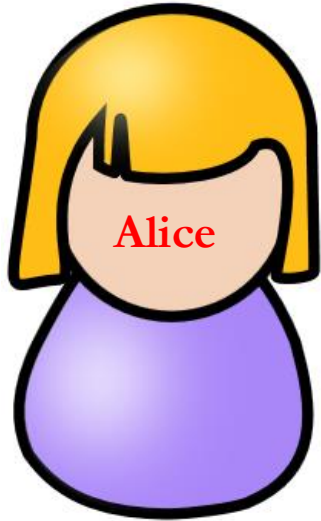


# Enhancing Private Set Intersection for Broader Applications

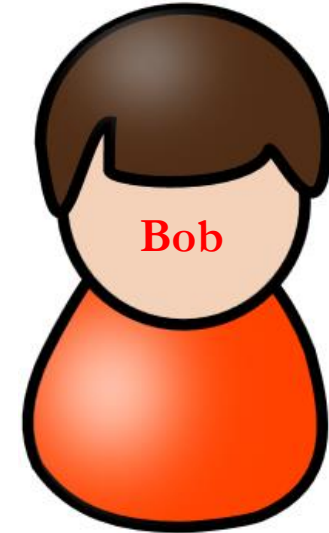
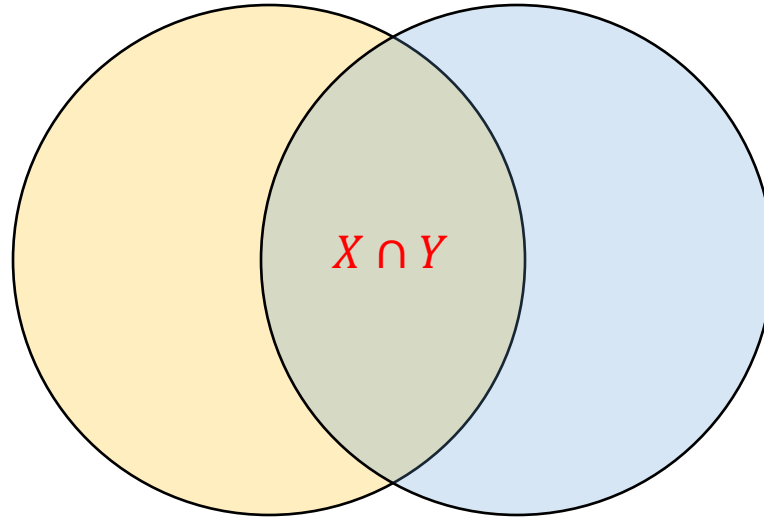
**Peihan Miao**



# Private Set Intersection (PSI)

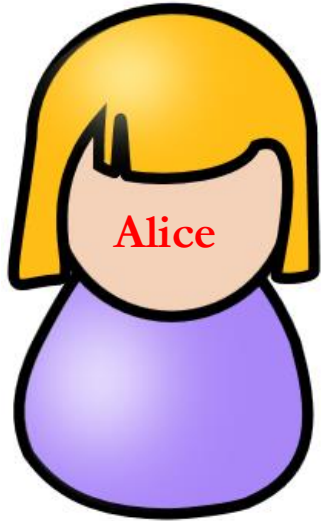


$$X = \{x_1, x_2, \dots, x_n\}$$

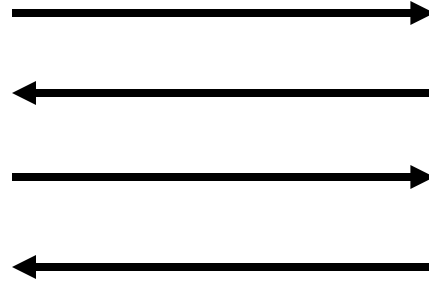


$$Y = \{y_1, y_2, \dots, y_m\}$$

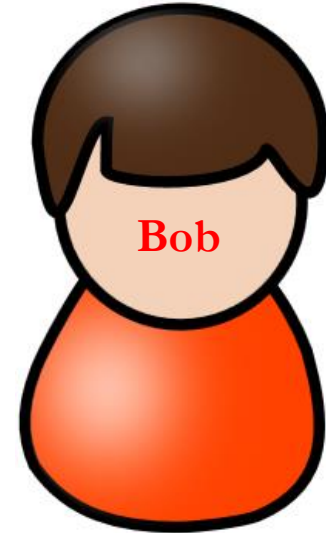
# Private Set Intersection (PSI)



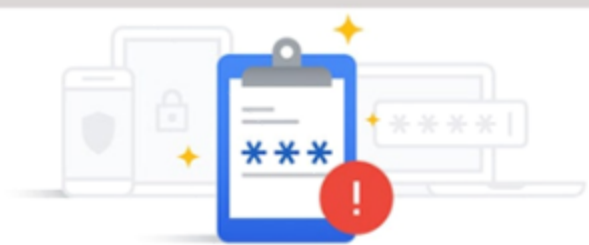
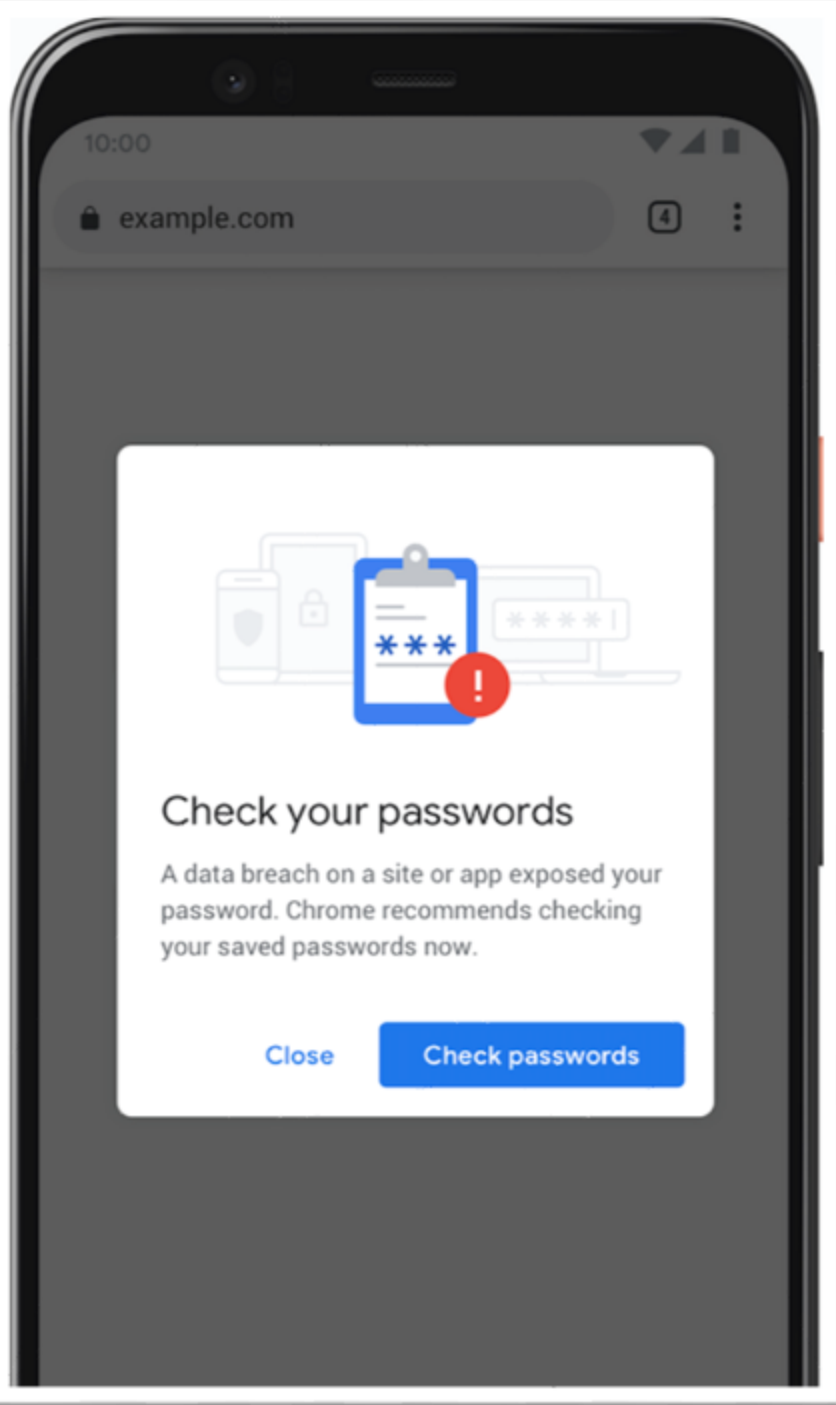
$$X = \{x_1, x_2, \dots, x_n\}$$






$$X \cap Y$$



$$Y = \{y_1, y_2, \dots, y_m\}$$



We analysed your saved passwords and found the following issues

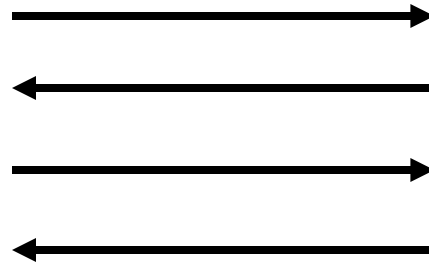
	<b>143 compromised passwords</b> Change these passwords now	▼
	<b>474 reused passwords</b> Create unique passwords	▼
	<b>487 accounts using a weak password</b> Create strong passwords	▼

# Application: Password Breach Detection



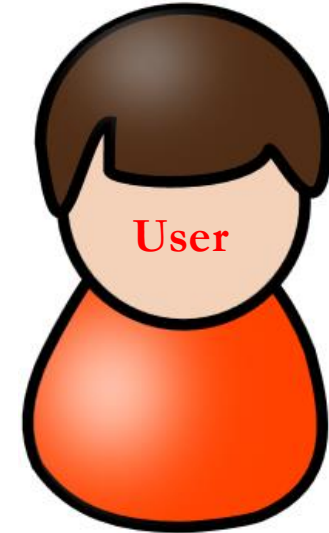
(user1, pwd1)
(user2, pwd2)
(user3, pwd3)
(user4, pwd4)
(user5, pwd5)

Passwords leaked online



(user2, pwd2)
(user2, pwd7)

User's passwords



## < Back Security Recommendations

### Detect Compromised Passwords



iPhone can securely monitor your passwords and alert you if they appear in known data leaks.

#### SECURITY RECOMMENDATION



### Compromised, easily guessed password

This password has appeared in a data breach, which puts this account at high risk of compromise. You should change your password immediately.

[Change Password on Website](#)

## Apple Platform Security

Communities Contact

Search this guide

[Table of Contents](#)

## Password Monitoring

Password Monitoring is a feature that matches passwords stored in the user's Password AutoFill keychain against a continuously updated and curated list of passwords known to have been exposed in leaks from different online organizations. If the feature is turned on, the monitoring protocol continuously matches the user's Password AutoFill keychain passwords against the curated list.

### How monitoring works

The user's device continuously performs round robin checks on a user's passwords, querying on an interval that's independent of the user's passwords. This helps ensure that verification states remain up to date with the current curated list of leaked passwords. To help prevent leakage of information related to how many unique passwords a user has, requests are batched and performed in parallel. A fixed number of passwords are verified in parallel on each check, and should the user have fewer than this number, random passwords are generated and added to the queries to make up the difference.

### How passwords are matched

Passwords are matched in a two-part process. The most commonly leaked passwords are contained within a local list on the user's device. If the user's password occurs on this list, the user is immediately notified without any external interaction. This is designed to ensure that no information is leaked about the passwords a user has that are most at risk due to a password breach.

If the password isn't contained on the most frequent list, it's matched against less frequently leaked passwords.

### Comparing users' passwords against a curated list

To verify whether a password not present in the local list is a match involves some interaction with Apple servers. To help ensure that legitimate users' passwords aren't sent to Apple, a form of cryptographic **private set intersection** deployed that compares the users' passwords against a large set of leaked passwords. This is designed to ensure that for passwords less at risk of breach, little information is shared with Apple. For a user's password, this information is limited to a 15-bit prefix of a cryptographic hash. The removal of the most frequently leaked passwords from this interactive process, using the local list of most



## See if you've been part of an online data breach.

Find out what hackers already know about you. Learn how to stay a step ahead of them.

Check for Breaches

Search for your email address in public data breaches going back to 2007.

### LATEST BREACH ADDED



#### Poshmark

Breach added:

September 2, 2019

Compromised data:

Passwords, Email addresses



We monitor your saved passwords for issues. [Learn how](#)



Microsoft Edge detected that the password for 1 site has been leaked

Change your passwords to secure your accounts.

View details

Not now

# Technology preview: Private contact discovery for Signal

moxie0 on 26 Sep 2017

At Signal, we've been thinking about [the difficulty of private contact discovery](#) for a long time. We've been working on strategies to improve our current design, and today we've [published a new private contact discovery service](#).

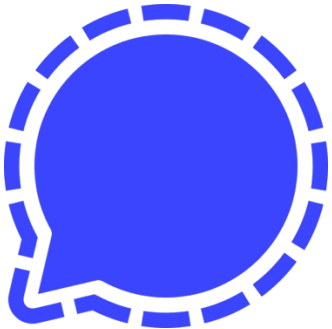
Using this service, Signal clients will be able to efficiently and scalably determine whether the contacts in their address book are Signal users **without revealing the contacts in their address book to the Signal service**.

## Trust but verify

Of course, what if that's not the source code that's actually running? After all, we could surreptitiously modify the service to log users' contact discovery requests. Even if we have no motive to do that, someone who hacks the Signal service could potentially modify the code so that it logs user contact discovery requests, or (although unlikely given present law) some government agency could show up and require us to change the service so that it logs contact discovery requests. More fundamentally for us, we simply don't want people to have to trust us. That's not what privacy is about.

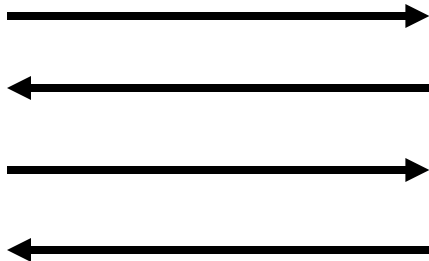
Doing better is difficult. There are a [range of options that don't work](#), like using bloom filters, encrypted bloom filters, sharded bloom filters, private information retrieval, or [private set intersection](#). What if, instead, there were just a way for clients to verify that the code running on our servers was the code they wanted to be running on our servers, and not something that we or a third party had modified?

# Application: Mobile Contact Discovery



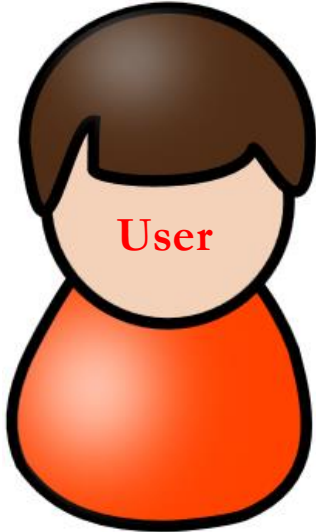
Adam
Grace
David
Eve
Charlie

Registered users



Eric
Frank
Charlie
Grace

Contact list



# Google takes the PIS out of advertising: New algo securely analyzes shared encrypted data sets without leaking contents

Plus: MongoDB crams end-to-end crypto into database tech

 [Thomas Claburn](#)

Wed 19 Jun 2019 21:47 UTC

Google on Wednesday released source code for a project called Private Join and Compute that allows two parties to analyze and compare shared sets of data without revealing the contents of each set to the other party.

This is useful if you want to see how your private encrypted data set of, say, ad-clicks-to-sales conversion rates, correlates to someone else's encrypted conversion rate data set without disclosing the actual numbers to either side.

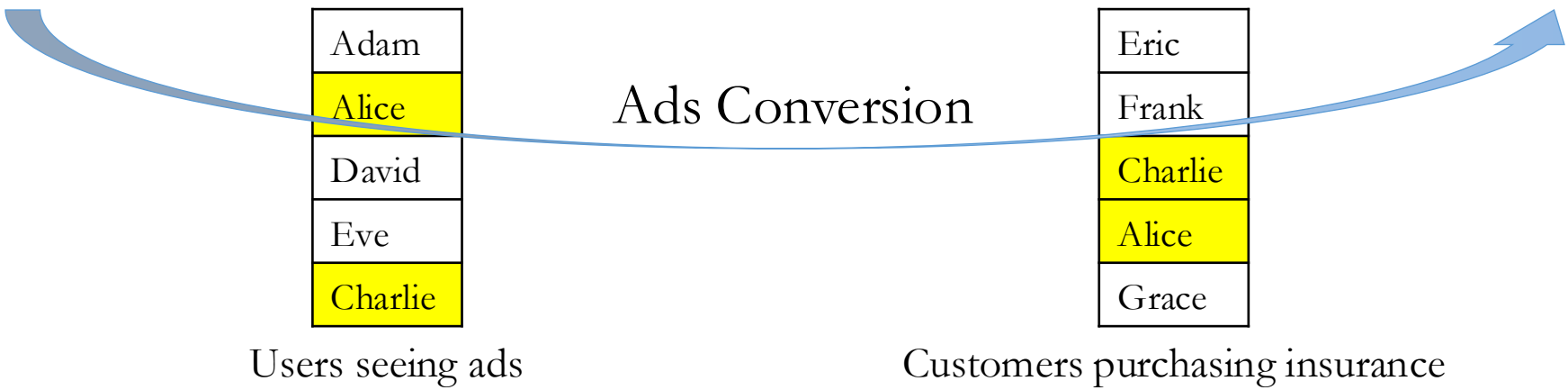
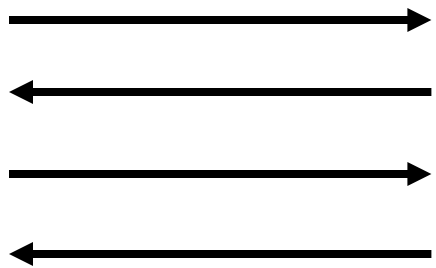
This particular technique is a type of secure multiparty computation that builds upon a cryptographic protocol called **Private Set-Intersection (PSI)**. Google employs this approach in a Chrome extension called [Password Checkup](#) that lets users test logins and passwords against a dataset of compromised credentials without revealing the query to the internet goliath.

[Private Join and Compute](#), also known as Private Intersection-Sum (PIS), takes PSI further by hiding the data that represents the intersection of the two data sets and revealing only the results of calculations based on the data.

The technique is described in [a research paper](#), "On Deploying Secure Computing Commercially: Private Intersection-Sum Protocols and their Business Applications," penned by nine Google researchers: Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung.

The paper describes how PIS can be computed using three cryptographic protocols: Random Oblivious Transfer, encrypted Bloom filters, and Pohlig–Hellman double masking.

# Application: Ads Conversion Analysis



# J.P. Morgan Privacy-Preserving Inventory Matching

Posted Jun 27, 2023 • Updated May 15, 2024



By Darya Kaviani

1 min read

Inventory matching is a mechanism for trading financial stocks in which buyers and sellers are paired. In the centralized setting, parties must share their order to buy or sell a stock, accompanied by the quantity. This is sensitive information that could cause adverse price movements prematurely, swaying prices adversely before a transaction finalizes. Prime Match allows clients to match their orders with reduced market impact while maintaining privacy through MPC. Prime Match runs in production at J.P Morgan, a large U.S. bank.

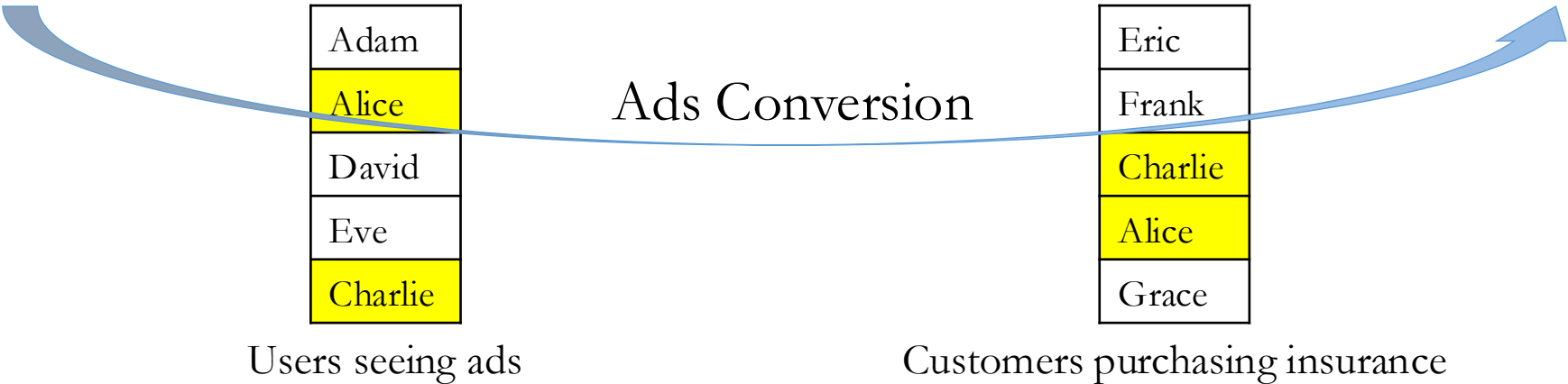
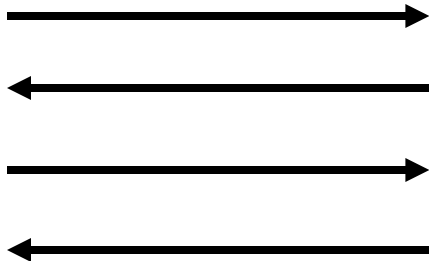
# Frameworks / Approaches

- Diffie-Hellman [Huberman-Franklin-Hogg'99, Jarecki-Liu'10, Ion-Kreuter-Nergiz-Patel-Saxena-Seth-Raykova-Shanahan-Yung'20, ...]
- Oblivious Polynomial Evaluation [Kissner-Song'05, Dachman-Soled-Malkin-Raykova-Yung'11, ...]
- RSA [De Cristofaro-Tsudik'10, Atenies-De Cristofaro-Tsudik'10, ...]
- Bloom Filters [Dong-Chen-Wen'13, Rindal-Rosulek'17, ...]
- Garbled Circuits [Huang-Evans-Katz'12, Pinkas-Schneider-Segev-Zohner'15, Pinkas-Schneider-Weinert-Wieder'18, Pinkas-Schneider-Tkachenko-Yanai'19, ...]
- Homomorphic Encryption [Chen-Laine-Rindal'17, Chen-Huang-Laine-Rindal'18, Cong-Moreno-da Gama-Dai-Iliashenko-Laine-Rosenberg'21, ...]
- Oblivious Transfer [Pinkas-Schneider-Zohner'14, Kolesnikov-Kumaresan-Rosulek-Trieu'16, Pinkas-Rosulek-Trieu-Yanai'19, Chase-M'20, Pinkas-Rosulek-Trieu-Yanai'20, ...]
- Vector Oblivious Linear Evaluation (VOLE) [Rindal-Schoppmann'21, Garimella-Pinkas-Rosulek-Trieu-Yanai'21, Raghuraman-Rindal'22, Bienstock-Patel-Seo-Yeo'23, ...]

# Outline

- Private Set Intersection (PSI)
- Updatable PSI
  - Motivation and Our Results
  - Construction from Oblivious Data Structure
- Fuzzy-Matching PSI
  - Motivation and Our Results
  - Construction from Function Secret Sharing
- PSI Over Committed Inputs
  - Motivation and Our Results

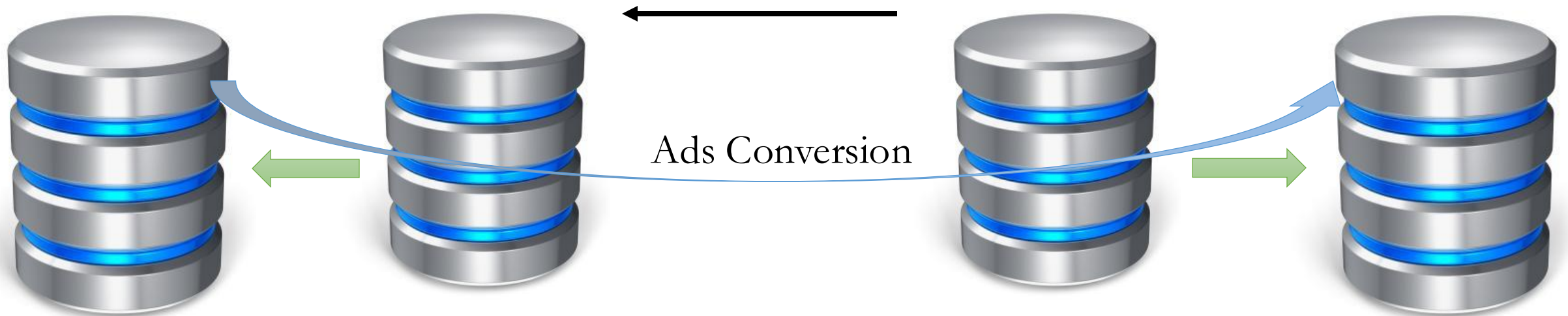
# Application: Ads Conversion Analysis



# Application: Ads Conversion Analysis

**Updatable** PSI where parties regularly run PSI on updated sets?

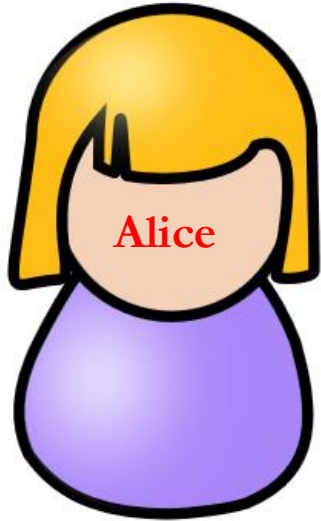
**Goal:** Computation & communication costs of every PSI only grow with their **updates** instead of the **entire** sets.



# Updatable PSI

- Prior Work
  - Computation & communication grow with **entire** sets
- Our Work [Badrinarayanan-M-Xie'22, Badrinarayanan-M-Shi-Tromanhauser-Zeng'24]
  - Formalizing UPSI
  - New protocols for UPSI
- Ongoing Work [Liu-M-Rosulek-Shi-Wang]
  - Minimizing public-key operations (VOLE-based)

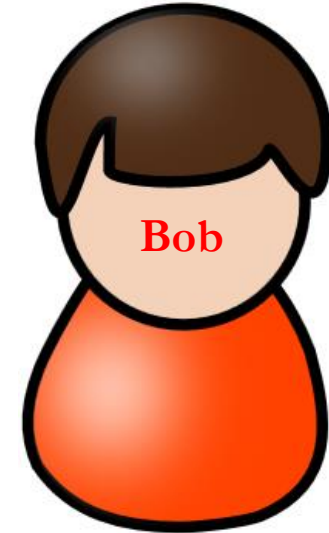
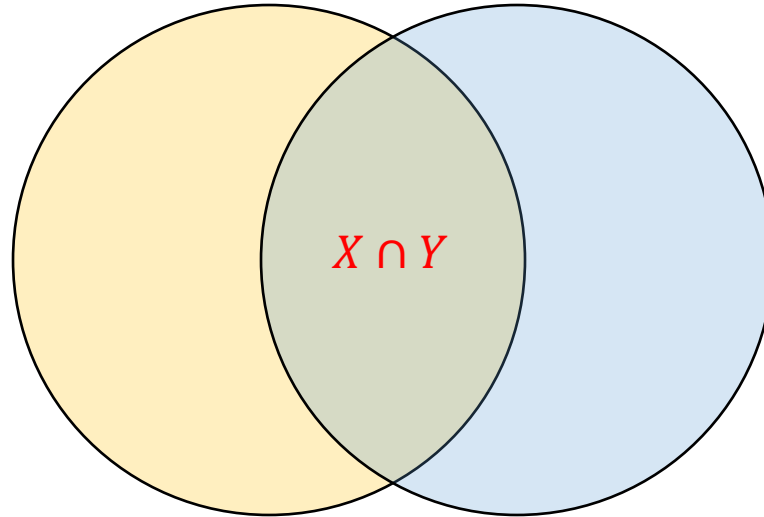
# Updatable PSI (UPSI)



$$X = \{x_1, x_2, \dots, x_n\}$$

Input:  $X^-, X^+$ , where  $|X^-|, |X^+| \ll |X|$

New set:  $X' := (X \setminus X^-) \cup X^+$

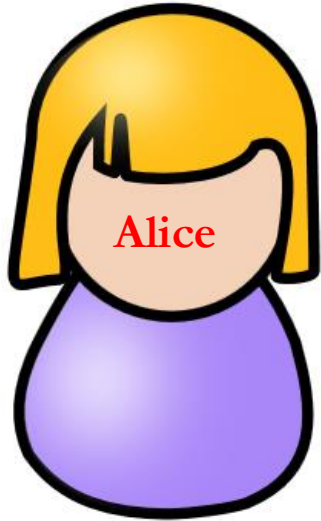


$$Y = \{y_1, y_2, \dots, y_m\}$$

Input:  $Y^-, Y^+$ , where  $|Y^-|, |Y^+| \ll |Y|$

New set:  $Y' := (Y \setminus Y^-) \cup Y^+$

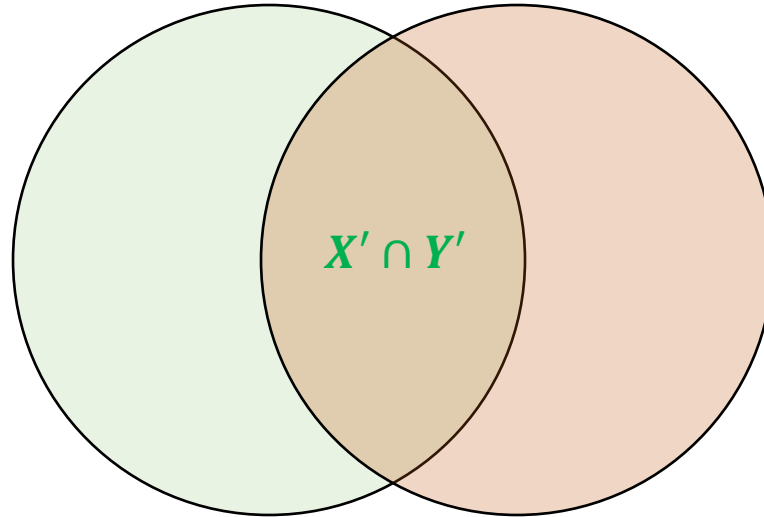
# Updatable PSI (UPSI)



$$X = \{x_1, x_2, \dots, x_n\}$$

**Input:**  $X^-, X^+$ , where  $|X^-|, |X^+| \ll |X|$

**New set:**  $X' := (X \setminus X^-) \cup X^+$



$$Y = \{y_1, y_2, \dots, y_m\}$$

**Input:**  $Y^-, Y^+$ , where  $|Y^-|, |Y^+| \ll |Y|$

**New set:**  $Y' := (Y \setminus Y^-) \cup Y^+$

**Goal:** Computation & communication costs of every UPSI only grow with their **updates** rather than the **entire** sets.

# Our Results [Badrinarayanan-M-Xie'22]

Functionality	Output	Security	Computation	Communication
Addition-Only UPSI	Two-Sided	Semi-Honest	$O(n)$	$O(n)$
Addition-Only UPSI	One-Sided	Semi-Honest	$O^*(n \cdot \log N)$	$O^*(n \cdot \log N)$
UPSI with Weak Deletion	Two-Sided	Semi-Honest	$O(n \cdot t)$	$O(n \cdot t)$

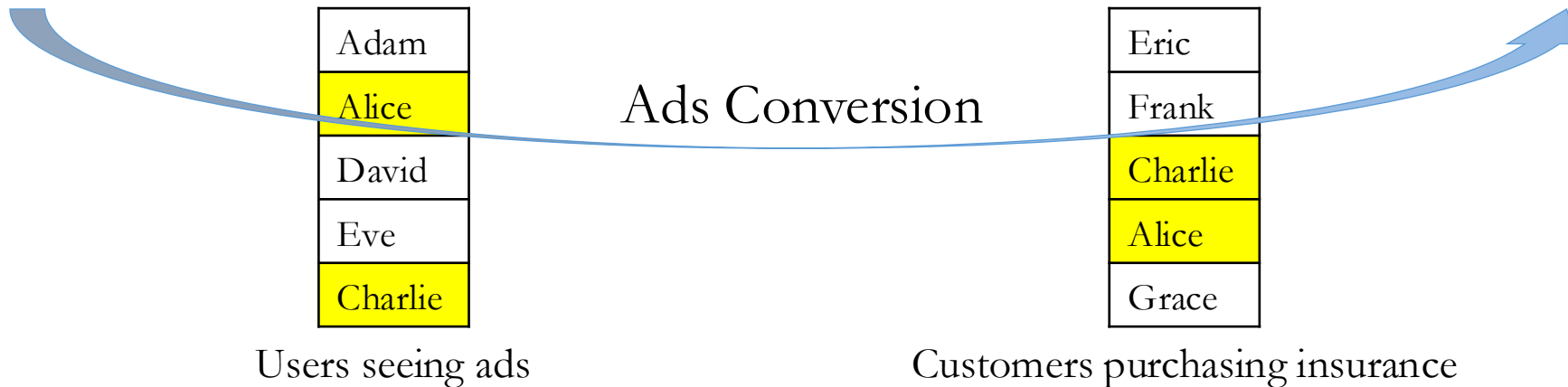
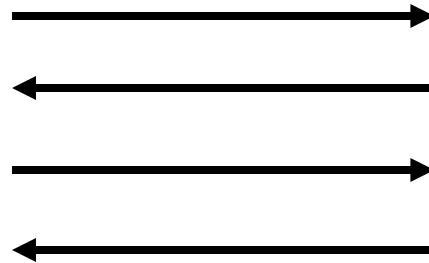
- Parameters
  - $N$ : size of entire sets
  - $n$ : size of updates
  - $O^*(\cdot)$ : amortized complexity
  - $t$ : number of days when parties refresh their sets

# Limitations

- Functionality
  - [BMX'22]: Plain PSI
  - In reality: PSI-Cardinality/Sum/Secret-Share

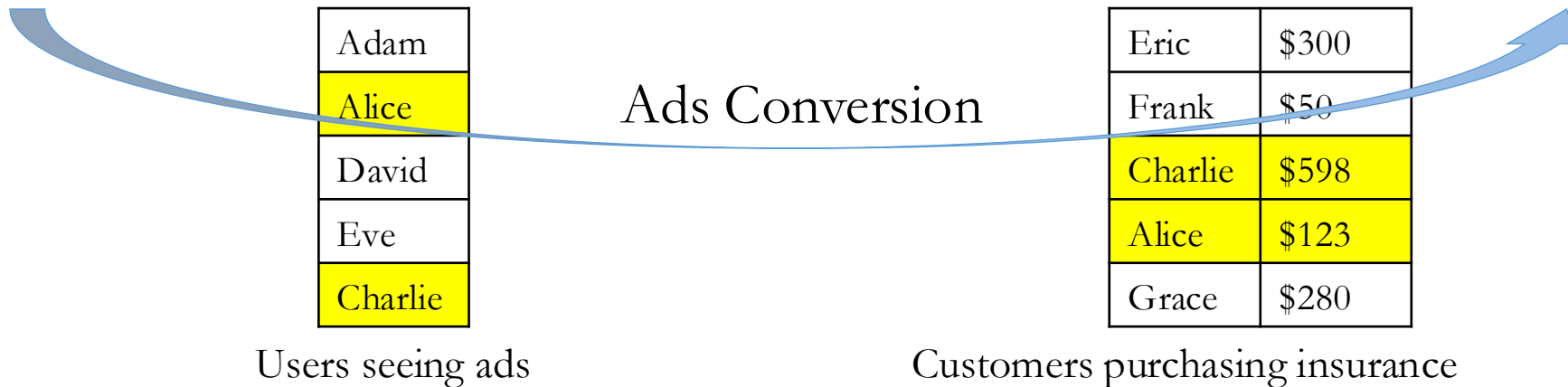
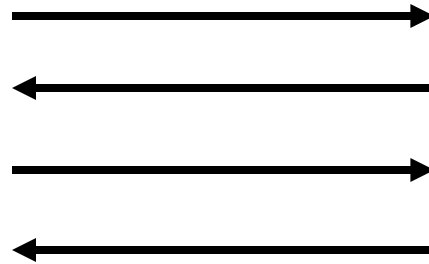
# Application: Ads Conversion Analysis

PSI-Cardinality:  $|X \cap Y|$



# Application: Ads Conversion Analysis

$$\text{PSI-Sum: } \sum_{y_i \in X} v_i$$



# Limitations

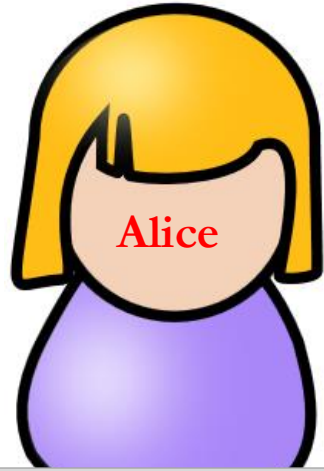
- Functionality
  - [BMX'22]: Plain PSI
  - In reality: PSI-Cardinality/Sum/Secret-Share
- Addition-Only
  - [BMX'22]: Parties can only add new elements
  - In reality: Parties may add or delete elements
- Amortized Complexity
  - [BMX'22]: Worst-case complexity of  $O(N)$

# Our Results [Badrinarayanan-M-Shi-Tromanhauser-Zeng'24]

Functionality	Output	Security	Computation	Communication
Addition-Only UPSI	Two-Sided	Semi-Honest	$O(n)$	$O(n)$
Addition-Only UPSI	One-Sided	Semi-Honest	$O^*(n \cdot \log N)$	$O^*(n \cdot \log N)$
UPSI with Weak Deletion	Two-Sided	Semi-Honest	$O(n \cdot t)$	$O(n \cdot t)$
Addition-Only UPSI/ Cardinality/Sum/Secret-Share	One-Sided	Semi-Honest	$O(n \cdot \log N)$	$O(n \cdot \log N)$
Addition-and-Deletion UPSI/ Cardinality/Sum	One-Sided	Semi-Honest	$O(n \cdot \log^2 N)$	$O(n \cdot \log^2 N)$

- Parameters
  - $N$ : size of entire sets
  - $n$ : size of updates
  - $O^*(\cdot)$ : amortized complexity
  - $t$ : number of days when parties refresh their sets

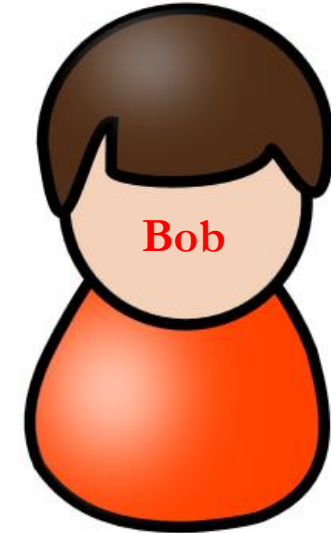
# UPSI from Oblivious Data Structure



$X^+ \cap (Y \cup Y^+) ?$

$x \in ?$

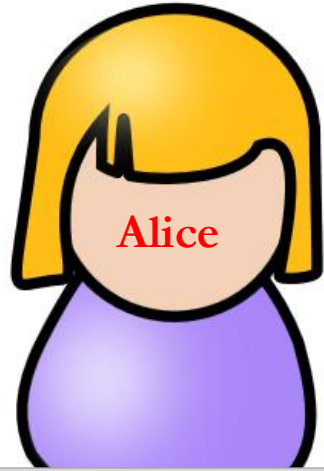
Enc( $y_{26}$ )
Enc( $y_2$ )
Enc( $y_{12}$ )
Enc( $y_{31}$ )
...
Enc( $y_5$ )



$Y = \{y_1, y_2, \dots, y_N\}$



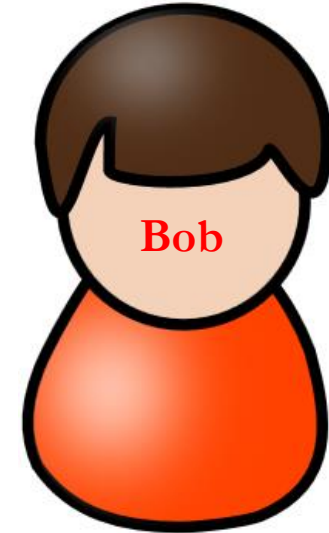
# UPSI from Oblivious Data Structure



$X^+ \cap (Y \cup Y^+) ?$

$x \in ?$

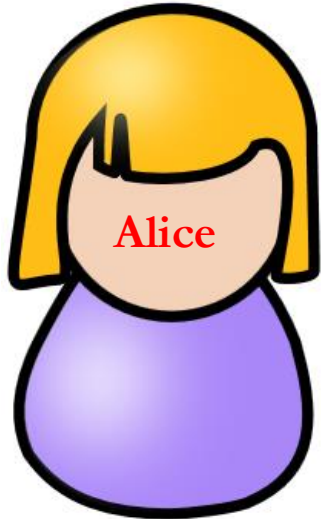
Enc( $y_{26}$ )
Enc( $y_2$ )
Enc( $y_{12}$ )
Enc( $y_{31}$ )
...
Enc( $y_5$ )



$Y = \{y_1, y_2, \dots, y_N\}$

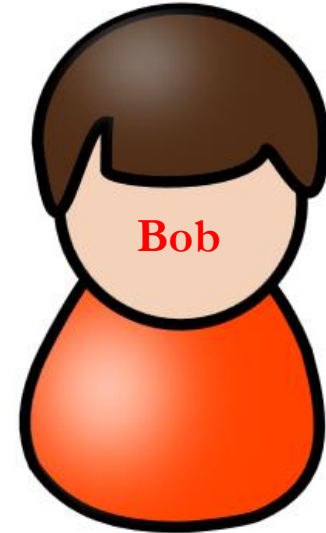


# Comparison of Encrypted Values



$(pk, sk_A)$

2-out-of-2 threshold additively homomorphic encryption



$(pk, sk_B)$

$x \stackrel{?}{\in} \{Enc_{pk}(y_1), Enc_{pk}(y_2), \dots, Enc_{pk}(y_k)\}$

$\{Enc_{pk}(x - y_j)\}_{j \in [k]}$



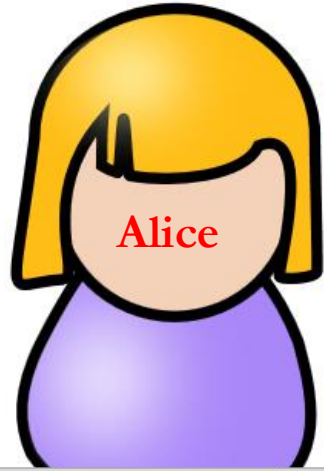
$\alpha_j \leftarrow \mathbb{F} \quad \forall j \in [k]$

$\{Dec_{sk_B}(Enc_{pk}((x - y_j) \cdot \alpha_j))\}_{shuffled}$



$(x - y_j) \cdot \alpha_j \stackrel{?}{=} 0$

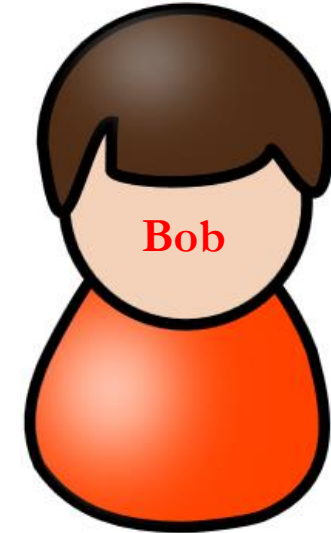
# Oblivious Data Structure?



$X^+ \cap (Y \cup Y^+) ?$

$x \in ?$

Enc( $y_{26}$ )
Enc( $y_2$ )
Enc( $y_{12}$ )
Enc( $y_{31}$ )
...
Enc( $y_5$ )



$Y = \{y_1, y_2, \dots, y_N\}$

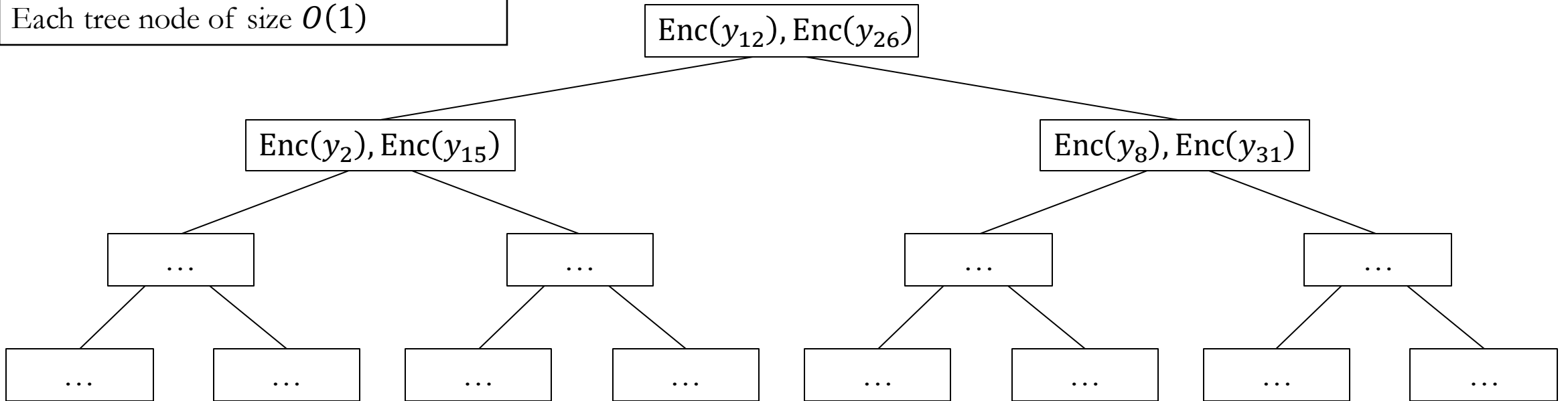


How to allow Bob to **update** the encrypted dataset in an **oblivious** way while allowing Alice to **query** it **obliviously**?

# Oblivious Data Structure

(Inspired by Path ORAM [Stefanov-van Dijk-Shi-Chan-Fletcher-Ren-Yu-Devadas'13])

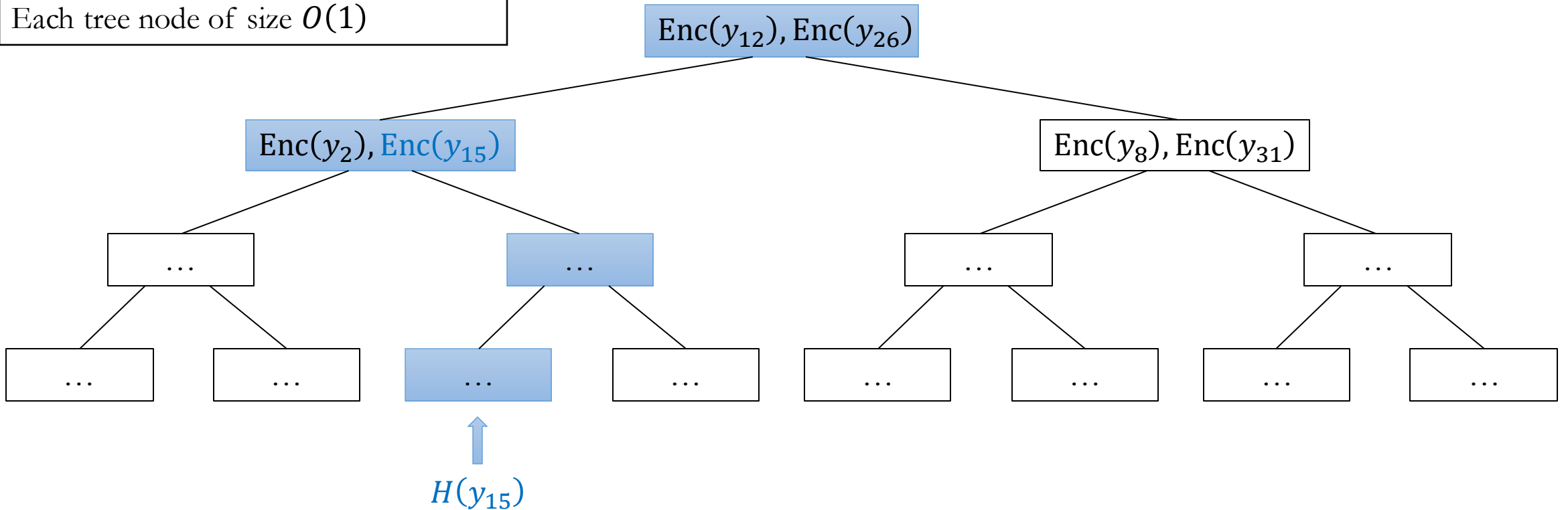
Full binary tree with depth  $O(\log N)$   
Each tree node of size  $O(1)$



# Oblivious Data Structure

(Inspired by Path ORAM [Stefanov-van Dijk-Shi-Chan-Fletcher-Ren-Yu-Devadas'13])

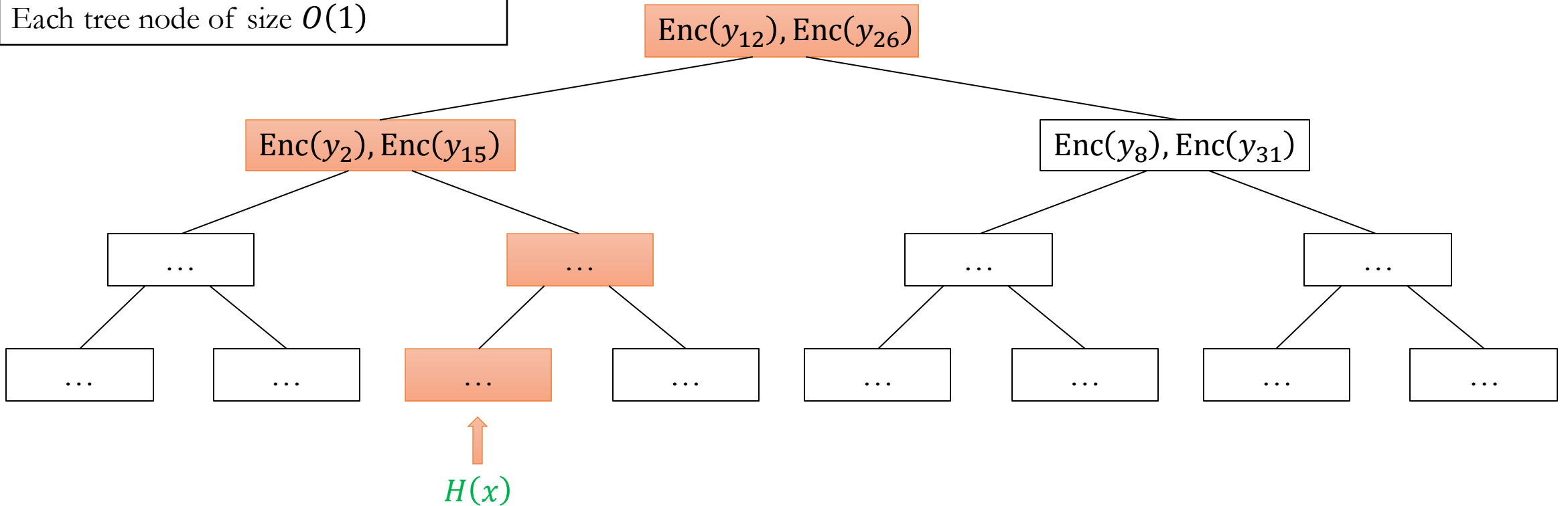
Full binary tree with depth  $O(\log N)$   
Each tree node of size  $O(1)$



**Invariant:** Each element  $y_i$  is always stored in the root-to-leaf path to  $H(y_i)$  for a public hash function  $H$  (or in the stash).

# Oblivious Data Structure: Alice to Query( $x$ )

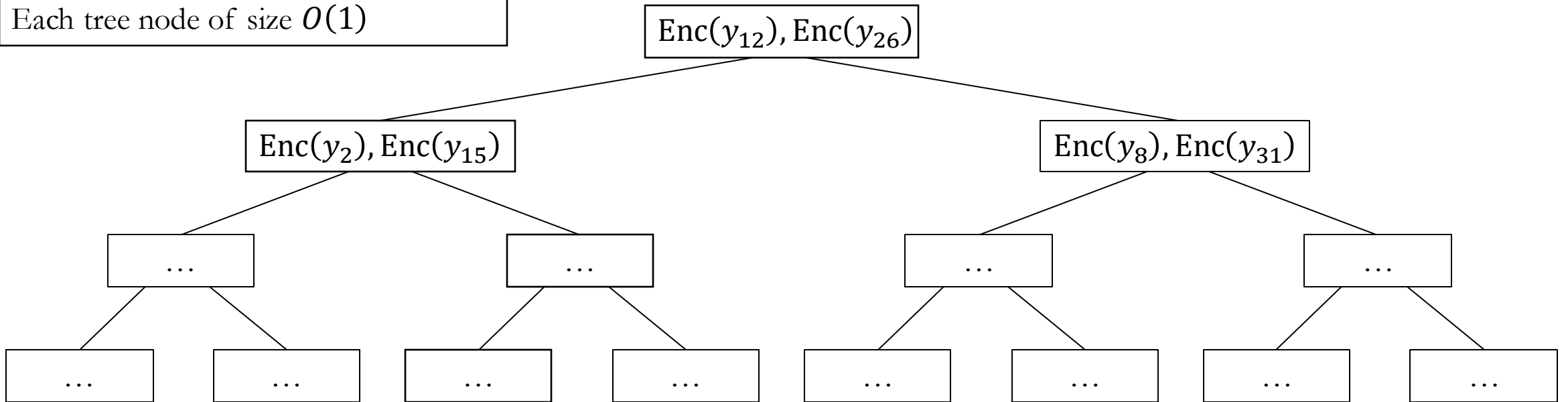
Full binary tree with depth  $O(\log N)$   
Each tree node of size  $O(1)$



**Invariant:** Each element  $x$  is always stored in the root-to-leaf path to  $H(x)$  for a public hash function  $H$  (or in the stash).

# Oblivious Data Structure: Bob to Add( $y_{36}$ )

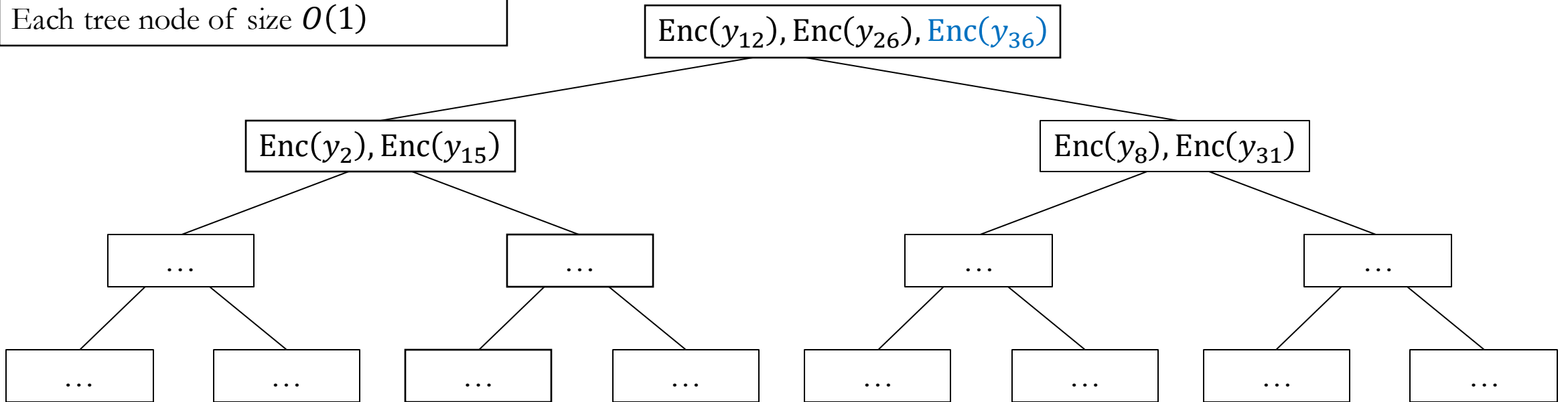
Full binary tree with depth  $O(\log N)$   
Each tree node of size  $O(1)$



**Invariant:** Each element  $y_i$  is always stored in the root-to-leaf path to  $H(y_i)$  for a public hash function  $H$  (or in the stash).

# Oblivious Data Structure: Bob to Add( $y_{36}$ )

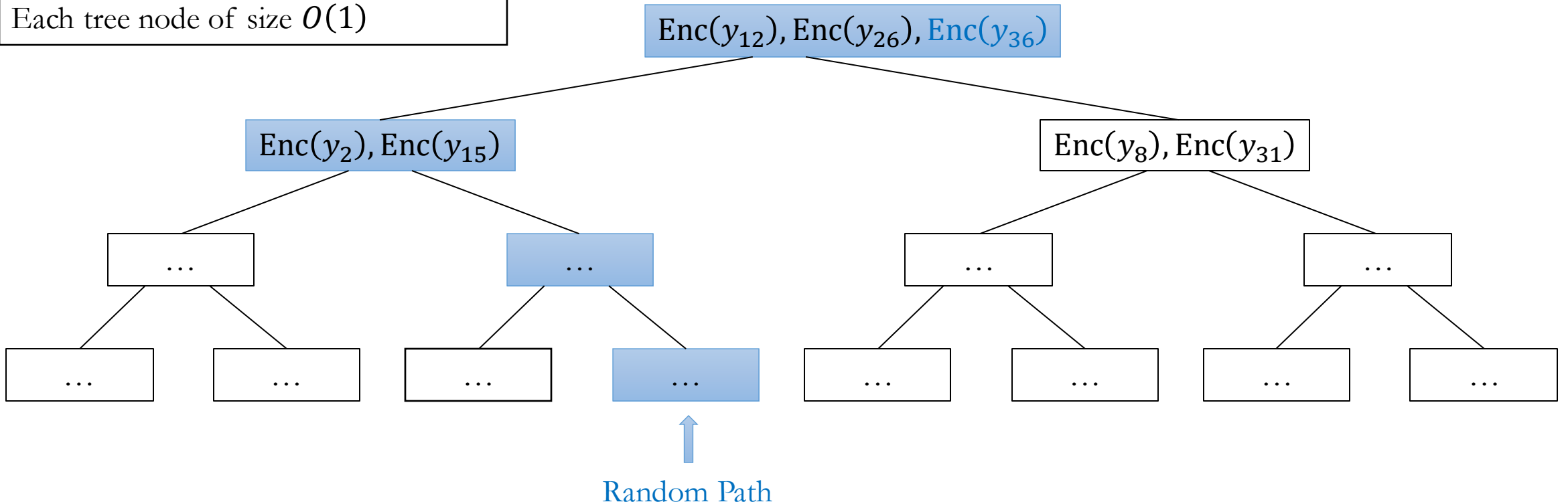
Full binary tree with depth  $O(\log N)$   
Each tree node of size  $O(1)$



**Invariant:** Each element  $y_i$  is always stored in the root-to-leaf path to  $H(y_i)$  for a public hash function  $H$  (or in the stash).

# Oblivious Data Structure: Bob to Add( $y_{36}$ )

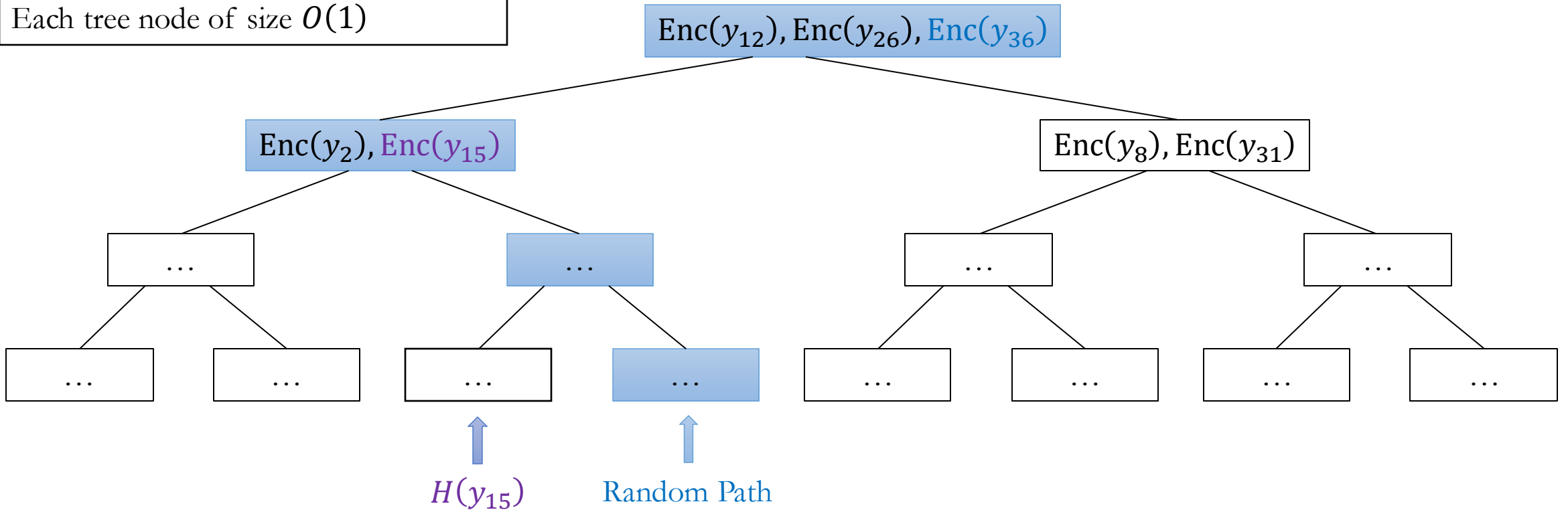
Full binary tree with depth  $O(\log N)$   
Each tree node of size  $O(1)$



**Invariant:** Each element  $y_i$  is always stored in the root-to-leaf path to  $H(y_i)$  for a public hash function  $H$  (or in the stash).

# Oblivious Data Structure: Bob to Add( $y_{36}$ )

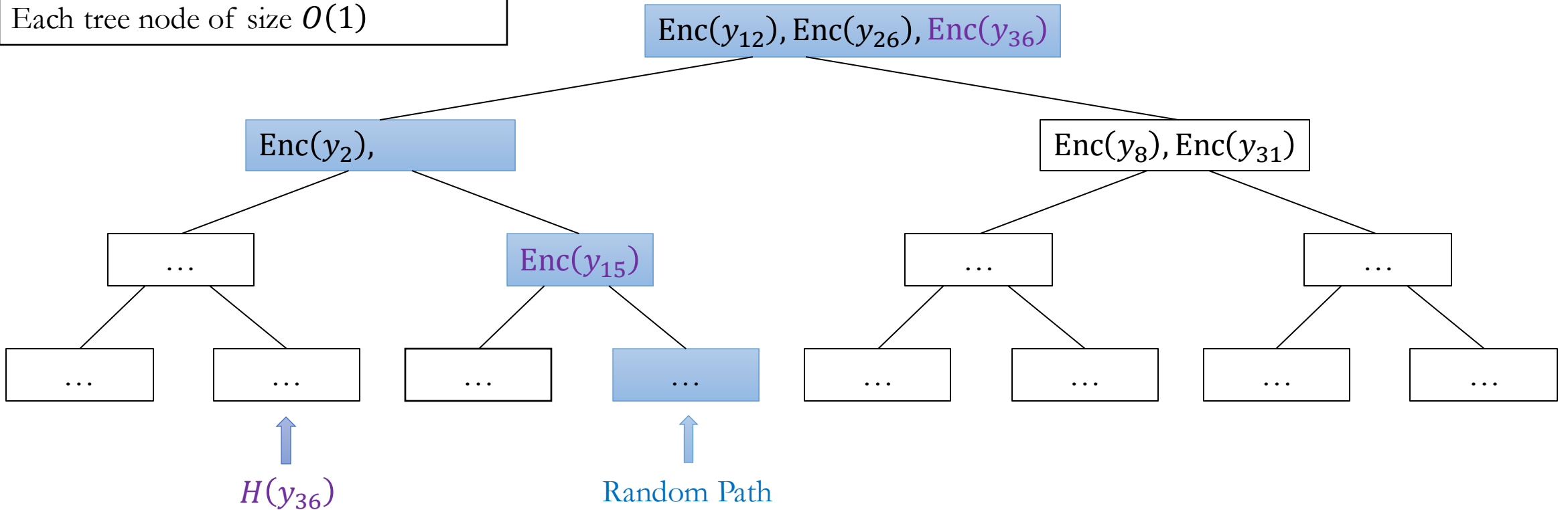
Full binary tree with depth  $O(\log N)$   
Each tree node of size  $O(1)$



**Invariant:** Each element  $y_i$  is always stored in the root-to-leaf path to  $H(y_i)$  for a public hash function  $H$  (or in the stash).

# Oblivious Data Structure: Bob to Add( $y_{36}$ )

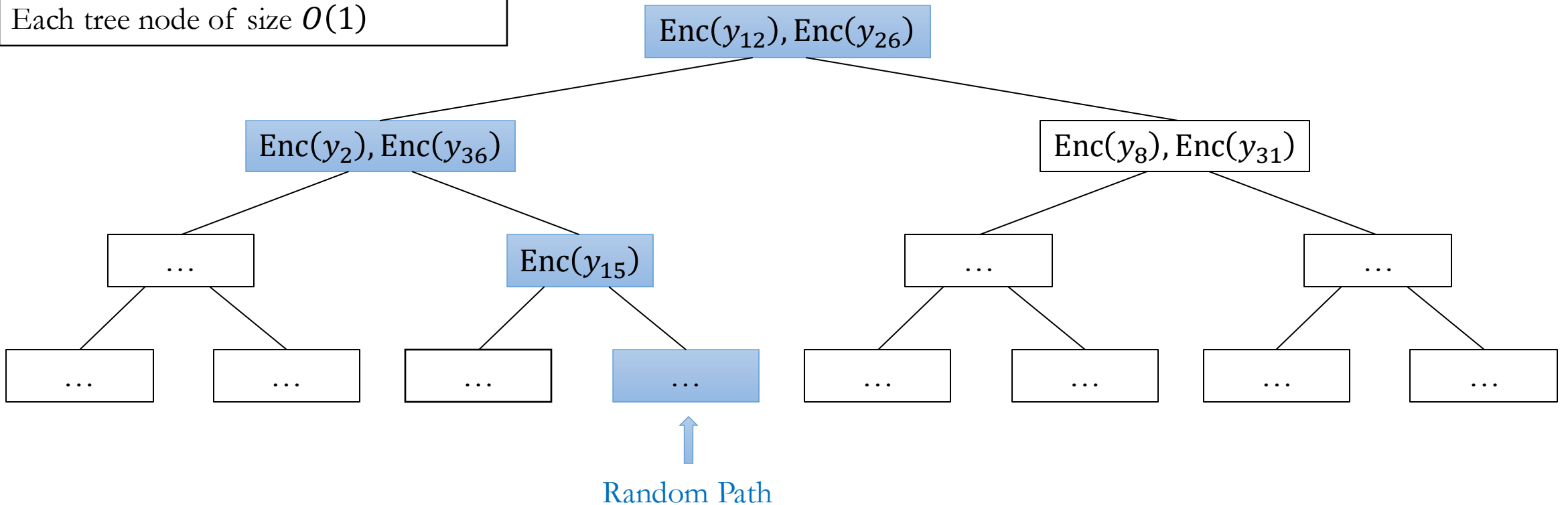
Full binary tree with depth  $O(\log N)$   
Each tree node of size  $O(1)$



**Invariant:** Each element  $y_i$  is always stored in the root-to-leaf path to  $H(y_i)$  for a public hash function  $H$  (or in the stash).

# Oblivious Data Structure: Bob to Add( $y_{36}$ )

Full binary tree with depth  $O(\log N)$   
Each tree node of size  $O(1)$



**Invariant:** Each element  $y_i$  is always stored in the root-to-leaf path to  $H(y_i)$  for a public hash function  $H$  (or in the stash).

# Performance

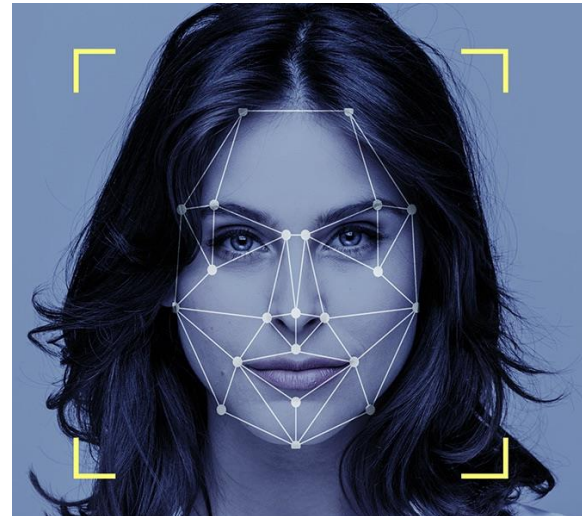
$N$	$N_d$	Protocol	Comm. (MB)	Total Running Time (s)			
				LAN	200Mbps	50Mbps	5Mbps
$2^{20}$	—	RR22	149	31.1	38.4	51.9	258
		CGS22 (C-PSI <sub>1</sub> )	2190	31.0	135	414	3771
		CGS22 (C-PSI <sub>2</sub> )	1408	24.3	92.8	268	3872
	$2^6$	$\Pi_{\text{UPSI-Add}_{ca}}$	3.03	7.59	8.14	8.46	12.6
	$2^8$		11.8	29.6	30.6	32.0	48.7
	$2^{10}$		45.7	116	121	127	194
	$2^6$	$\Pi_{\text{UPSI-Add}_{sum}}$	5.70	11.8	12.5	13.1	21.5
	$2^8$		22.3	45.9	47.2	49.3	82.0
	$2^{10}$		87.1	178	184	195	321
	$2^6$	$\Pi_{\text{UPSI-Add}_{circuit}}$	17.1	81.7	83.1	85.3	110
	$2^8$		67.0	318	327	330	427
	$2^{10}$		264	1251	1263	1295	1674

# Outline

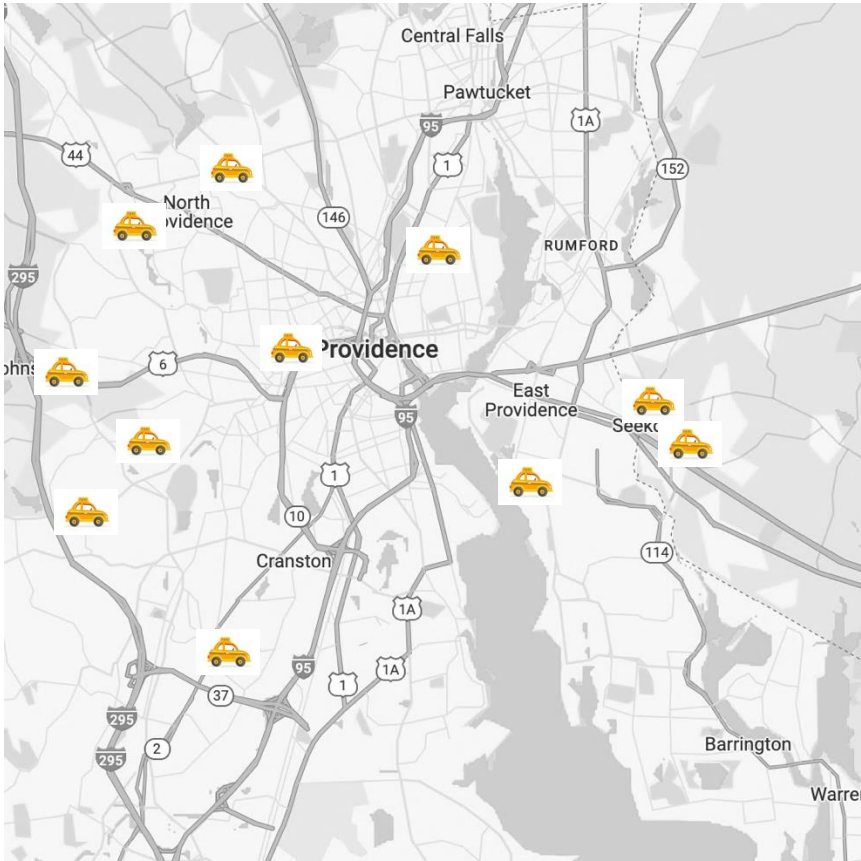
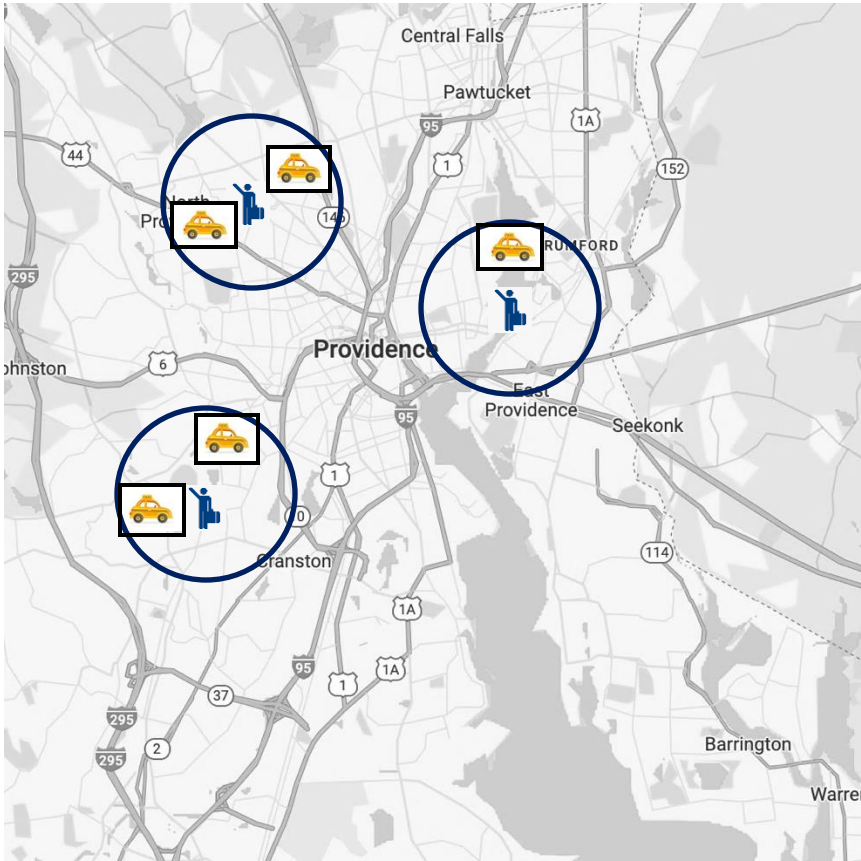
- Private Set Intersection (PSI)
- Updatable PSI
  - Motivation and Our Results
  - Construction from Oblivious Data Structure
- Fuzzy-Matching PSI
  - Motivation and Our Results
  - Construction from Function Secret Sharing
- PSI Over Committed Inputs
  - Motivation and Our Results

# Fuzzy-Matching PSI

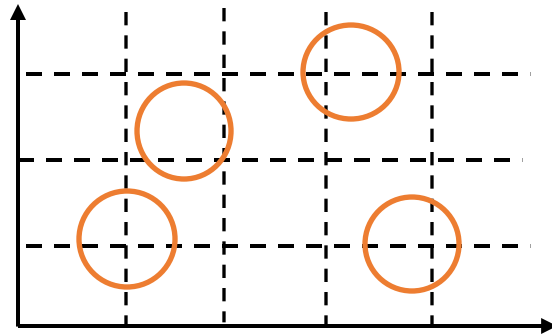
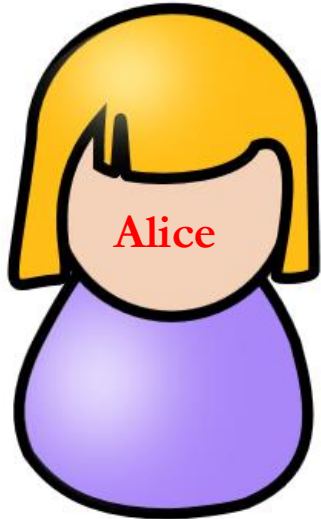
- Biometric Matching: fingerprints, facial recognition, ...
- Password Breach Detection: typos
- Ads Conversion Analysis: fuzzy PII (physical addresses, ...)



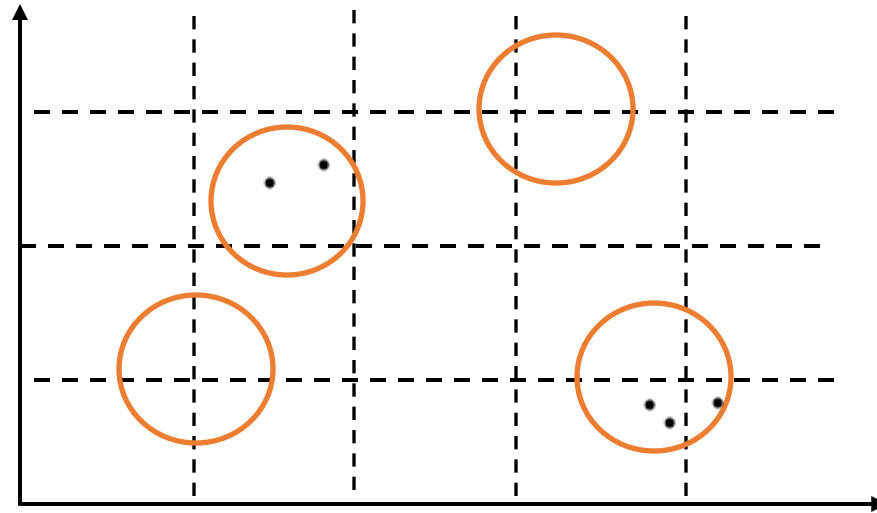
# Privacy-Preserving Ride Sharing



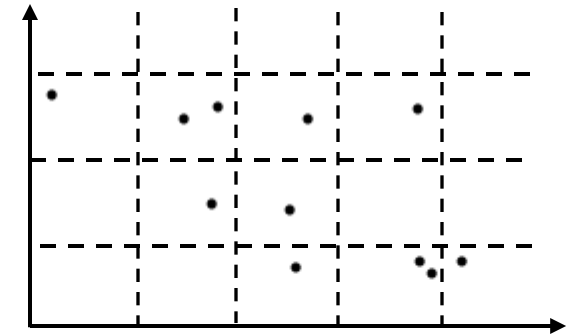
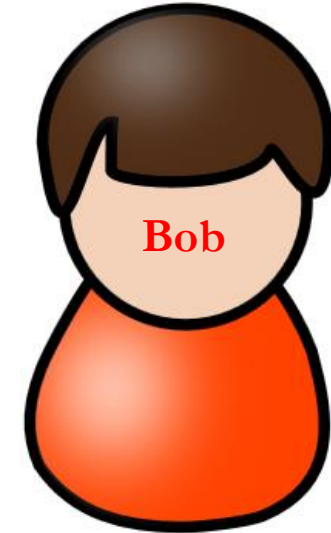
# Naïve Approach: Reduce to Plain PSI



$X = \{\text{all nearby points}\}$

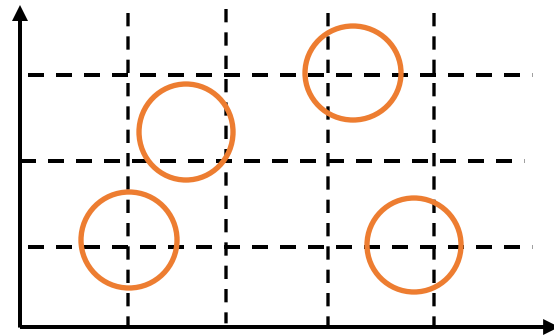
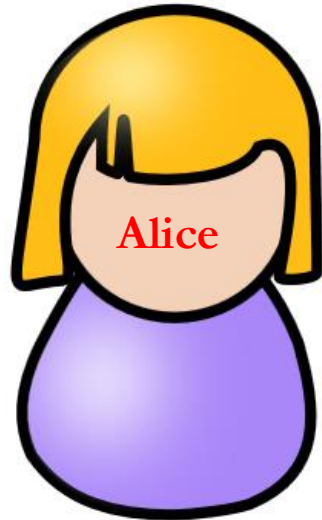


$X \cap Y$

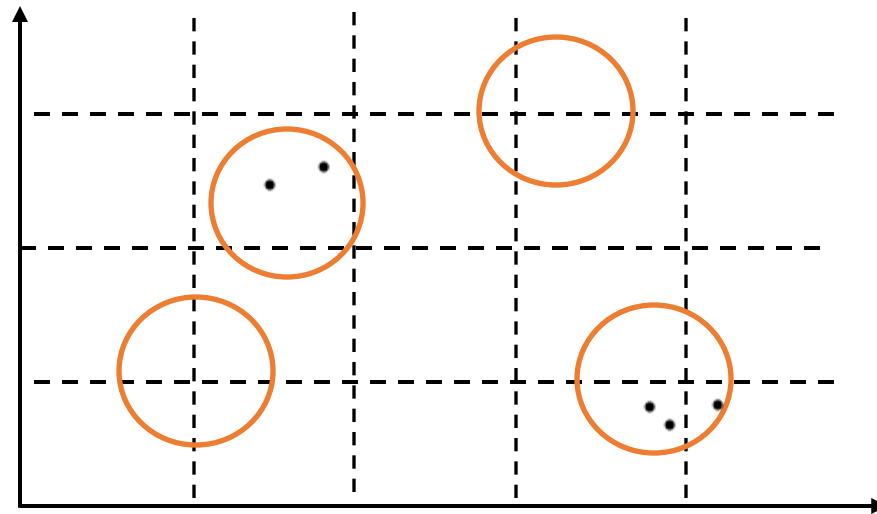


$Y = \{\text{unstructured points}\}$

# Structure-Aware PSI [Garimella-Rosulek-Singh'22]

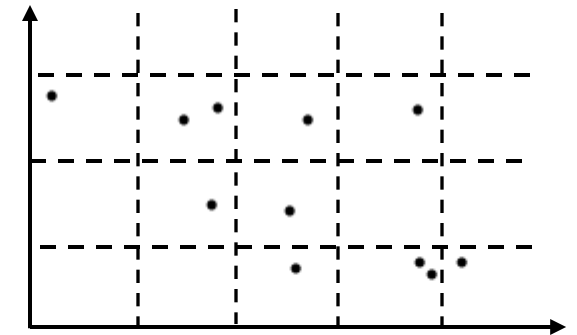
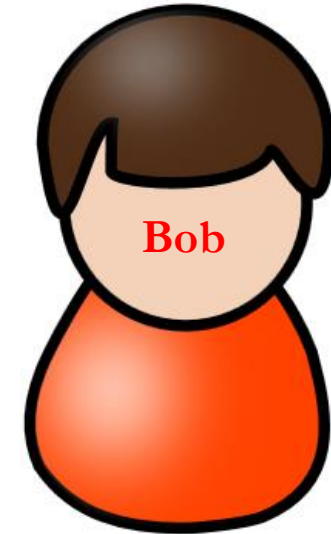


$X = \text{structured set}$



$X \cap Y = \{\text{all points inside structured set}\}$

**Goal:** Computation & communication costs only grow with the **description size** of  $X$  rather than  $|X|$ .

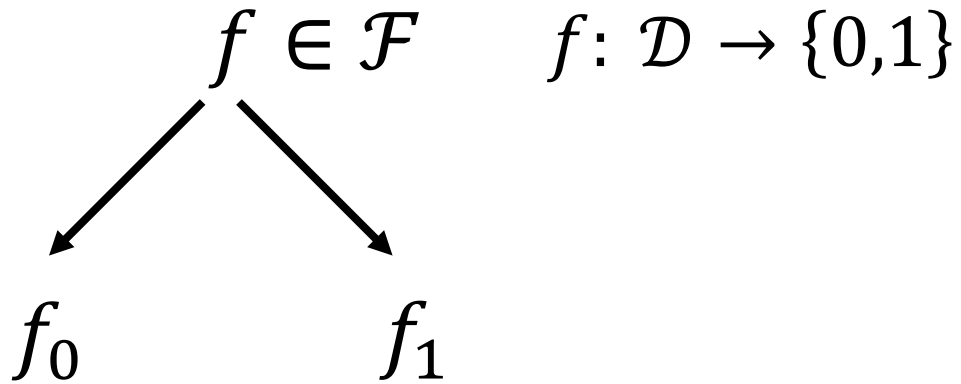


$Y = \{\text{unstructured points}\}$

# Structure-Aware PSI

- Prior Work [Garimella-Rosulek-Singh'22, Garimella-Rosulek-Singh'23] ← FSS
  - **Complexity:** **Communication** only grows with **description size** of  $X$
  - **Functionality:** **Plain** PSI
  - **Structure:** Union of **disjoint** balls
  - **Output:** Only **Alice** learns output
- Our Results [Garimella-Goff-M'24] ← Incremental FSS
  - **Complexity:** **Communication & Computation** only grow with **description size** of  $X$
  - **Functionality:** **PSI/PSI-Cardinality/PSI-Sum**
  - **Structure:** Union of **overlapping** balls
  - **Output:** **Alice or Bob** learns output
- New Framework [Bui-Garimella-M-Pham'25] ← Distributed (Incremental) FSS
  - **Efficiency:** Eliminate a hidden **multiplicative overhead of security parameter (128x)**

# Function Secret Sharing (FSS) [Boyle-Gilboa-Ishai'15]

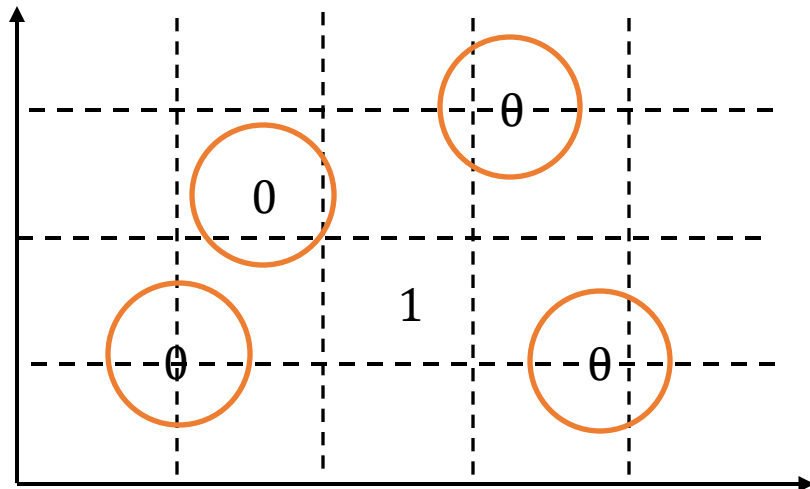


- **Correctness:**  $f_0 \oplus f_1 = f$ 
  - $\forall x \in \mathcal{D}, f_0(x) \oplus f_1(x) = f(x)$
- **Privacy:**  $f_0, f_1$  look random
  - $f_b \approx \text{Sim}(1^\lambda, b)$
- **Succinctness:**  $|f_0|, |f_1| \ll |\mathcal{D}|$

# FSS for Structured Sets

$$f_X(x) = \begin{cases} 0 & \text{if } x \in X \\ 1 & \text{otherwise} \end{cases}$$

$f_0$        $f_1$



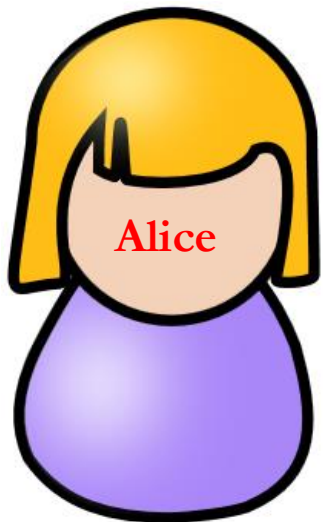
- **Correctness:**  $f_0 \oplus f_1 = f$ 
  - $\forall x \in \mathcal{D}, f_0(x) \oplus f_1(x) = f(x)$
- **Privacy:**  $f_0, f_1$  look random
  - $f_b \approx \text{Sim}(1^\lambda, b)$
- **Succinctness:**  $|f_0|, |f_1| \ll |\mathcal{D}|$

- $x \in X: f_0(x) = f_1(x)$
- $x \notin X: f_0(x) \neq f_1(x)$

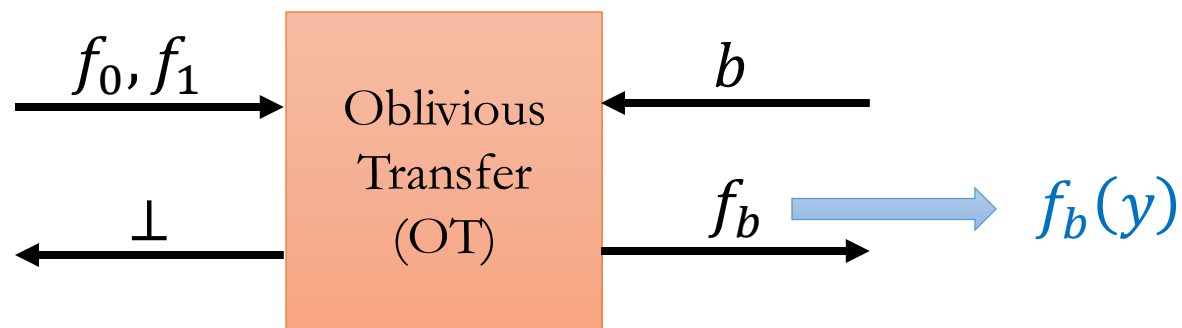
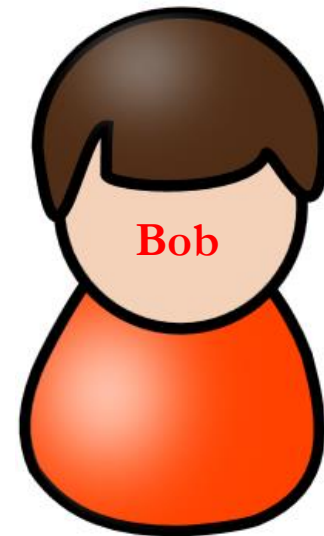
# Structure-Aware PSI from FSS

$$f_X(x) = \begin{cases} 0 & \text{if } x \in X \\ 1 & \text{otherwise} \end{cases}$$

$f_0$        $f_1$

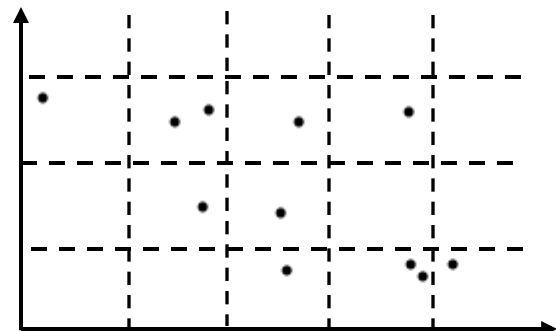


$$b \leftarrow \{0,1\}$$



- $y \in X$ :  $f_0(y) = f_1(y) \rightarrow f_b(y)$  computable by Alice
- $y \notin X$ :  $f_0(y) \neq f_1(y) \rightarrow f_b(y)$  unpredictable by Alice

$X =$  structured set



$Y =$  {unstructured points}

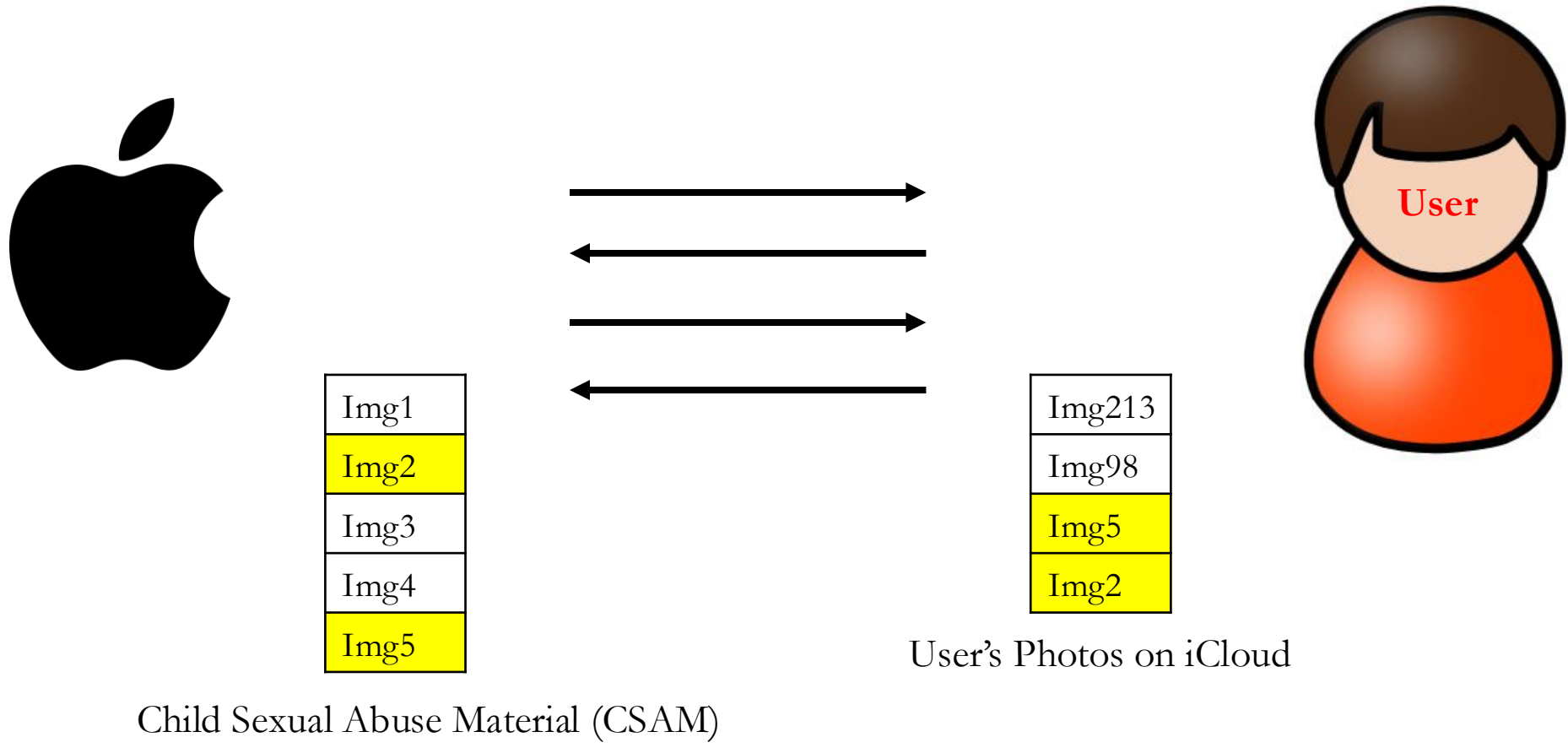
# Structure-Aware PSI

- Prior Work [Garimella-Rosulek-Singh'22, Garimella-Rosulek-Singh'23] ← FSS
  - **Complexity:** **Communication** only grows with **description size** of  $X$
  - **Functionality:** **Plain** PSI
  - **Structure:** Union of **disjoint** balls
  - **Output:** Only **Alice** learns output
- Our Results [Garimella-Goff-M'24] ← Incremental FSS
  - **Complexity:** **Communication & Computation** only grow with **description size** of  $X$
  - **Functionality:** **PSI/PSI-Cardinality/PSI-Sum**
  - **Structure:** Union of **overlapping** balls
  - **Output:** **Alice or Bob** learns output
- New Framework [Bui-Garimella-M-Pham'25] ← Distributed (Incremental) FSS
  - **Efficiency:** Eliminate a hidden **multiplicative overhead of security parameter (128x)**

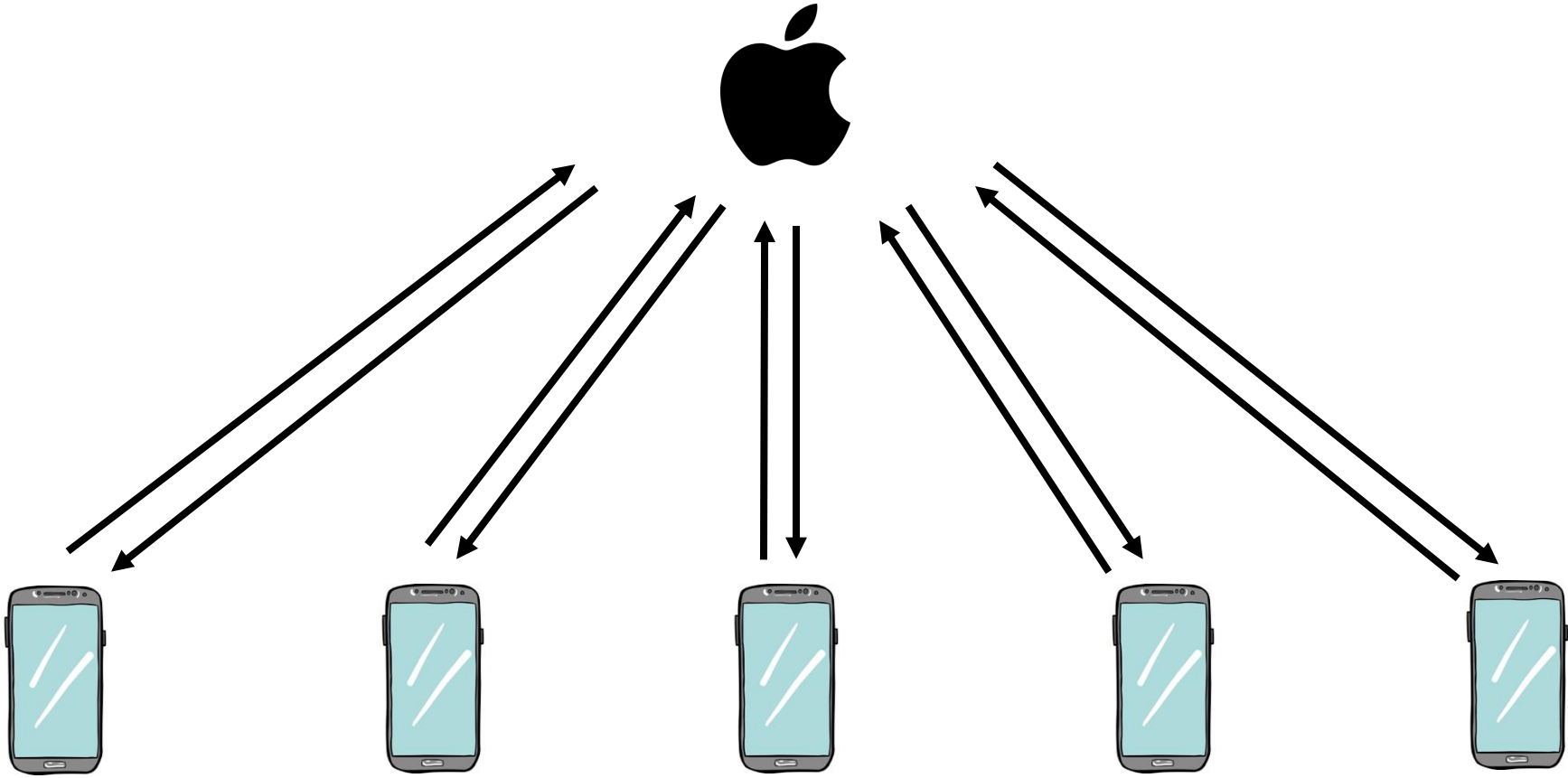
# Outline

- Private Set Intersection (PSI)
- Updatable PSI
  - Motivation and Our Results
  - Construction from Oblivious Data Structure
- Fuzzy-Matching PSI
  - Motivation and Our Results
  - Construction from Function Secret Sharing
- PSI Over Committed Inputs
  - Motivation and Our Results

# Application: CSAM Detection

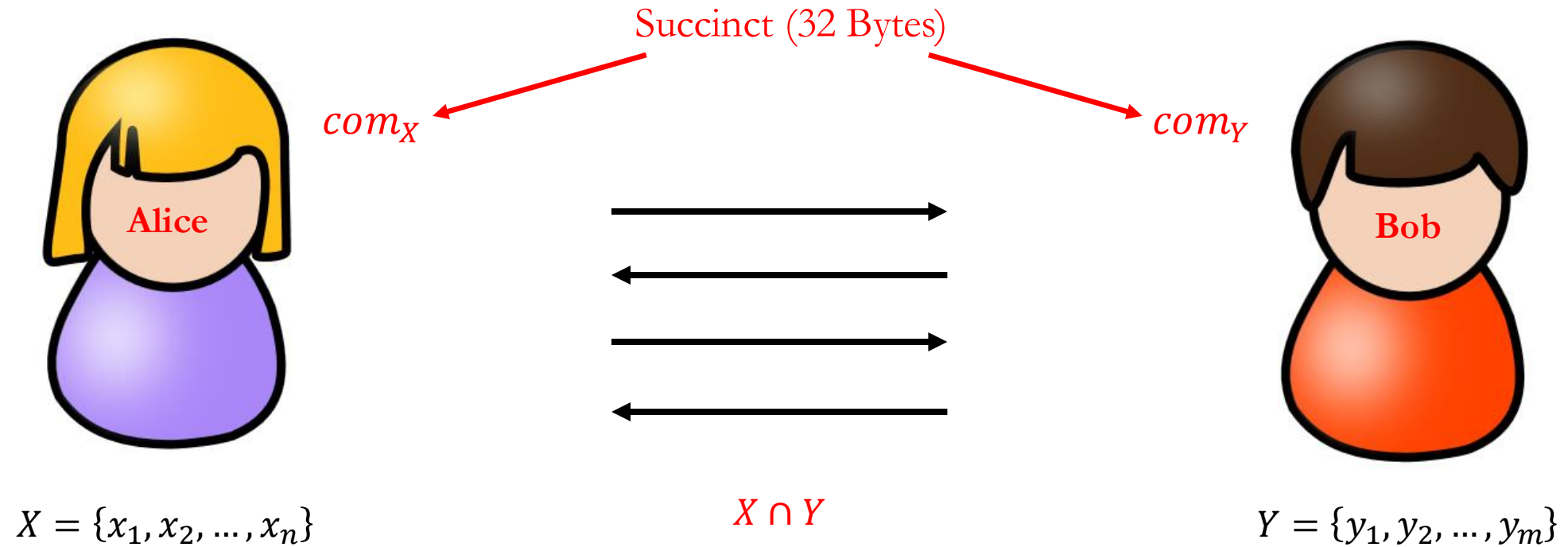


# Application: CSAM Detection

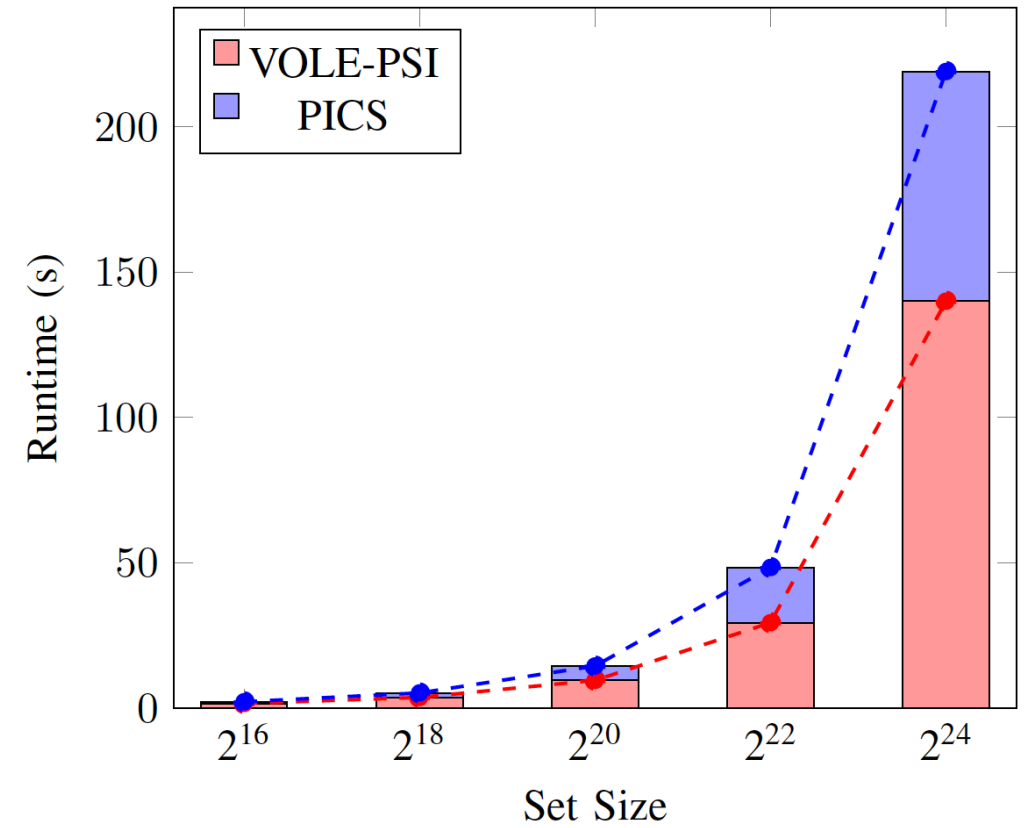
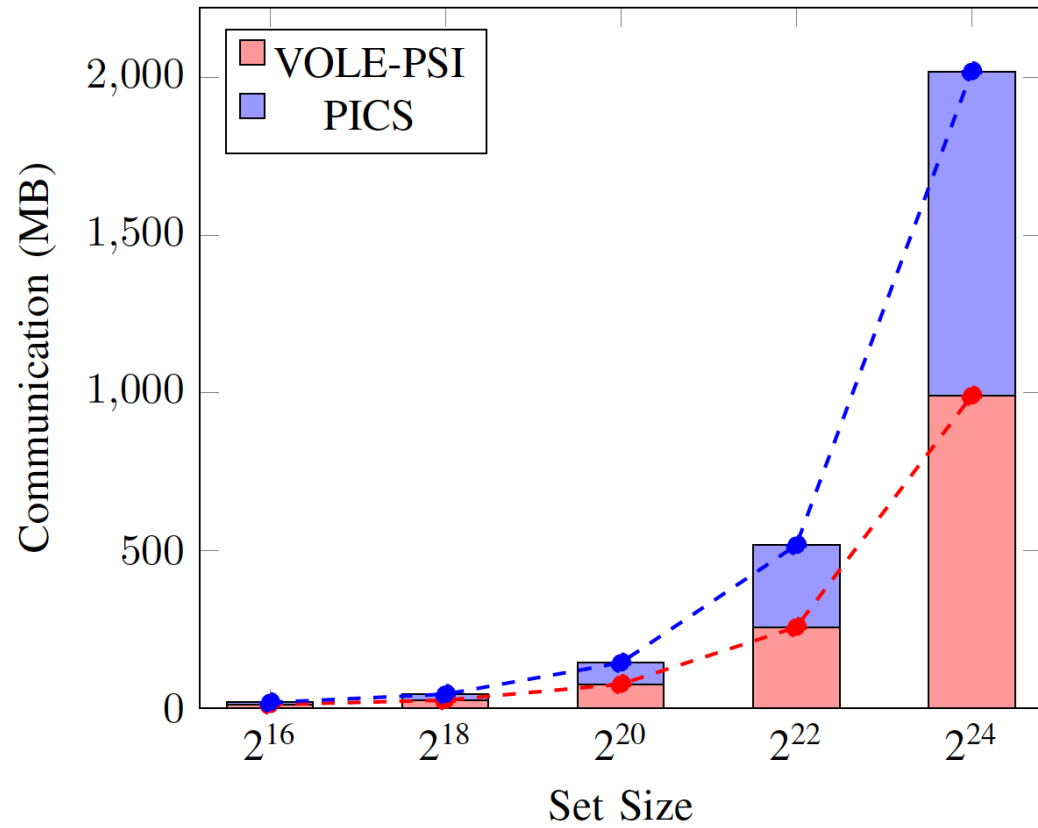


Consistent set across all executions?

# PSI Over Committed Sets [Goel-M-Pham-Sing'25]



# Performance



# Future Directions

- Updatable PSI
  - Malicious security?
  - Updatable oblivious key-value stores (OKVS)?
- Fuzzy-Matching PSI
  - FSS for broader class of structured sets?
  - Support more distance metrics (Hamming, edit,  $\ell_p$ )?
- PSI Over Committed Inputs
  - Updatable?
  - PSI-Cardinality/Sum/Circuit-PSI?
- Broader Applications?



Thank you!