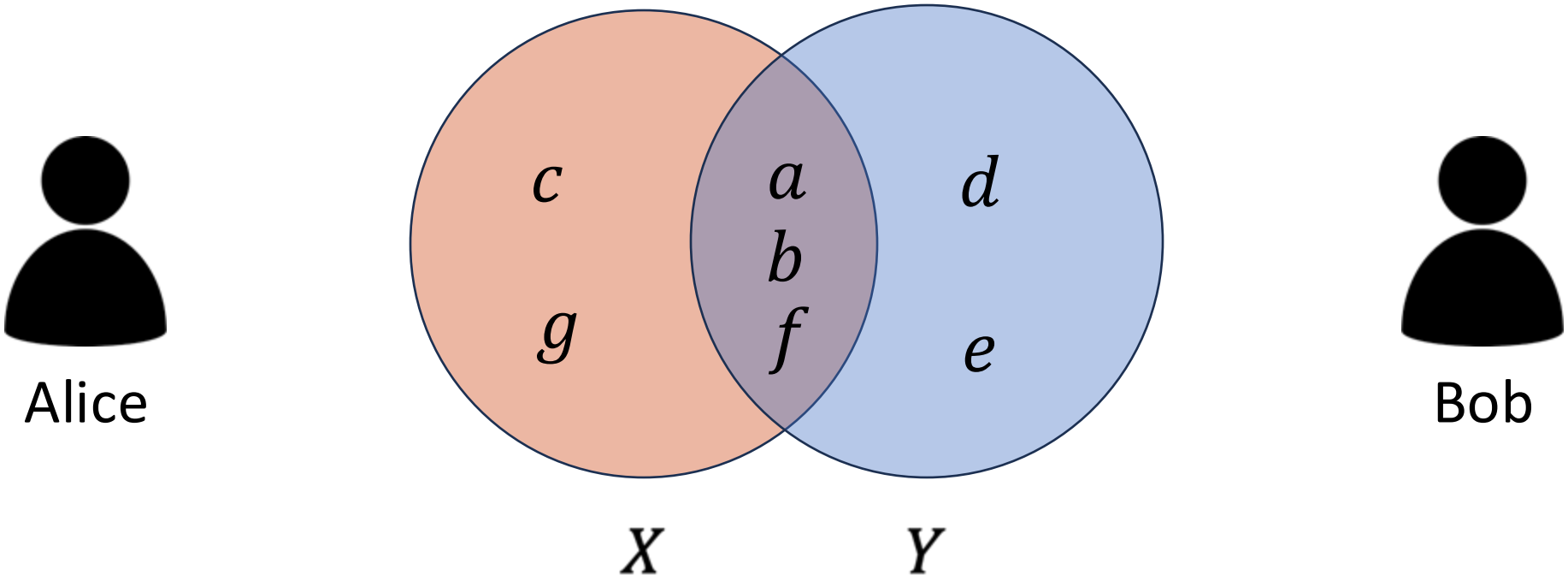
Goal: Compute the intersection
- Privately
- Efficiently

Output: $X \cap Y = \{a, b, f\}$

Private Set Intersection (PSI) Applications

- Mobile contact discovery
- Bot net detection
- Testing of sequenced human genomes
- Ad conversion

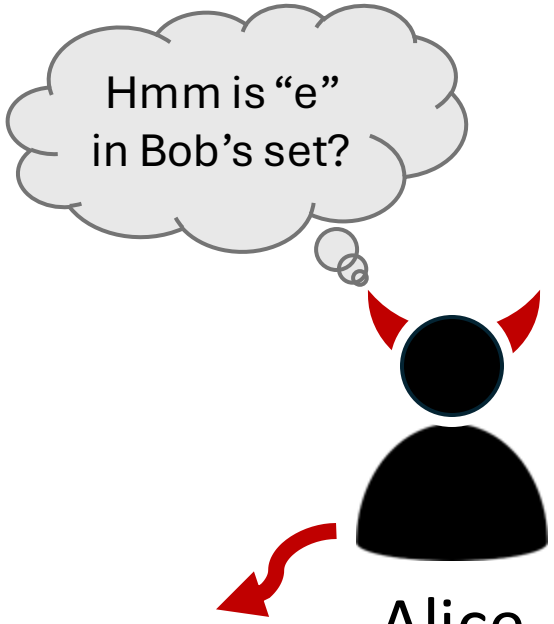
Private Set Intersection Cardinality (PSI-CA)

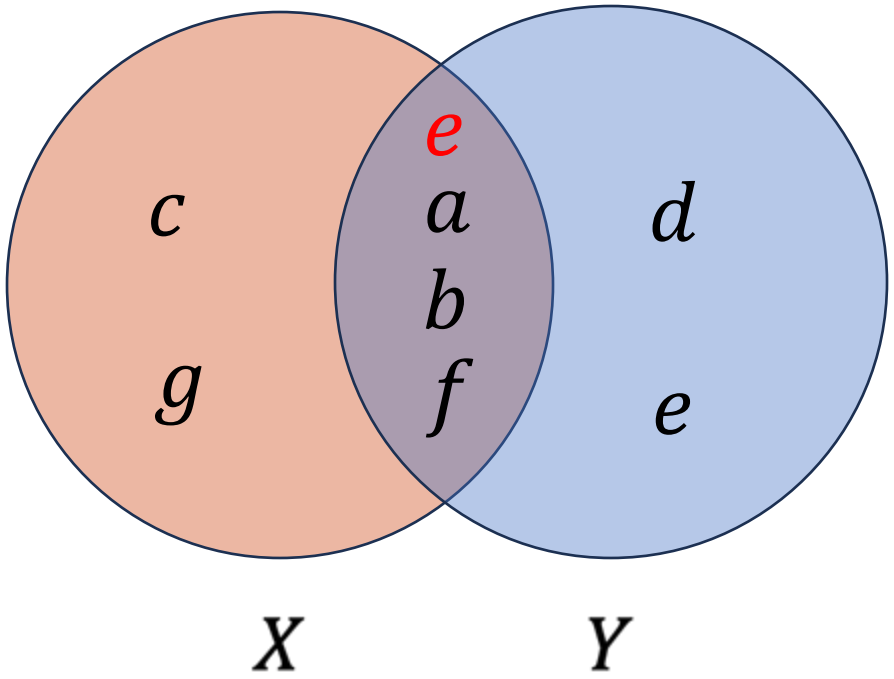


Goal: Compute the cardinality
- Privately
- Efficiently

Output: $|X \cap Y|=3$

Private Set Intersection Cardinality (PSI-CA)





Output: $|X \cap Y|=3$

Case Study: Apple PSI

Private set intersection (PSI) allows Apple to learn if an image hash matches the known CSAM image hashes, without learning anything about image hashes that do not match. PSI also prevents the user from learning whether there was a match.

https://9to5mac.com/2021/08/05/apple-announces-new-protections-for-child-safety-imessage-safety-icloud-photo-scanning-more/

The Apple PSI System

Abhishek Bhowmick
Apple Inc.

Dan Boneh
Stanford University

Steve Myers
Apple Inc.

Kunal Talwar
Apple Inc.

Karl Tarbe
Apple Inc.

July 29, 2021

Abstract

This document describes the constraints that drove the design of the Apple *private set intersection* (PSI) protocol. Apple PSI makes use of a variant of PSI we call *private set intersection with associated data* (PSI-AD), and an extension called *threshold private set intersection with associated data* (tPSI-AD). We describe a protocol that satisfies the constraints, and analyze its security. The context and motivation for the Apple PSI system are described on the main project site.

A Concrete-Security Analysis of the Apple PSI Protocol

Mihir Bellare

Department of Computer Science and Engineering
University of California, San Diego

July 30, 2021

A Review of the Cryptography Behind the Apple PSI System

Benny Pinkas

Dept. of Computer Science
Bar-Ilan University

July 9, 2021

CSAM

Apple confirms that it has stopped plans to roll out CSAM detection system

Filipe Espósito | Dec 7 2022 - 10:28 am PT | 57 Comments

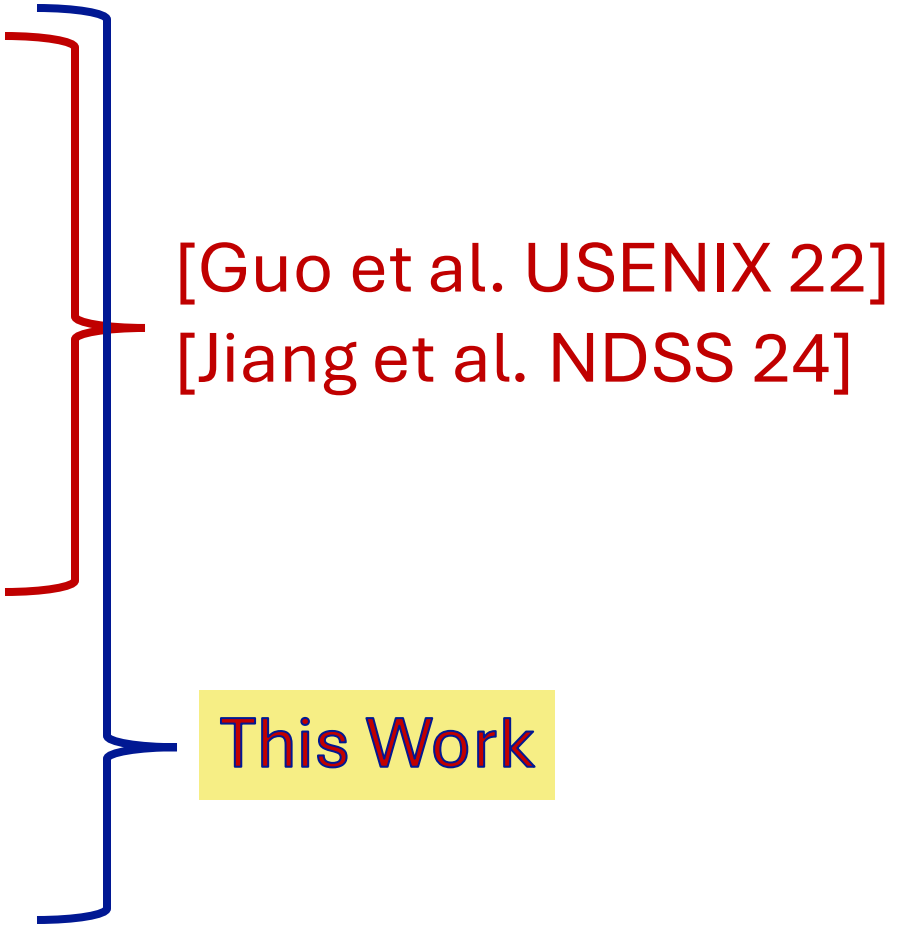
Back in 2021, Apple announced a number of **new child safety features**, including Child Sexual Abuse Material (CSAM) detection for iCloud Photos. However, the move was widely criticized due to privacy concerns. After putting it on hold indefinitely, Apple has now confirmed that it has stopped its plans to roll out the CSAM detection system.

Secure Multi-party Computation (MPC)

- Provides input privacy
- Only considers that the computation is private
  - Guarantees that nothing is learned about the input from the transcripts
  - Excludes information learned from the function output
- What's the criteria that a functionality is fine?
  - Not really clear!

Not the End of the Story

- Facebook's Private-ID
- PSI-CA
  - Size of the intersection
- PSI-SUM
  - Sum of the associated values in the intersection
- PSI-Inner product
  - AKA Private Join and Compute
  - Generalizes the above



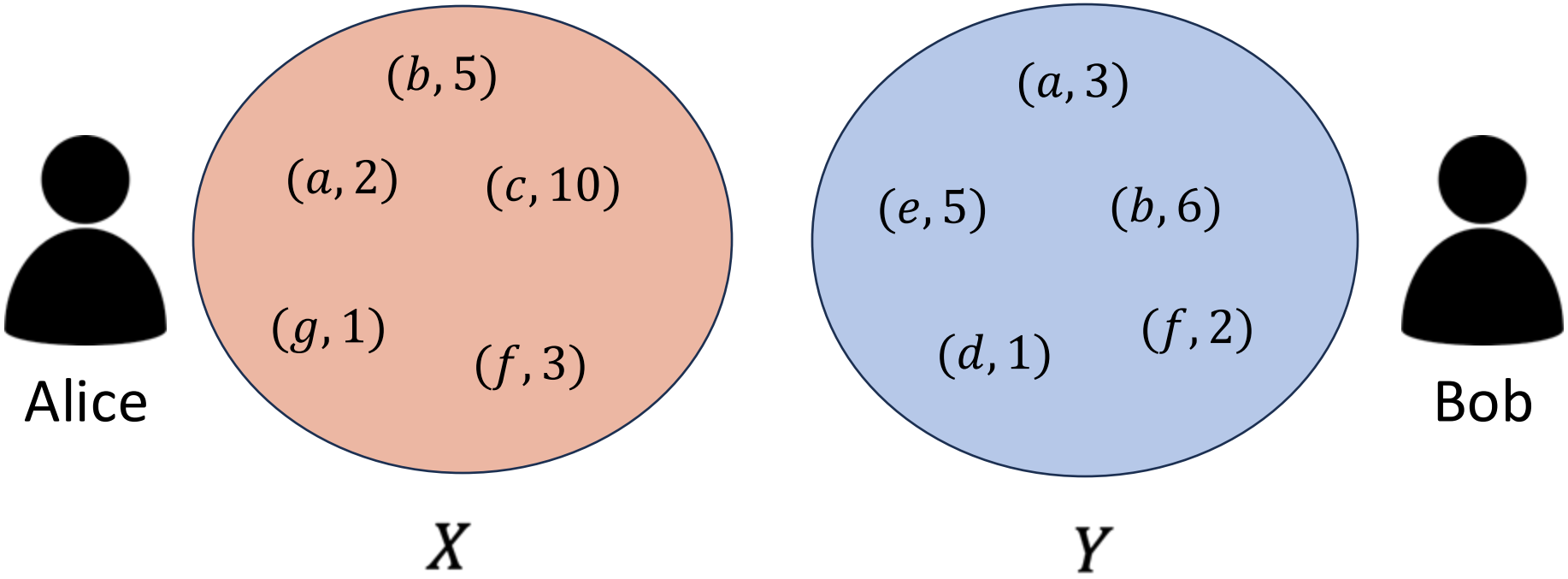
Our Contributions

- We describe 4 attacks against the PJC functionality
  - #1 Search Tree Attack
  - #2 Statistical Inference
  - #3 and #4 Signal Processing Techniques
- We implement and evaluate our attacks

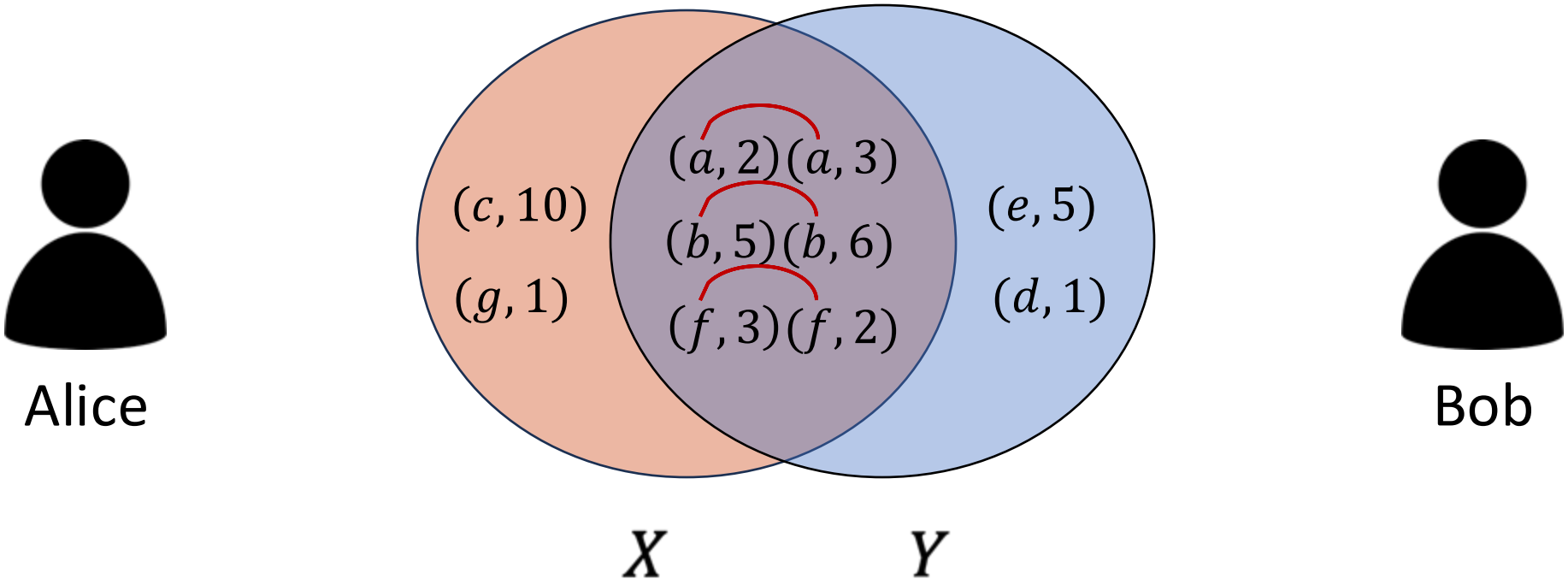
Overview

- Introduction
- **Background**
- Search Tree Attack
- Statistical and Algebraic Attacks
- Experiments
- Conclusion

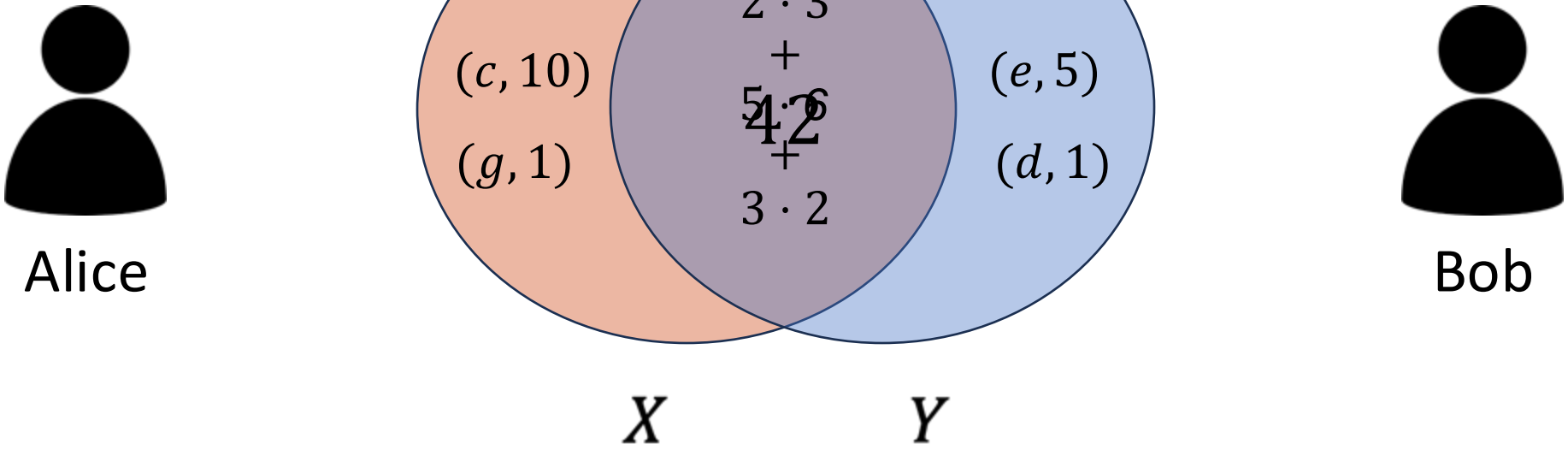
Google's Private Join and Compute (PJC)



Google's Private Join and Compute (PJC)



Google's Private Join and Compute (PJC)



Output: $\sum_{\substack{(k,v)\in X \\ (k',v')\in Y}} 1\{k = k'\} \cdot v \cdot v'$

Disclaimer

Maliciously Chosen Inputs

We note that our protocol does not authenticate that parties use "real" input, nor does it prevent them from arbitrarily changing their input. We suggest careful analysis of whether any party has an incentive to lie about their inputs. This risk can also be mitigated by external enforcement such as code audits.

Leakage from the Intersection-Sum.

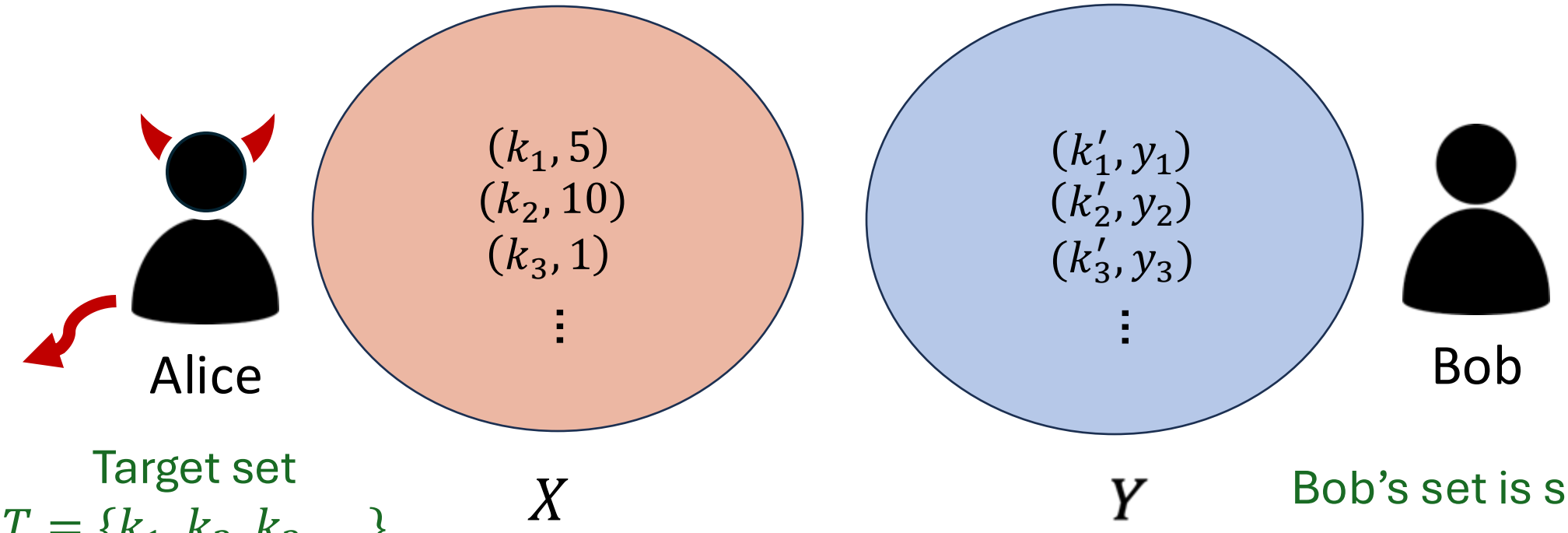
While the Private Join and Compute functionality is supposed to reveal only the intersection-size and intersection-sum, it is possible that the intersection-sum itself could reveal something about which identifiers were in common.

For example, if an identifier has a very unique associated integer values, then it may be easy to detect if that identifier was in the intersection simply by looking at the intersection-sum. One way this could happen is if one of the identifiers has a very large associated value compared to all other identifiers. In that case, if the intersection-sum is large, one could reasonably infer that that identifier was in the intersection. To mitigate this, we suggest scrubbing inputs to remove identifiers with "outlier" values.

Another way that the intersection-sum may leak which identifiers are in the intersection is if the intersection is too small. This could make it easier to guess which combination of identifiers could be in the intersection in order to yield a particular intersection-sum. To mitigate this, one could abort the protocol if the intersection-size is below a certain threshold, or to add noise to the output of the protocol.

(Note that these mitigations are not currently implemented in this open-source library.)

Attack Setting



Goal = learn which keys in T are also in Y and what their values are.

Overview

- Introduction
- Background
- **Search Tree Attack**
- Statistical and Algebraic Attacks
- Experiments
- Conclusion

Basic attack

- Assume Bob's values come from a small domain
  - $minval = 1, maxval = 100$
- We can recover n keys and their associated values using a single query!

Basic attack

- Domain separation
- Target set = $\{k_1, k_2, k_3\}$
- Adversarial input: $(k_1, 10^0), (k_2, 10^3), (k_3, 10^6)$

output = $4{,}070{,}000 = 4 \cdot 10^6 + 70 \cdot 10^3 + 0 \cdot 10^0$

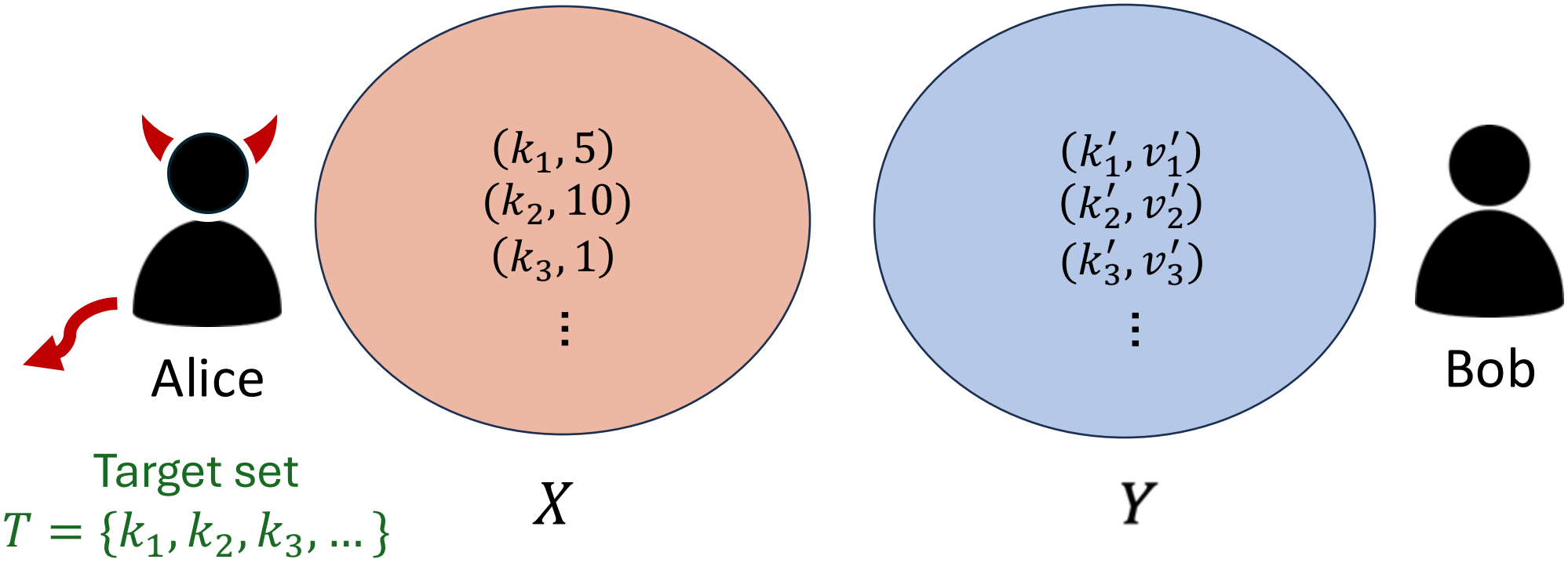
Attack #1: Search Tree Attack

- There is a limit on the length of a data type in implementations
  - E.g., unsigned long long type
- **Solution:** Search the space in a binary-search-like way
  - Partition the search space

Overview

- Introduction
- Background
- Search Tree Attack
- **Statistical and Algebraic Attacks**
- Experiments
- Conclusion

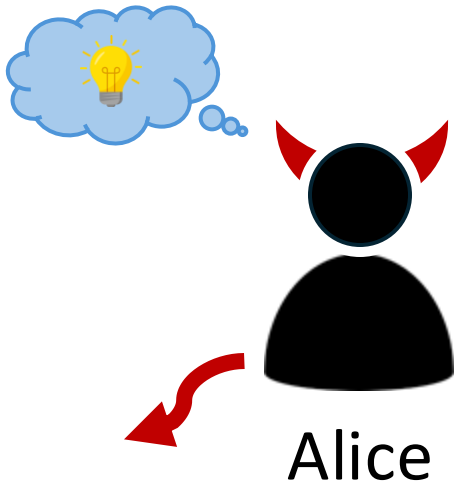
An algebraic representation?

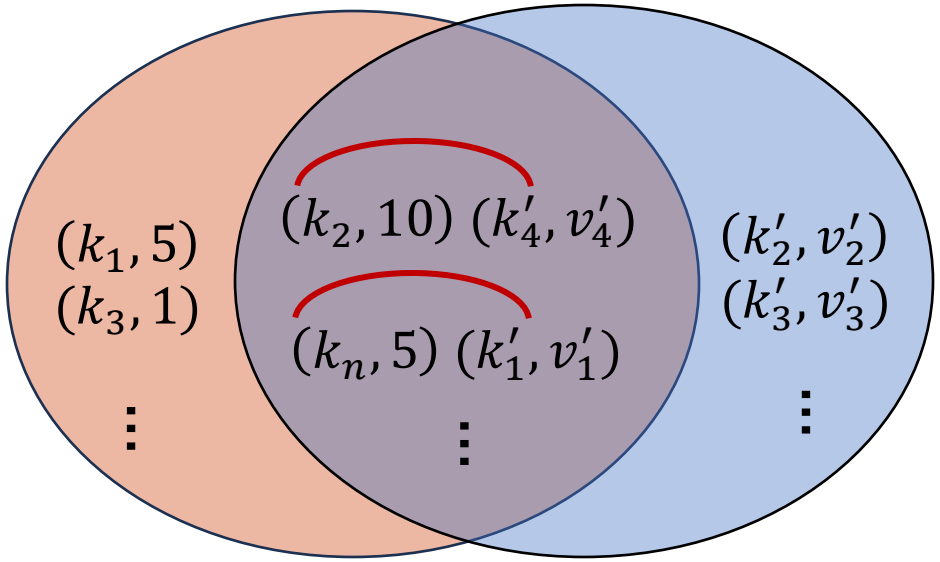


Target set

$T = \{k_1, k_2, k_3, \dots\}$

An algebraic representation?

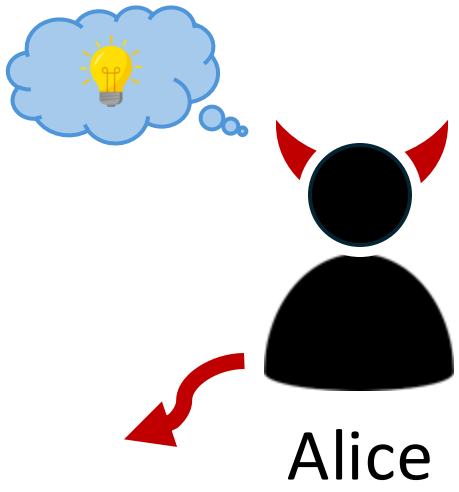


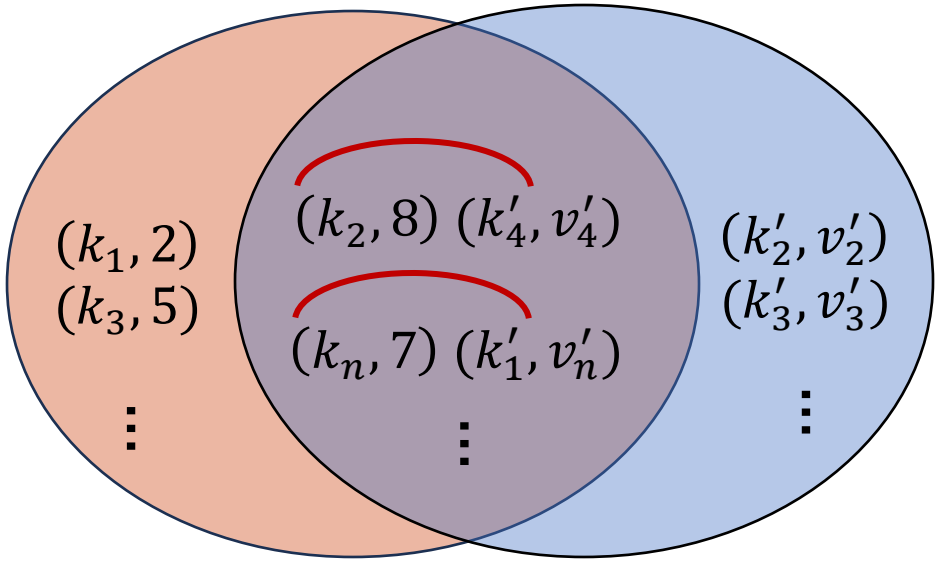


$57 = 5y_1 + 10y_2 + \cdots + 5y_n$

Output: 57

An algebraic representation?





$$57 = 5y_1 + 10y_2 + \cdots + 5y_n$$
$$45 = 2y_1 + 8y_2 + \cdots + 7y_n$$

Output: 45

More Formally

Inputs: $X = \{(k_i, v_i)\}_{i \in [n]}$ and $Y = \{(k'_j, v'_j)\}_{j \in [m]}$

Output: $\sum_{\substack{(k,v) \in X \\ (k',v') \in Y}} 1\{k = k'\} \cdot v \cdot v'$

- If k_2 is contained in Y, then:

  $v_1 \cdot v'_{j_1} + v_2 \cdot v'_{j_2} + \cdots + v_n \cdot v'_{j_n}$ = output

- Otherwise, the corresponding term becomes 0

  $v_1 \cdot v'_{j_1} + \quad 0 \quad + \cdots + v_n \cdot v'_{j_n}$ = output

- We can thus write down a linear constraint of n unknowns $(y_1, \ldots, y_n)$

More Formally

Inputs: $X = \{(k_i, v_i)\}_{i \in [n]}$ and $Y = \{(k'_j, v'_j)\}_{j \in [m]}$

Output: $\sum_{\substack{(k,v) \in X \\ (k',v') \in Y}} 1\{k = k'\} \cdot v \cdot v'$

Let $\mathbf{s} = (v'_{j_1}, \ldots, v'_{j_n})$

Thus, if $\mathbf{s}$ is fixed in between the protocol invocations

$A_1 = (v_1^{(1)}, v_3^{(1)}, \ldots, v_n^{(1)})$

$A_2 = (v_1^{(2)}, v_3^{(2)}, \ldots, v_n^{(2)})$

$\vdots$

$A_q = (v_1^{(q)}, v_3^{(q)}, \ldots, v_n^{(q)})$

$$\mathbf{A} \quad \mathbf{s} = \mathbf{b}$$

q x n n x 1 q x 1

Baseline Attack

- For n unknowns, n linearly independent equations are needed for Gaussian elimination.
- Or we simply probe each of the n entries by setting one value to something non-zero one and the rest to zero.

How about $q < n$?

- Requires solving an under-determined linear system!
- There are an infinite # of solutions :/

Attack #2: Maximum Likelihood Estimation

- Assume known distribution (e.g., Normal distribution)
- Output the most likely solution

$$\begin{aligned} &\text{minimize } (\tilde{\mathbf{s}} - \boldsymbol{\mu})^T (\tilde{\mathbf{s}} - \boldsymbol{\mu}) && \text{subject to} \\ &\quad \text{minval} \leq \tilde{s}_i \leq \text{maxval} && and \\ &\quad A\tilde{\mathbf{s}} = \mathbf{b}. \end{aligned}$$

- If the values in Y lie within the range $[-100, 100]$, running $0.95n$ queries, still incurs ~7 in ℓ_1-loss.

What if the intersection size is known?

- Intersection size $= k$, target set size $= n$
- Suppose ($k \ll n$) i.e., k-sparse

Attacks #3 and #4: DFT and CS

- Discrete Fourier Transform (DFT)
  - Need to know the exact k
  - Not robust to output noise
- Compressed Sensing (CS)
  - Robust to small-scale noises and sparsity defect (need a rough estimate of the intersection size)
  - In essence requires matrix A to satisfy a particular property that ensures a unique solution.

Overview

- Introduction
- Background
- Search Tree Attack
- Statistical and Algebraic Attacks
- **Experiments**
- Conclusion

Experimental Evaluation

- Tree / basic attack: can scale n up to 1 million.
- The remaining experiments were tested on $n \leq 10,000$:
  - MLE: $q = 0.95n$ $(k = n)$
  - DFT: $q = 2k$
  - CS:
    - concrete query bound of $2k\log_2(\frac{n}{k})$
    - $q = 0.6n$ for $k < 0.2n$

Overview

- Introduction
- Background
- Search Tree Attack
- Statistical and Algebraic Attacks
- Experiments
- **Conclusion**

Disclosure

README | Code of conduct | Contributing | Apache-2.0 license | Security

Leakage from Intersection-Sum with Cardinality.

While the Private Join and Compute functionality is supposed to reveal only the intersection-size and intersection-sum, it is possible that these outputs themselves could reveal something about the inputs.

For example, if an identifier has a very unique associated integer values, then it may be easy to detect if that identifier was in the intersection simply by looking at the intersection-sum. One way this could happen is if one of the identifiers has a very large associated value compared to all other identifiers. In that case, if the intersection-sum is large, one could reasonably infer that that identifier was in the intersection.

Another way that the intersection-sum may leak which identifiers are in the intersection is if the intersection is too small. This could make it easier to guess which combination of identifiers could be in the intersection in order to yield a particular intersection-sum.

Finally, a sequence of computations on the same or related input data could allow inferring more about the inputs.

Possible mitigations include requiring the inputs to be sufficiently large and sufficiently different across different executions, pruning outlier values, adding differential privacy noise to the outputs, and aborting if the intersection size is too small.

(Note that these mitigations are not currently implemented in this open-source library.)

Works including Guo et al ("Birds of a Feather Flock Together", USENIX '22) systematically study the leakage of intersection-sum with cardinality, and also explore mitigations. We refer readers to these works for further discussion.

Countermeasures and future work

- Preventing malicous input
  - Certified PSI
- Rate limiting / query budgets
- Adding differential privacy

Thank you!
Questions?

Find our paper here!