Computation-Efficient Structure-Aware PSI from Incremental Function Secret Sharing

Gayathri Garimella
Benjamin Goff
Peihan Miao

Brown University

*thank you Peihan Miao for the pictures

CRYPTO 2024

Private Set Intersection (PSI)

Alice

input A

special case of two-party secure computation

Bob

input B

{p, r, i, v, a, t, e}

{s, e, c, u, r, i, t, y}

Private Set Intersection (PSI)

Alice

input A

special case of two-party secure computation

Alice learns A ∩ B

Bob

input B

$\{\mathtt{p}, \underline{\mathtt{r}}, \underline{\mathtt{i}}, \mathtt{v}, \mathtt{a}, \underline{\mathtt{t}}, \underline{\mathtt{e}}\}$

$\{?, \mathtt{e}, ?, ?, \mathtt{r}, \mathtt{i}, \mathtt{t}, ?\}$

Bob learns nothing extra

PSI Research

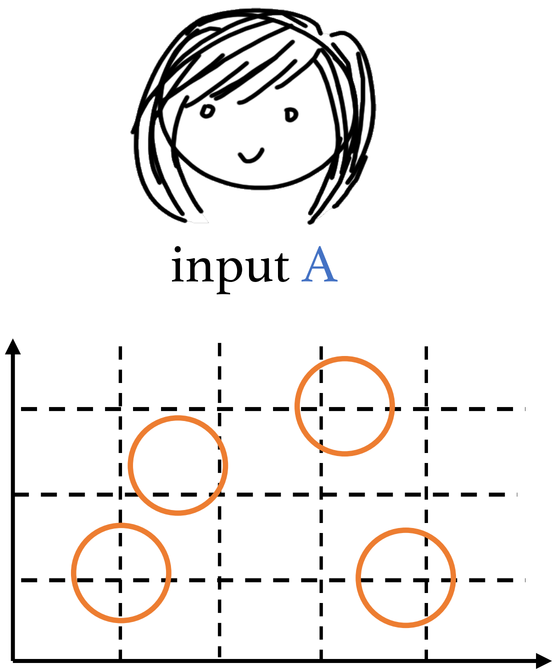
Approaches:
Diffie-Hellman [Mea86, HFH99, JL10, DKT10, IKN+20, RT21…]
Oblivious Polynomial Evaluation [FNP04, KS05, dMRY11…]
RSA [DT10, ADT11]
Bloom Filters [DCW13, RR17a]
FHE [CLR17, CHLR18, CMDG+21]
Circuit-based [HEK12, PSSZ15, PSWW18, PSTY19, GMR+21]
OT [PSZ14, PSSZ15, KKRT16, RR17, PRTY19, CM20, PRTY20, RS21, GPR+21]
Vector OLE [RS21, GPR+21, CRR21, RR22, BPSY23…]

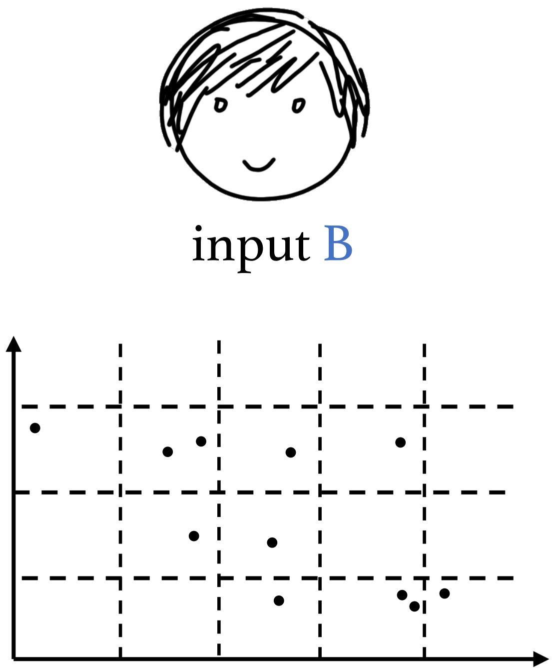
Settings:
Semi-honest/Malicious [RR17, OOS17, CHLR18, PRTY20, RS21, GPR+21, BPSY23…]
Plain/Cardinality/Associated-sum: [PSTY19, KK20, MPR+20, IKN+20, GMR+21, RS21, CGS22...]
PS Union: [DC17, KRTY19, GPR+21, JSZ+22, LG23, BPSY23, GNT24…]
Balanced/Unbalaned/Laconic: [ABD+21, ALOS22, DKL+23, GHMM24..]
Two-party/Multi-party: [HV17, NTY21, BMRR21, CDG+21, GPR+21, ENOP22, BHV+23, GTY24..]
Updatable: [KLS+17, ATD20, BMX22..]
Fuzzy PSI: [CFR+21,UCK+21 ..]

Structure-Aware Private Set Intersection (sa-PSI)

[GRS22, GRS23, vBP24] a variant of PSI where Alice's input has a publicly known structure

Examples - interval, ball or union of balls in some metric space, ...





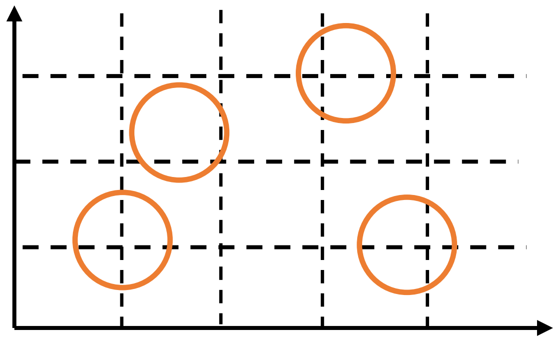
Structure-Aware PSI, with applications to Fuzzy Matching. CRYPTO 2022. *Gayathri Garimella*, Mike Rosulek and Jaspal Singh

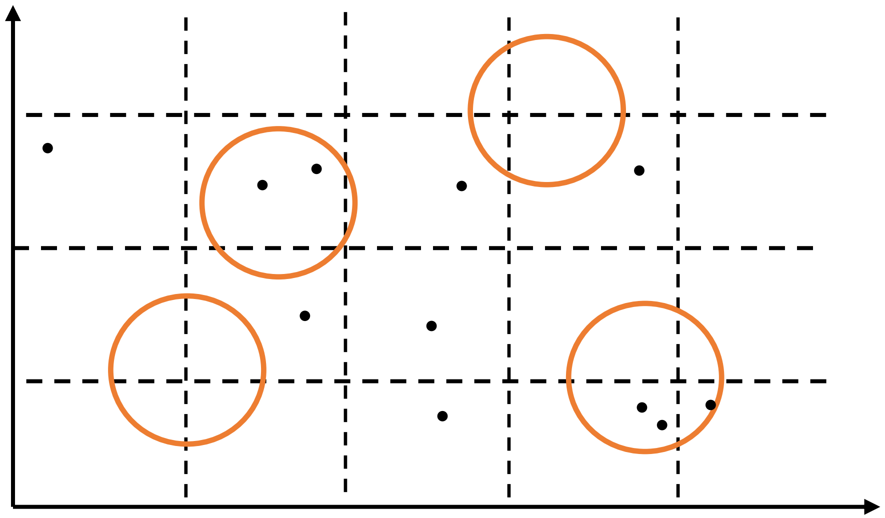
Structure-Aware Private Set Intersection (sa-PSI)

[GRS22, GRS23, vBP24] a variant of PSI where Alice's input has a publicly known structure

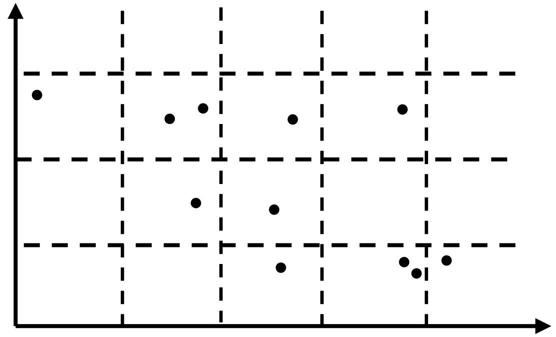
Examples - interval, ball or union of balls in some metric space, …

input A





input B

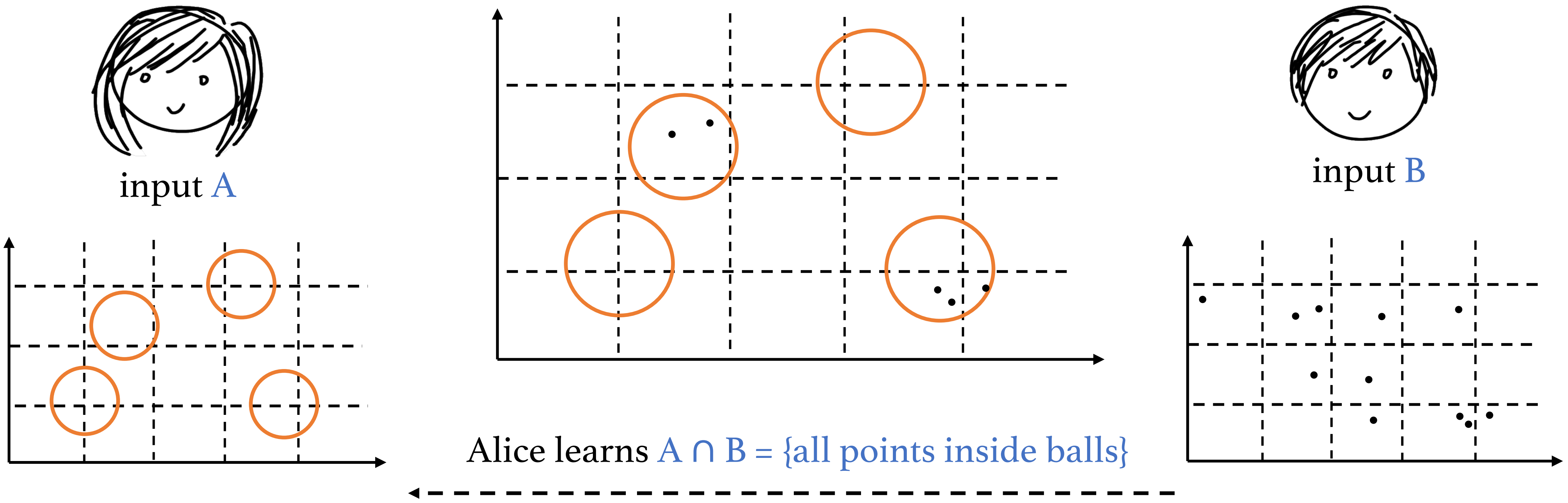


Structure-Aware PSI, with applications to Fuzzy Matching. CRYPTO 2022. Gayathri Garimella, Mike Rosulek and Jaspal Singh

Structure-Aware Private Set Intersection (sa-PSI)

[GRS22, GRS23, vBP24] a variant of PSI where Alice's input has a publicly known structure

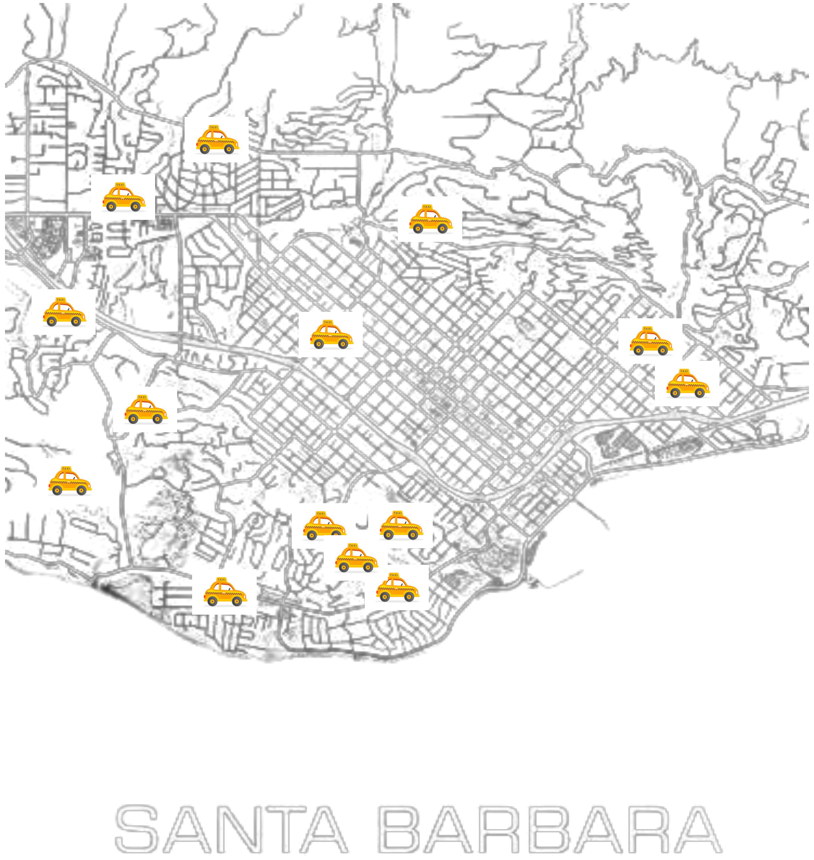
Examples - interval, ball or union of balls in some metric space, …



Structure-Aware PSI, with applications to Fuzzy Matching. CRYPTO 2022. *Gayathri Garimella*, Mike Rosulek and Jaspal Singh

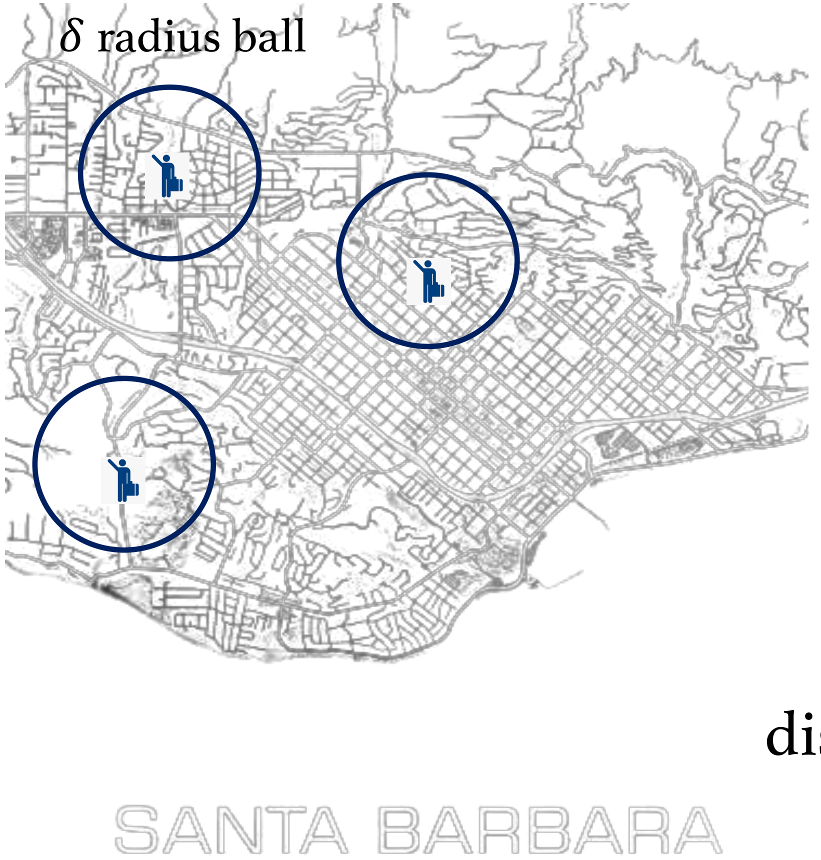
Fuzzy matching

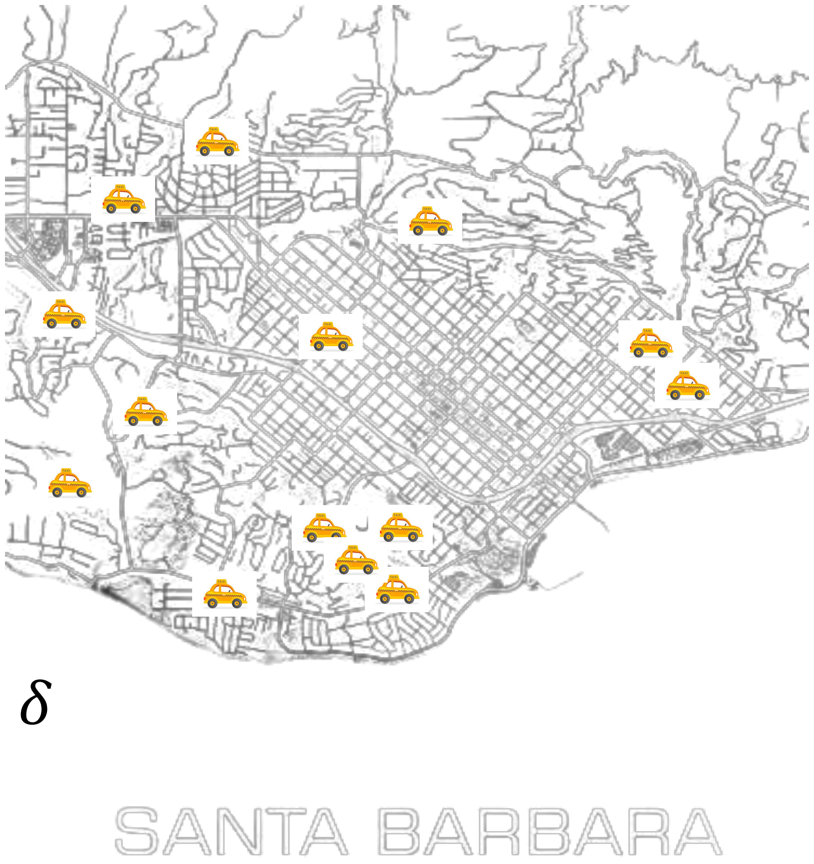
privacy-preserving ride hailing service



Fuzzy matching

privacy-preserving ride hailing service

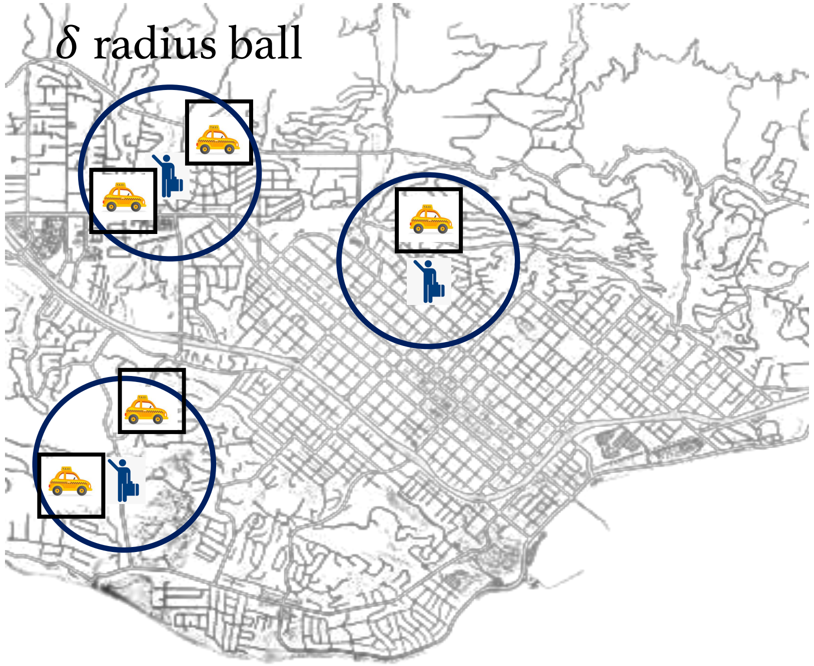


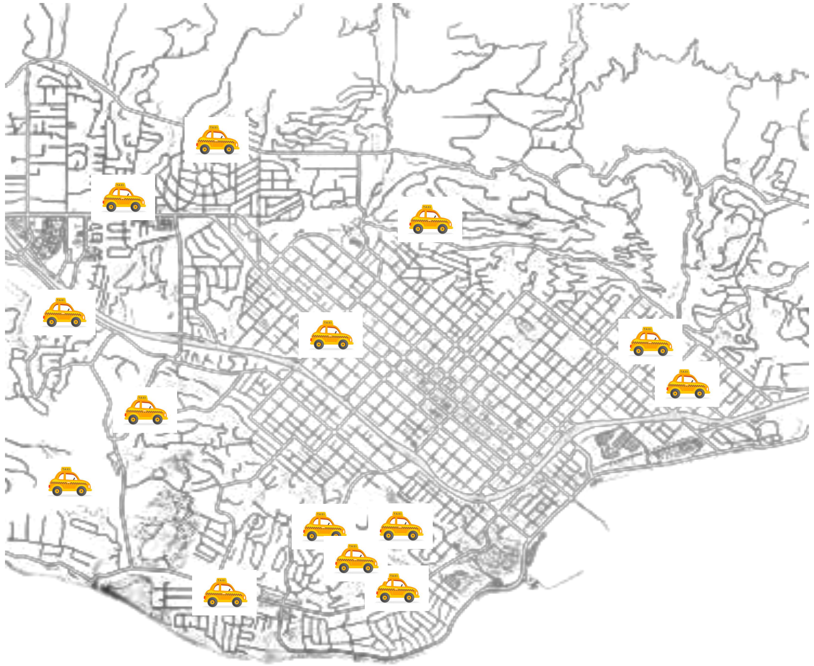


$$\mathrm{dist}(\;,\;) \leq \delta$$

Fuzzy matching

privacy-preserving ride hailing service

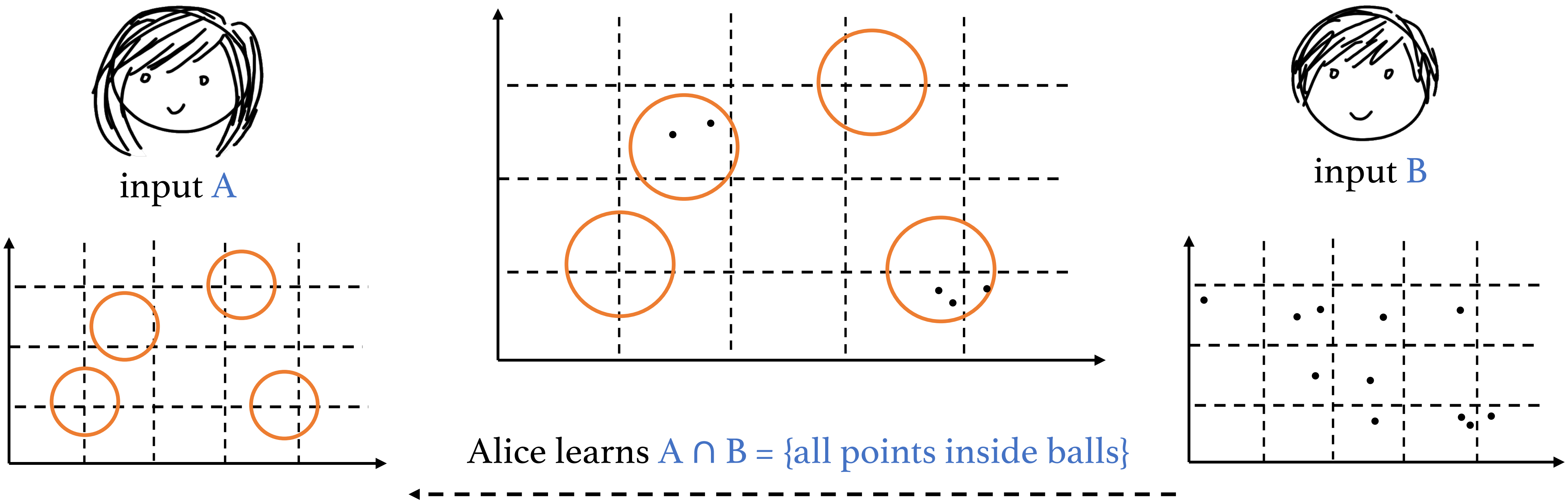




$$\mathrm{dist}(\ ,\ ) \leq \delta$$

Naïve solution

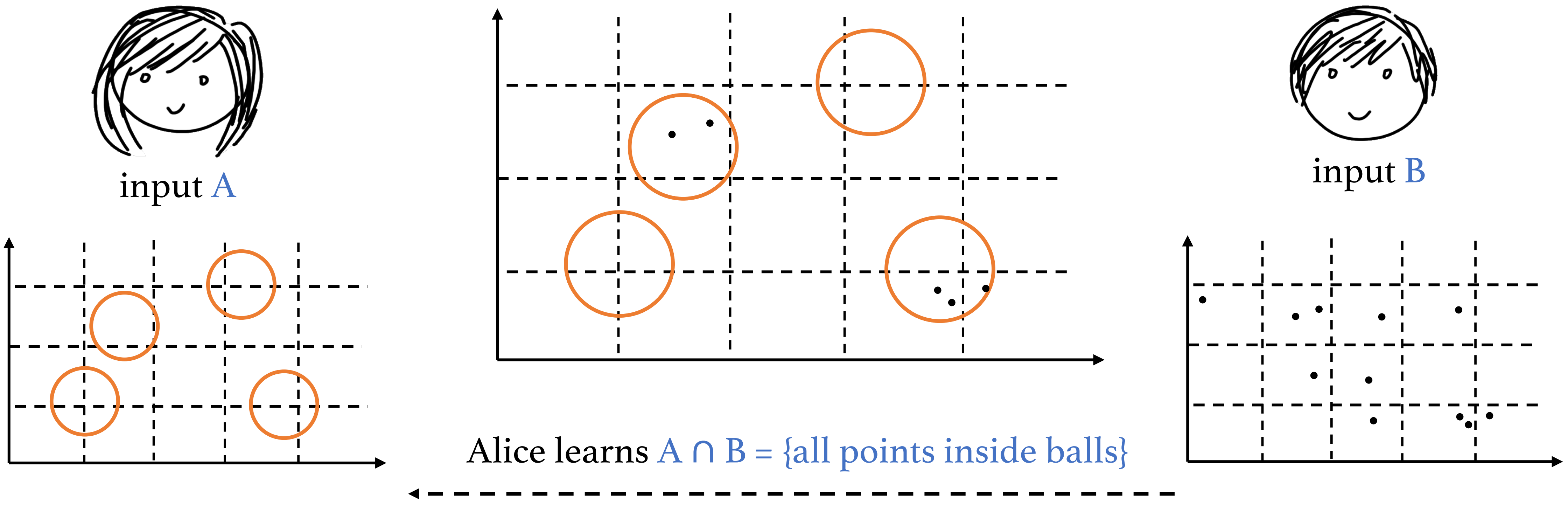
Alice's enumerates her structured input A, reduces to plain PSI



Naïve solution

Alice's enumerates her structured input A, reduces to plain PSI

Comm and / or Comp cost $O((|A| + |B|).\,\kappa)$, ~total volume $|A|$ of balls in Alice's input



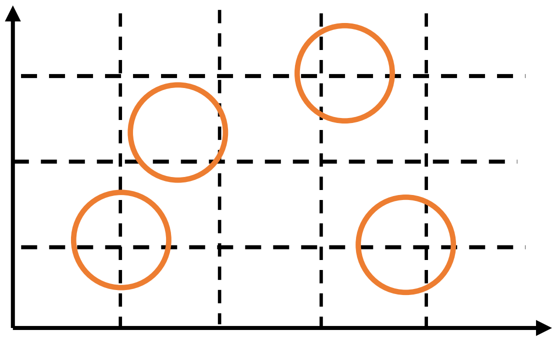
Naïve solution

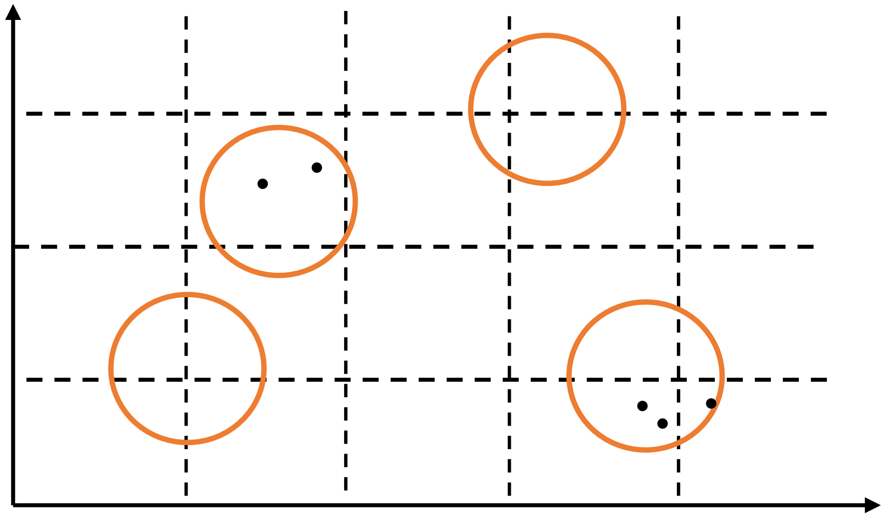
Alice's enumerates her structured input A, reduces to plain PSI

Comm and / or Comp cost $O((|A| + |B|). \kappa)$, ~total volume $|A|$ of balls in Alice's input

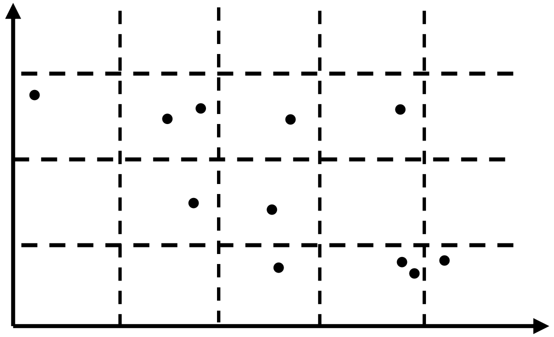
instead, cost scale with # of balls (description size) in Alice's input?

input A





input B



Alice learns A ∩ B = {all points inside balls}

State of the art

public knowledge: structured set family

[GarimellaRosulekSingh'22]
communication-efficient
Structure-aware PSI framework
(semi-honest)

↑ from

advantage: lightweight symmetric key based operations

*Structure-Aware PSI, with applications to Fuzzy Matching. CRYPTO 2022. *Gayathri Garimella*, Mike Rosulek and Jaspal Singh

State of the art

[GarimellaRosulekSingh'22]
communication-efficient
Structure-aware PSI framework
(semi-honest)

limitation

computation scales with Alice's set size |A|

from

advantage: lightweight symmetric key based operations

*Structure-Aware PSI, with applications to Fuzzy Matching. CRYPTO 2022. *Gayathri Garimella*, Mike Rosulek and Jaspal Singh

Our Contribution

public knowledge: structured set family

[GarimellaGoffMiao'24]
computation and
communication efficient
Structure-aware PSI framework
(semi-honest)

↑ from

advantage: lightweight symmetric key based operations

For the rest of the talk..

[GarimellaRosulekSingh'22]
communication-efficient
<u>Structure-aware PSI framework</u>
(semi-honest)

↑ from

boolean Function Secret Sharing
+ Oblivious Transfer

[GarimellaGoffMiao'24]
computation and
communication efficient
<u>Structure-aware PSI framework</u>
(semi-honest)

↑ from

incremental
boolean Function Secret Sharing
+ Oblivious Transfer

For the rest of the talk..

[GarimellaRosulekSingh'22]
communication-efficient
<u>Structure-aware PSI framework</u>
(semi-honest)

from

1. What?
boolean Function Secret Sharing
+ Oblivious Transfer

[GarimellaGoffMiao'24]
computation and
communication efficient
<u>Structure-aware PSI framework</u>
(semi-honest)

from

incremental
boolean Function Secret Sharing
+ Oblivious Transfer

For the rest of the talk..

[GarimellaRosulekSingh’22]
communication-efficient
Structure-aware PSI framework
(semi-honest)

2. How ? from

1. What ? boolean Function Secret Sharing
+ Oblivious Transfer

[GarimellaGoffMiao’24]
computation and
communication efficient
Structure-aware PSI framework
(semi-honest)

from

incremental
boolean Function Secret Sharing
+ Oblivious Transfer

For the rest of the talk..

[GarimellaRosulekSingh'22]
communication-efficient
<u>Structure-aware PSI framework</u>
(semi-honest)

2. How ? from

1. What ?
boolean Function Secret Sharing
\+ Oblivious Transfer

[GarimellaGoffMiao'24]
computation and
communication efficient
<u>Structure-aware PSI framework</u>
(semi-honest)

from

3. What ?
incremental
boolean Function Secret Sharing
\+ Oblivious Transfer

For the rest of the talk..

[GarimellaRosulekSingh'22]
communication-efficient
Structure-aware PSI framework
(semi-honest)

2. How ? from

1. What ?
boolean Function Secret Sharing
+ Oblivious Transfer

[GarimellaGoffMiao'24]
computation and
communication efficient
Structure-aware PSI framework
(semi-honest)

from 4. How ?

3. What ?
incremental
boolean Function Secret Sharing
+ Oblivious Transfer

First, we look at

[GarimellaRosulekSingh'22]
communication-efficient
<u>Structure-aware PSI framework</u>
(semi-honest)

from

1. What ?
boolean Function Secret Sharing
+ Oblivious Transfer

[GarimellaGoffMiao'24]
computation and
communication efficient
<u>Structure-aware PSI framework</u>
(semi-honest)

from

incremental
boolean Function Secret Sharing
+ Oblivious Transfer

boolean Function Secret Sharing

given input A ∈ $\mathcal{S}$ from a class of structured sets

boolean Function Secret Sharing (bFSS)

[BoyleGilboaIshai15] – style FSS for set membership in A function

boolean Function Secret Sharing

given input $A \in \mathcal{S}$ from a class of structured sets

<u>boolean Function Secret Sharing</u> (bFSS)

[BoyleGilboaIshai15] – style FSS for set membership in A function

share(A) $\longrightarrow$, where , $\approx \$\$$

$\forall x \in A \Longrightarrow$ **ev**(, x) $\oplus$ **ev**(, x) $= 0$

$\forall x \notin A \Longrightarrow$ **ev**(, x) $\oplus$ **ev**(, x) $= 1$

boolean Function Secret Sharing

given input $A \in \mathcal{S}$ from a class of structured sets

boolean Function Secret Sharing (bFSS)

[BoyleGilboaIshai15] – style FSS for set membership in A function

share(A) $\longrightarrow$, where , $\approx \$\$$

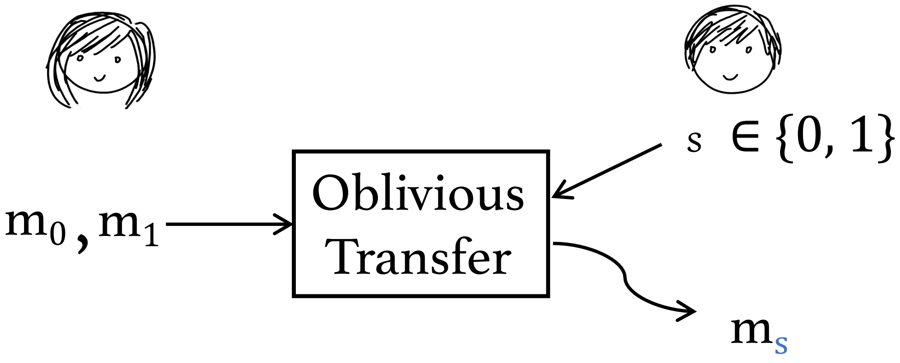
$\forall x \in A \Longrightarrow$ **ev**(, x) $\oplus$ **ev**(, x) $= 0$

$\forall x \notin A \Longrightarrow$ **ev**(, x) $\oplus$ **ev**(, x) $= 1$

succinctness: $| \; |, | \; | = \sigma << |A|$

[BGI15, BGI16, BCG+21, BGIK22] - PRG based constructions for set family membership functions like {singleton, 1-d interval, d-dimensional interval..}

Oblivious Transfer [Rabin'81]



OT can be instantiated efficiently (largely using symmetric key operations) from OT extension [IKNP03]

How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, 2005. Michael O Rabin.
Extending Oblivious Transfers Efficiently. CRYPTO 2003. Yuval Ishai, Joe Kilian, Kobbi Nissim and Erez Petrank.

Now, let's see how to realize sa-PSI

[GarimellaRosulekSingh'22]
communication-efficient
<u>Structure-aware PSI framework</u>
(semi-honest)

2. How ? ↑ from

boolean Function Secret Sharing
\+ Oblivious Transfer

[GarimellaGoffMiao'24]
computation and
communication efficient
<u>Structure-aware PSI framework</u>
(semi-honest)

from ↑

incremental
boolean Function Secret Sharing
\+ Oblivious Transfer

How does [GRS'22] work?

assumptions: OT-hybrid (Oblivious Transfer[Rabin81]), hamming correlation robust hash

input A

input B

How does [GRS'22] work?

input A

1. generates κ instances of bFSS shares

input B

2. picks κ choice bits to learn or

share(A) $\longrightarrow$ 1 , 1

share(A) $\longrightarrow$ 2 , 2

$\vdots$

share(A) $\longrightarrow$ κ , κ

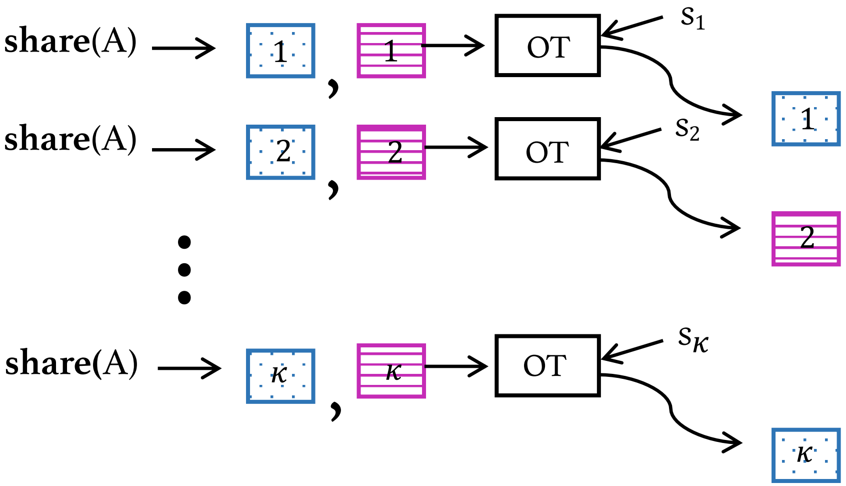
How does [GRS'22] work?

input A

1. generates κ instances of bFSS shares

input B

2. picks κ choice bits to learn $\boxed{}$ or $\boxed{}$



Bob computes F(x) on all his inputs

$$\boxed{F(x) = \mathbf{H}(\mathbf{ev}(\boxed{1}, x) \| \mathbf{ev}(\boxed{2}, x) \| \cdots \mathbf{ev}(\boxed{\kappa}, x))}$$

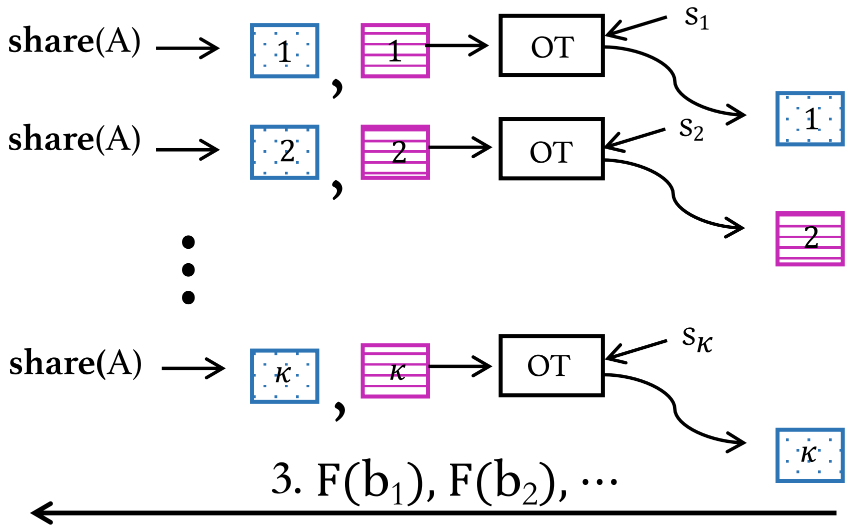
How does [GRS'22] work?

input A

1. generates κ instances of bFSS shares

input B

2. picks κ choice bits to learn or



Bob computes $F(x)$ on all his inputs

$$F(x) = \mathbf{H}(\mathbf{ev}(\boxed{1}, x) \| \mathbf{ev}(\boxed{2}, x) \| \cdots \mathbf{ev}(\boxed{\kappa}, x))$$

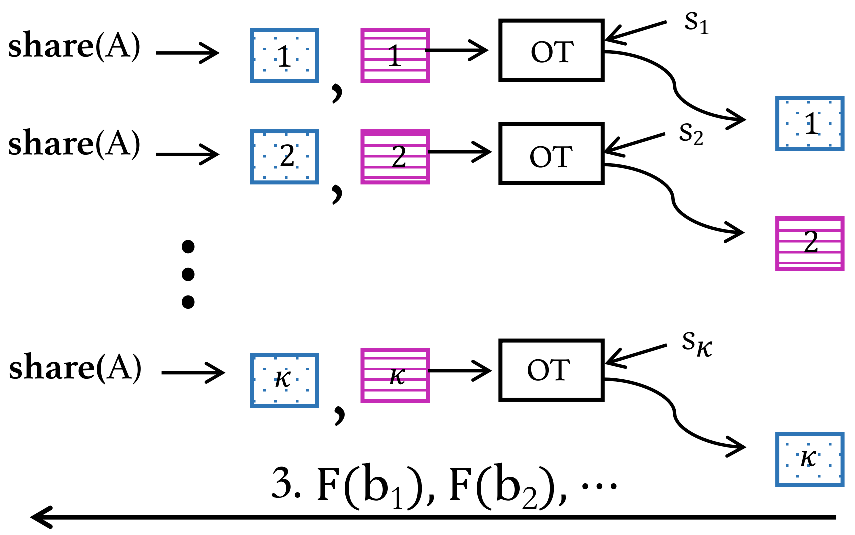
How does [GRS'22] work?

input A

1. generates κ instances of bFSS shares

input B

2. picks κ choice bits to learn $\boxed{\ }$ or $\boxed{\ }$



if $x \in A \implies$ Alice can compute $F(x)$

if $x \notin A \implies F(x) \approx \$\$$ looks random to Alice

$$\boxed{\text{if } x \in A \implies \mathbf{ev}(\boxed{\ }, x) = \mathbf{ev}(\boxed{\ }, x)}$$

Bob computes $F(x)$ on all his inputs

$$\boxed{F(x) = \mathbf{H}(\mathbf{ev}(\boxed{1}, x) \| \mathbf{ev}(\boxed{2}, x) \| \cdots \mathbf{ev}(\boxed{\kappa}, x))}$$

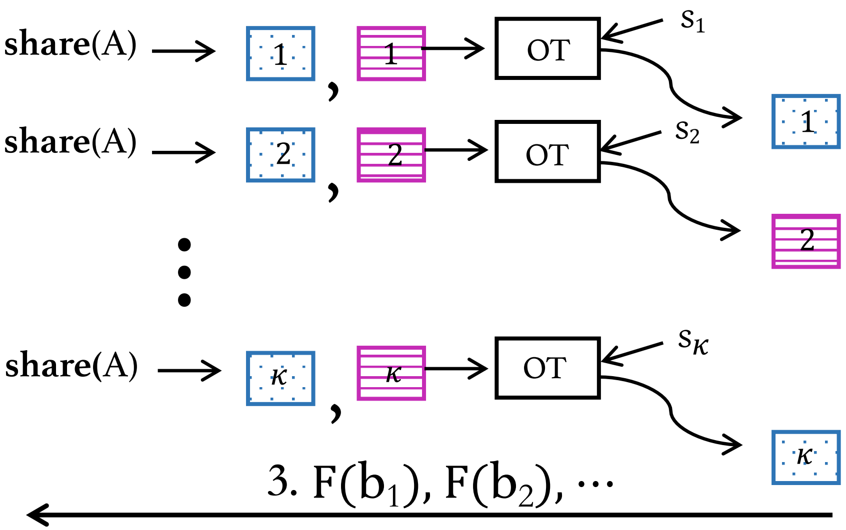
How does [GRS'22] work?

input A

1. generates κ instances of bFSS shares

input B

2. picks κ choice bits to learn $\boxed{}$ or $\boxed{}$



3. $\forall a \in A$, compute $F(a)$
4. locally compare to learn intersection

$$\text{if } x \in A \implies \mathbf{ev}(\boxed{}, x) = \mathbf{ev}(\boxed{}, x)$$

Bob computes $F(x)$ on all his inputs

$$F(x) = \mathbf{H}(\mathbf{ev}(\boxed{1}, x) \| \mathbf{ev}(\boxed{2}, x) \| \cdots \mathbf{ev}(\boxed{\kappa}, x))$$

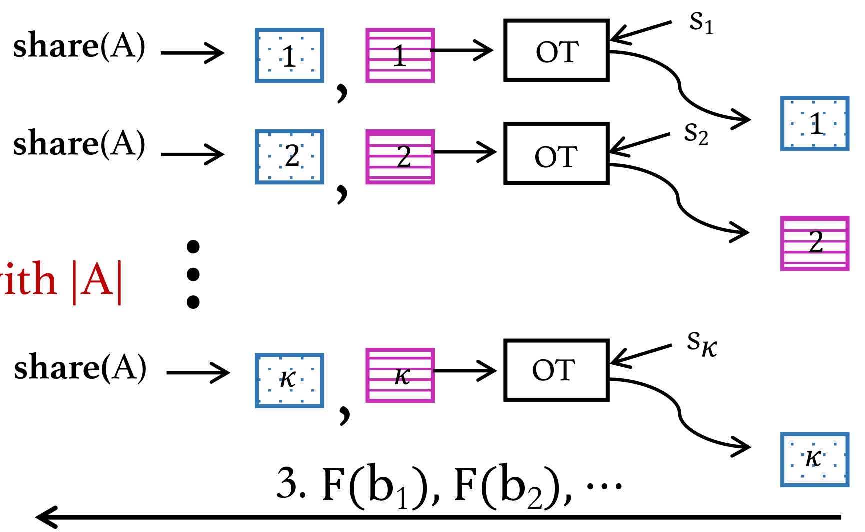
Computation scales with structured set size

input A

1. generates κ instances of bFSS shares

input B

2. picks κ choice bits to learn ▫ or ▤



computation scales with |A|

3. $\forall a \in A$, compute F(a)
4. locally compare to learn intersection

$$\boxed{\text{if } x \in A \implies \mathbf{ev}(\boxed{}, x) = \mathbf{ev}(\boxed{}, x)}$$

Bob computes F(x) on all his inputs

$$\boxed{F(x) = \mathbf{H}(\mathbf{ev}(\boxed{1}, x) \| \mathbf{ev}(\boxed{2}, x) \| \cdots \mathbf{ev}(\boxed{\kappa}, x))}$$

We present a new framework

NEW approach!

[GarimellaRosulekSingh'22]
communication-efficient
Structure-aware PSI framework
(semi-honest)

from

boolean Function Secret Sharing
+ Oblivious Transfer

[GarimellaGoffMiao'24]
computation and
communication efficient
Structure-aware PSI framework
(semi-honest)

from

incremental
boolean Function Secret Sharing
+ Oblivious Transfer

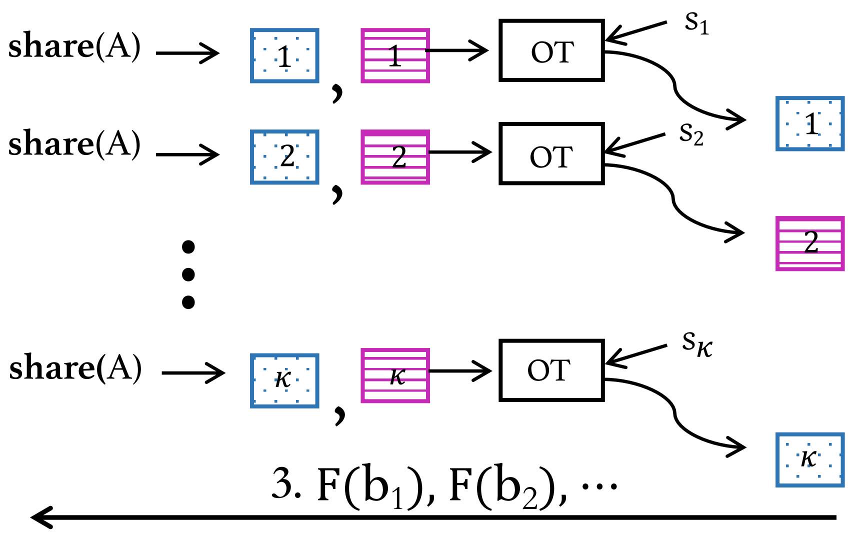
High-level idea

input A

1. generates κ instances of bFSS shares

input B

2. picks κ choice bits to learn or



+

specially crafted hints

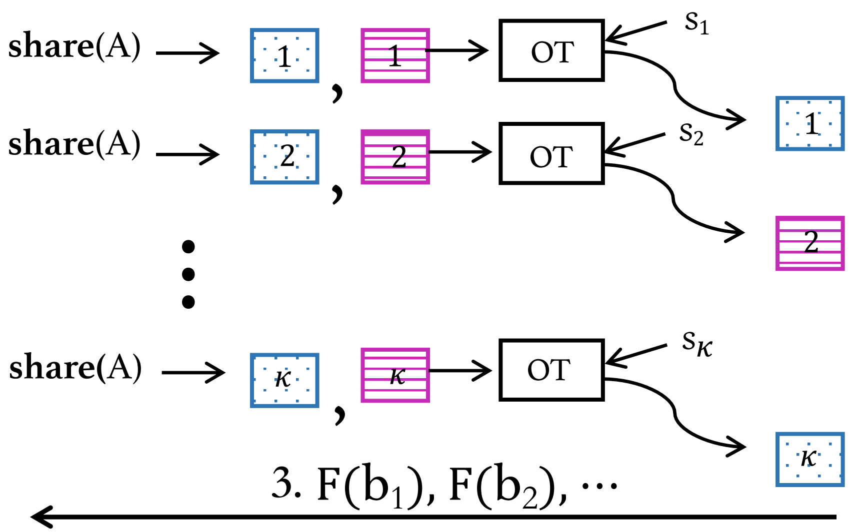
High-level idea

input A

1. generates κ instances of bFSS shares

input B

2. picks κ choice bits to learn or



3. can efficiently identify and search for matching values

+

specially crafted hints

Crafting hints from incremental bFSS

[GarimellaRosulekSingh'22]
communication-efficient
Structure-aware PSI framework
(semi-honest)

from

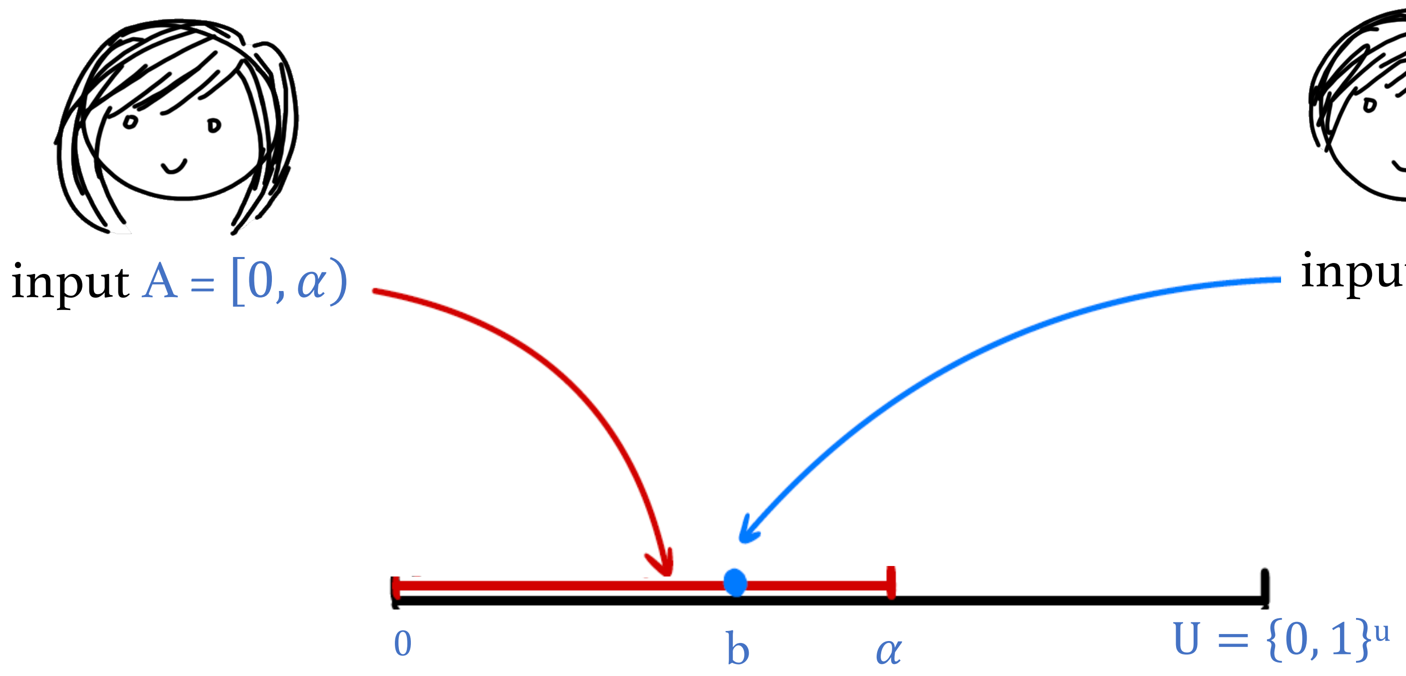
boolean Function Secret Sharing
+ Oblivious Transfer

[GarimellaGoffMiao'24]
computation and
communication efficient
Structure-aware PSI framework
(semi-honest)

from 4. How ?

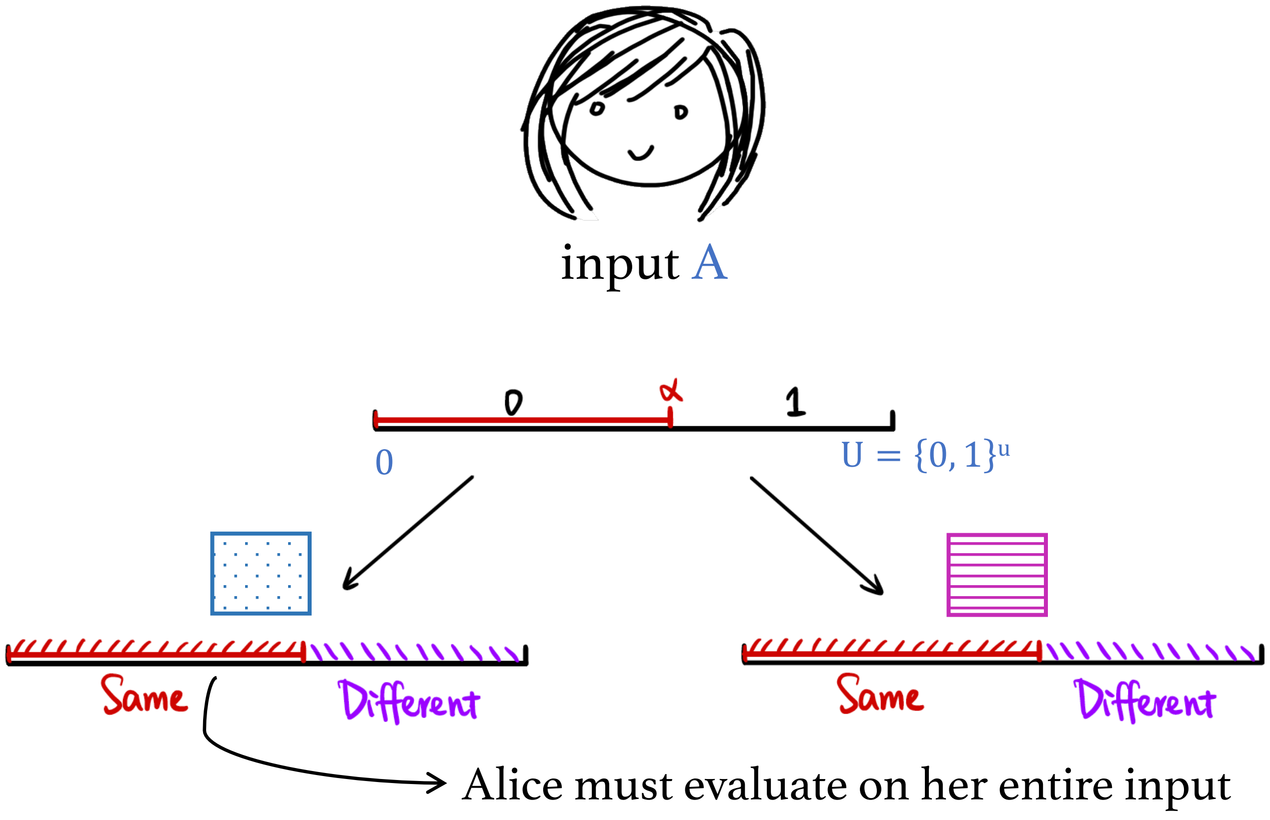
3. What ?
incremental
boolean Function Secret Sharing
+ Oblivious Transfer

One-sided Interval Single-Point sa-PSI

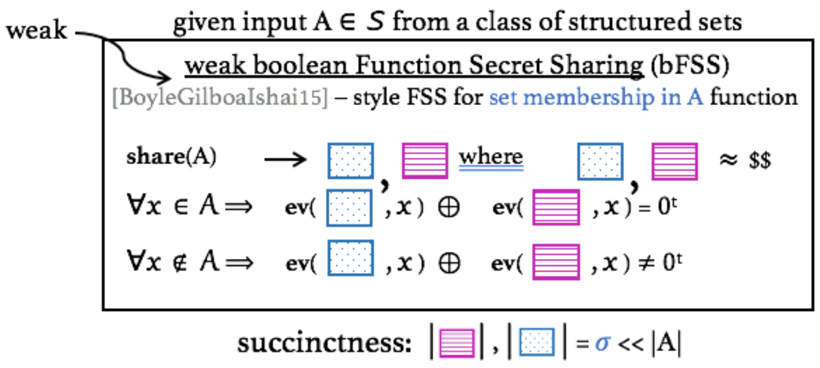


Alice learns output $A \cap B$

(previously) Boolean FSS for One-Sided Interval

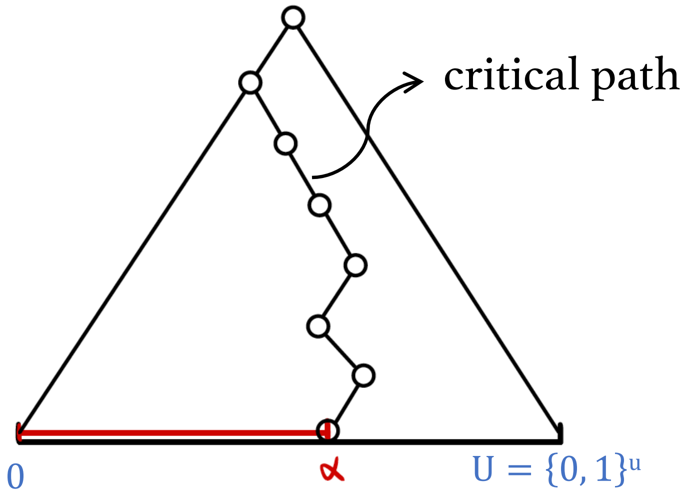


Recall definition:



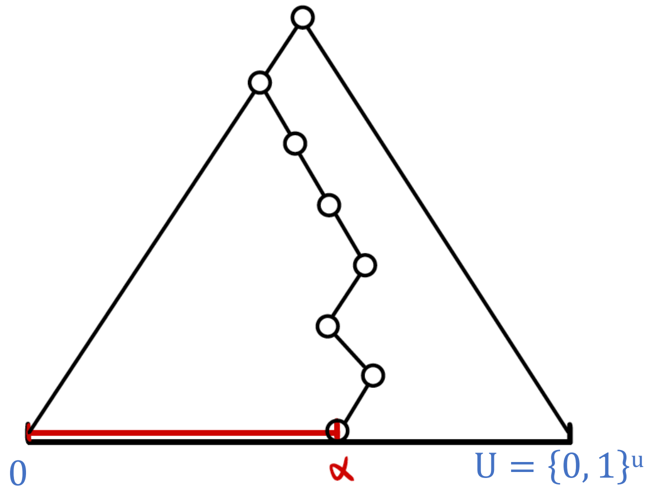
Identifying membership for One-sided Interval

How would we check if a point ‘b’ belongs to the interval?
bit-wise comparison with the critical path

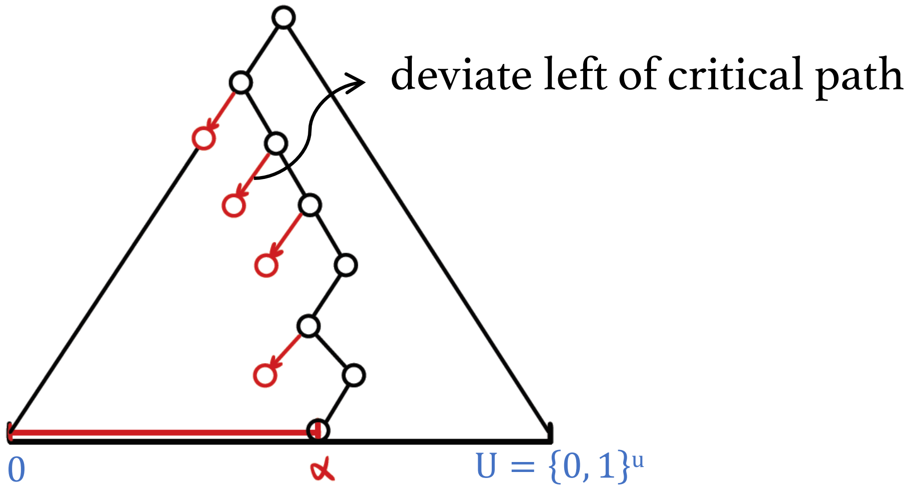


one-sided interval $[0, \alpha)$

Identifying membership for One-sided interval

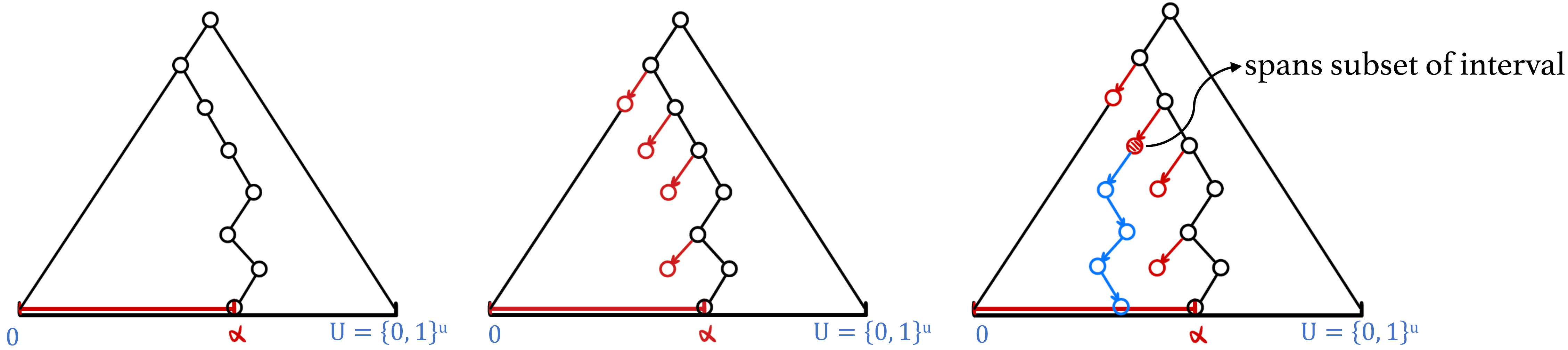


one-sided interval $[0, \alpha)$



critical prefixes span interval

Identifying membership for One-sided interval

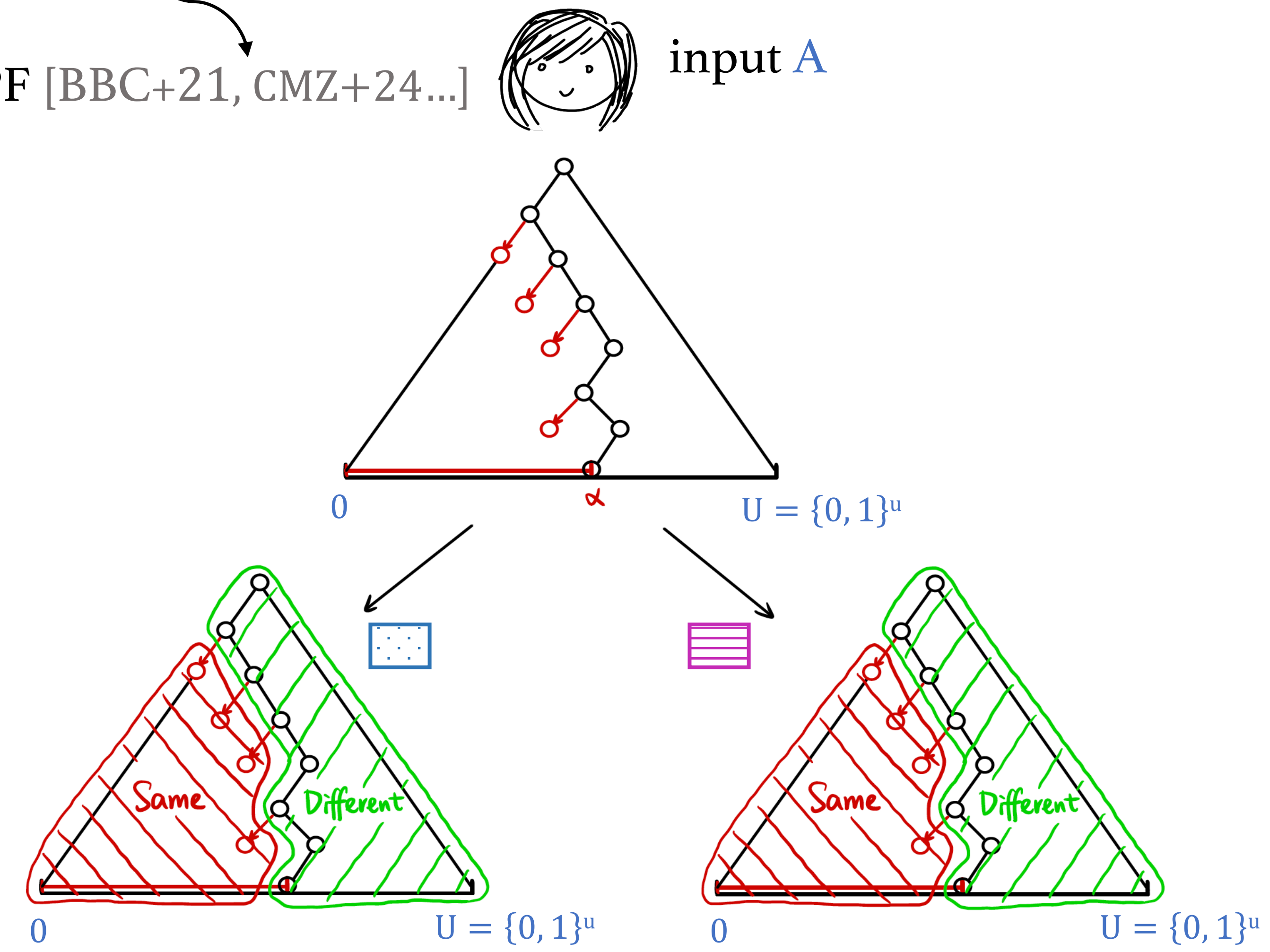


one-sided interval $[0, \alpha)$ critical prefixes span interval matching prefix implies set membership

Incremental Boolean FSS for One-Sided Interval

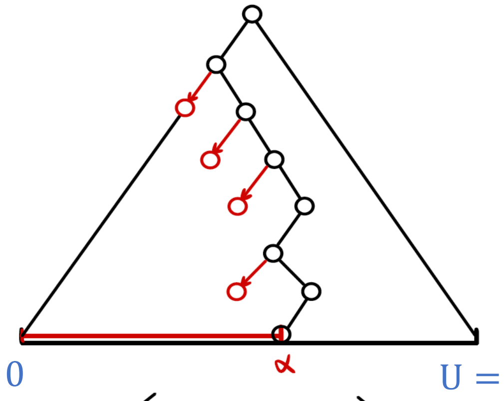
DPF [BBC+21, CMZ+24...]

input A



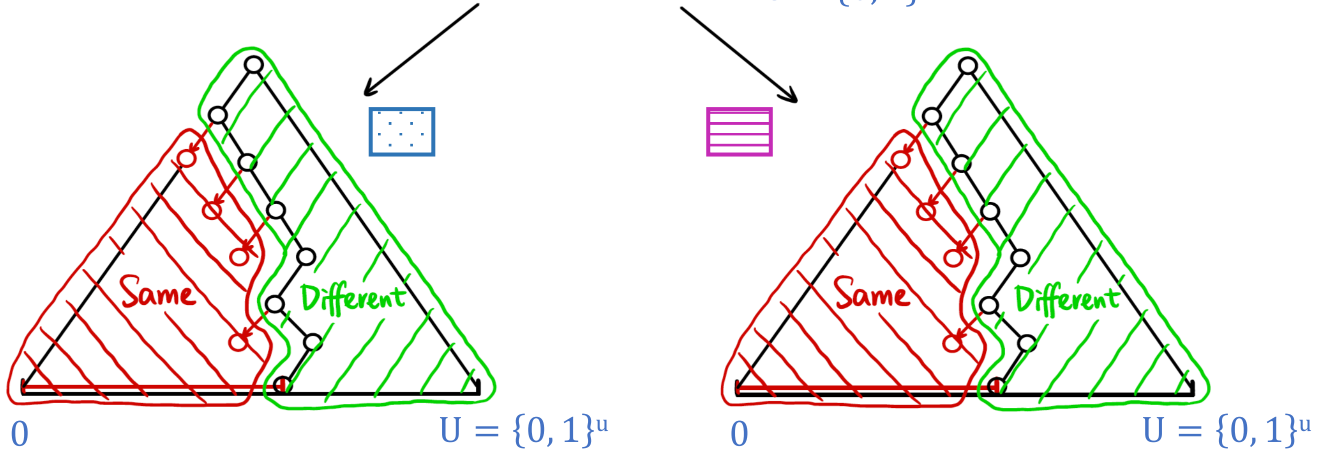
Incremental Boolean FSS for One-Sided Interval

input A

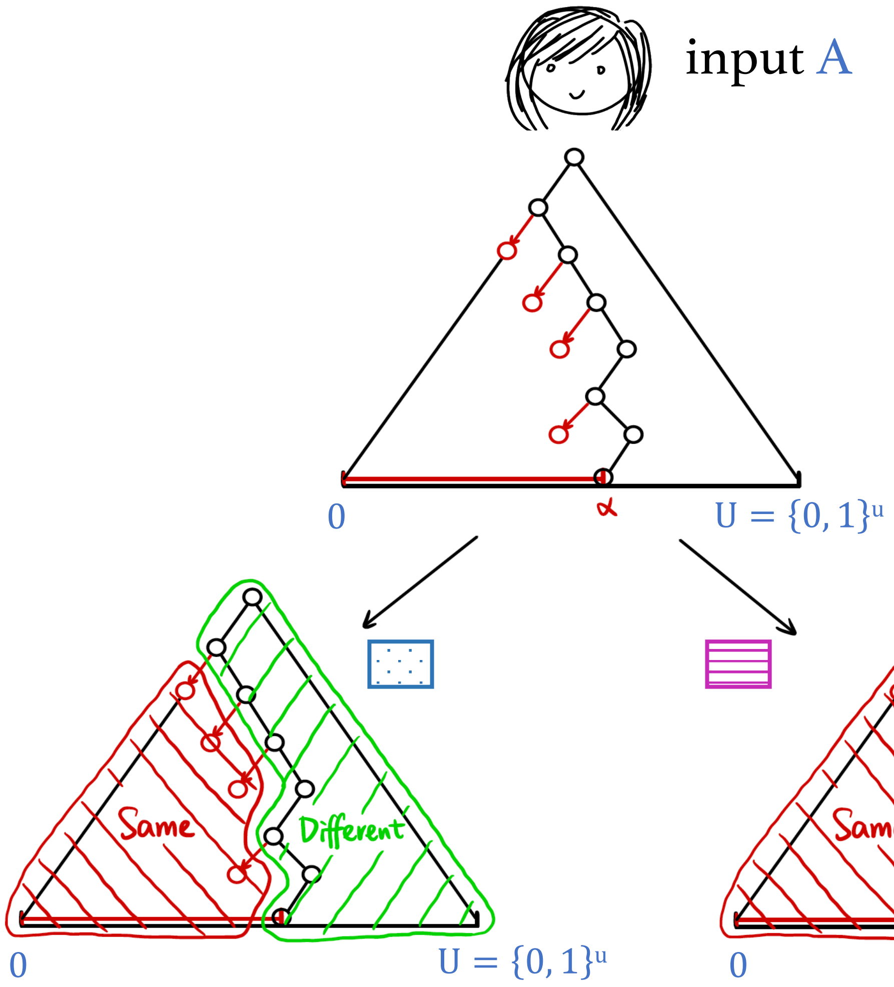


secret share such that:

- can evaluate on intermediate and leaf nodes
- evaluations match once you deviate left of critical path



Incremental Boolean FSS for One-Sided Interval



given input A ∈ $\mathcal{S}$ from a class of structured sets

<u>incremental boolean Function Secret Sharing</u> (ibFSS)

[BoyleGilboaIshai15] – style FSS for set membership in A function

share(A) ⟶ , where , ≈ $$

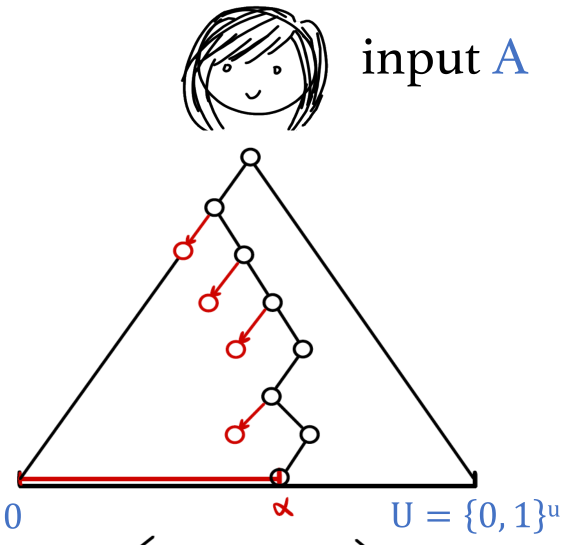
∀x with special prefix ⟹ **ev**(, x) ⊕ **ev**(, x) = 0^t

∀x w/o critical prefix ⟹ **ev**(, x) ⊕ **ev**(, x) ≠ 0^t

, where $x \in \bigcup_{l=0}^{u}\{0,1\}^l$

can evaluate on intermediate nodes

Incremental Boolean FSS for One-Sided Interval



given input A ∈ $\mathcal{S}$ from a class of structured sets

<u>incremental boolean Function Secret Sharing</u> (ibFSS)

[BoyleGilboaIshai15] – style FSS for set membership in A function

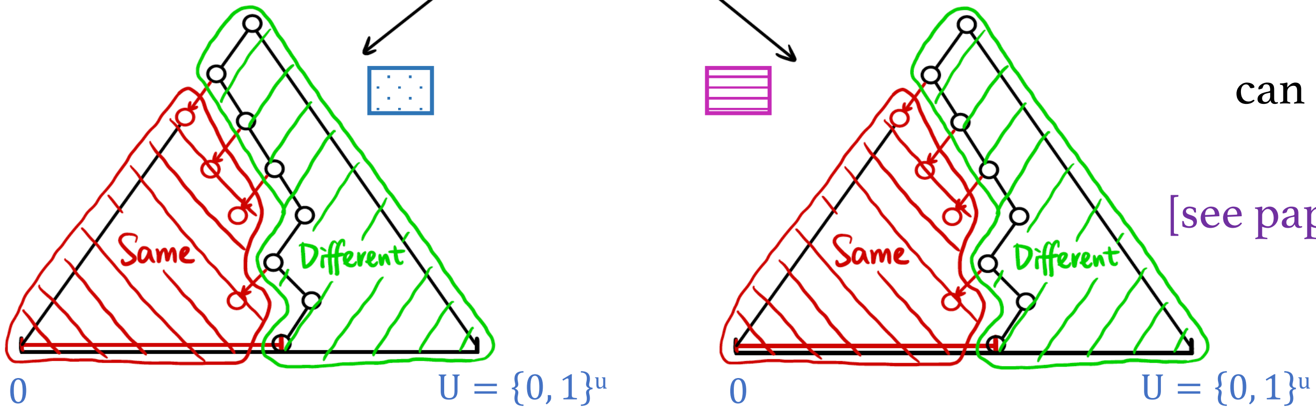
share(A) ⟶ , where , ≈ $$

$\forall x$ with special prefix ⟹ **ev**(, x) ⊕ **ev**(, x) = 0^t

$\forall x$ w/o critical prefix ⟹ **ev**(, x) ⊕ **ev**(, x) ≠ 0^t

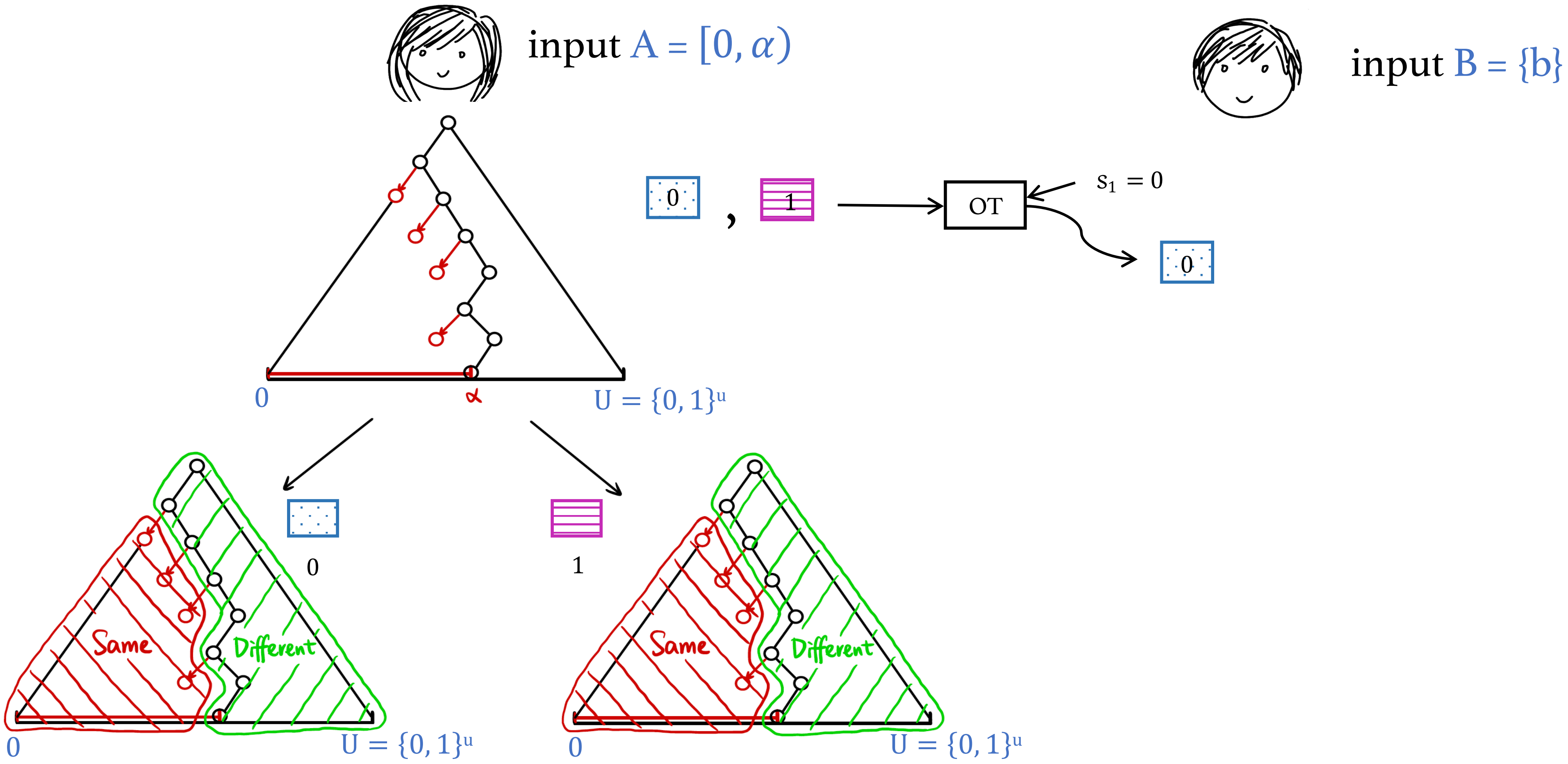
, where $x \in \cup_{l=0}^{u}\{0,1\}^l$

can evaluate on intermediate nodes

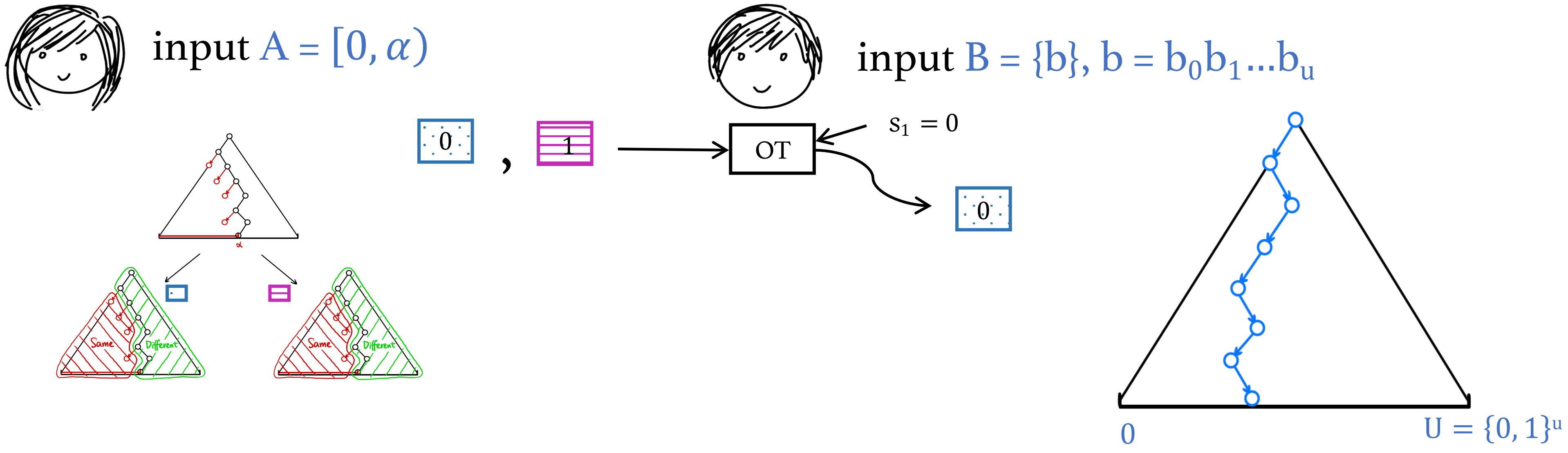


[see paper] for definition and constructions

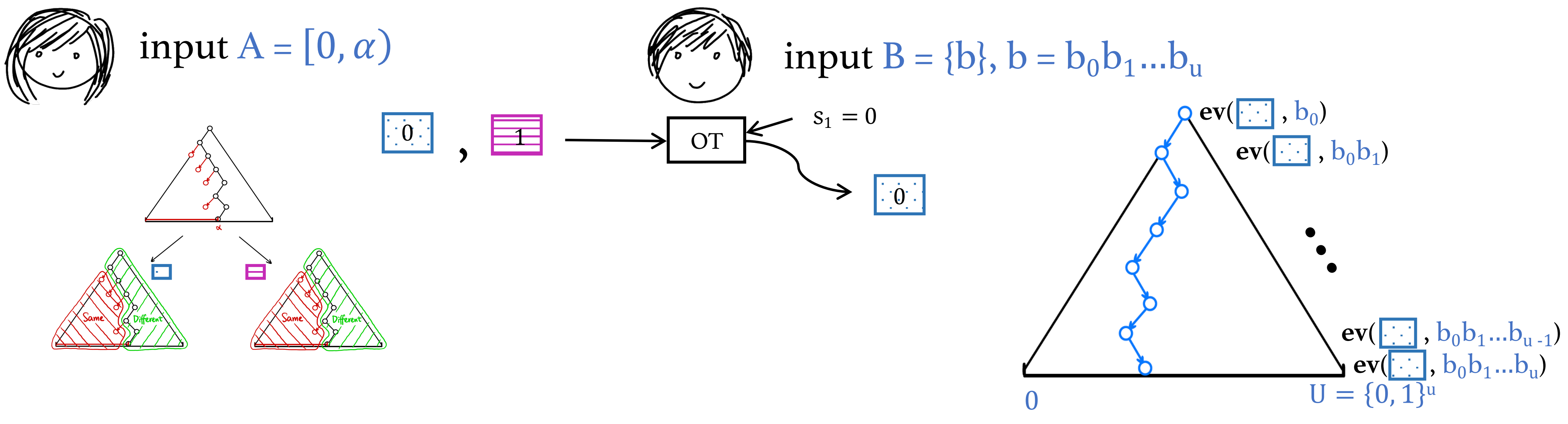
One-Sided Interval Single Point sa-PSI



One-Sided Interval Single Point sa-PSI

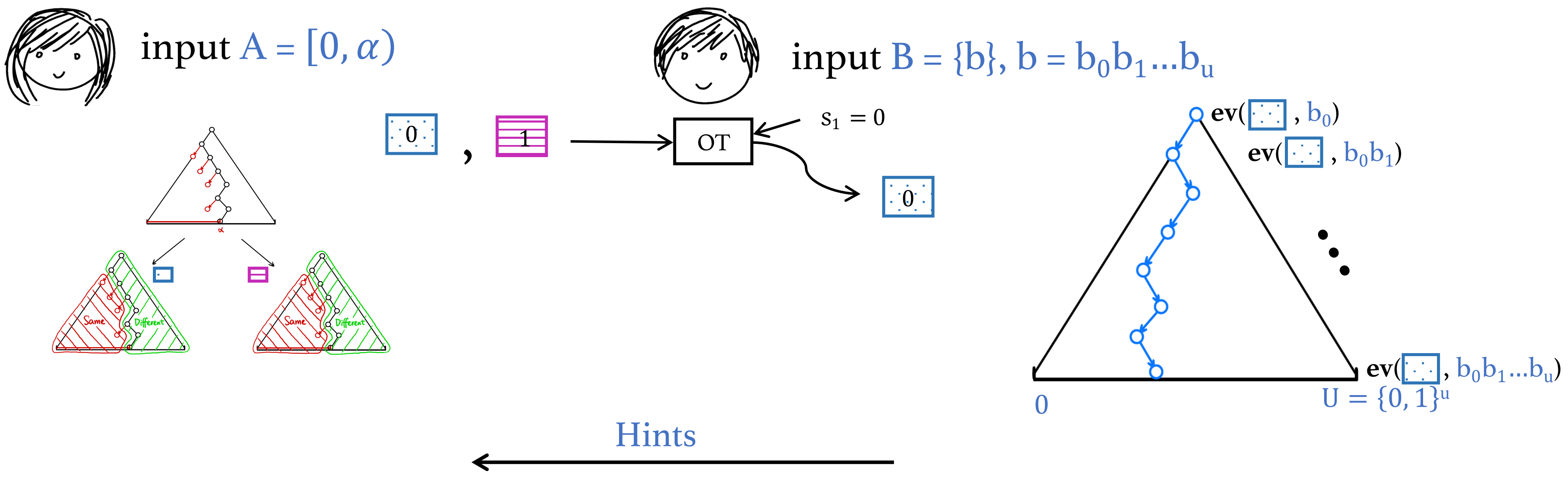


One-Sided Interval Single Point sa-PSI



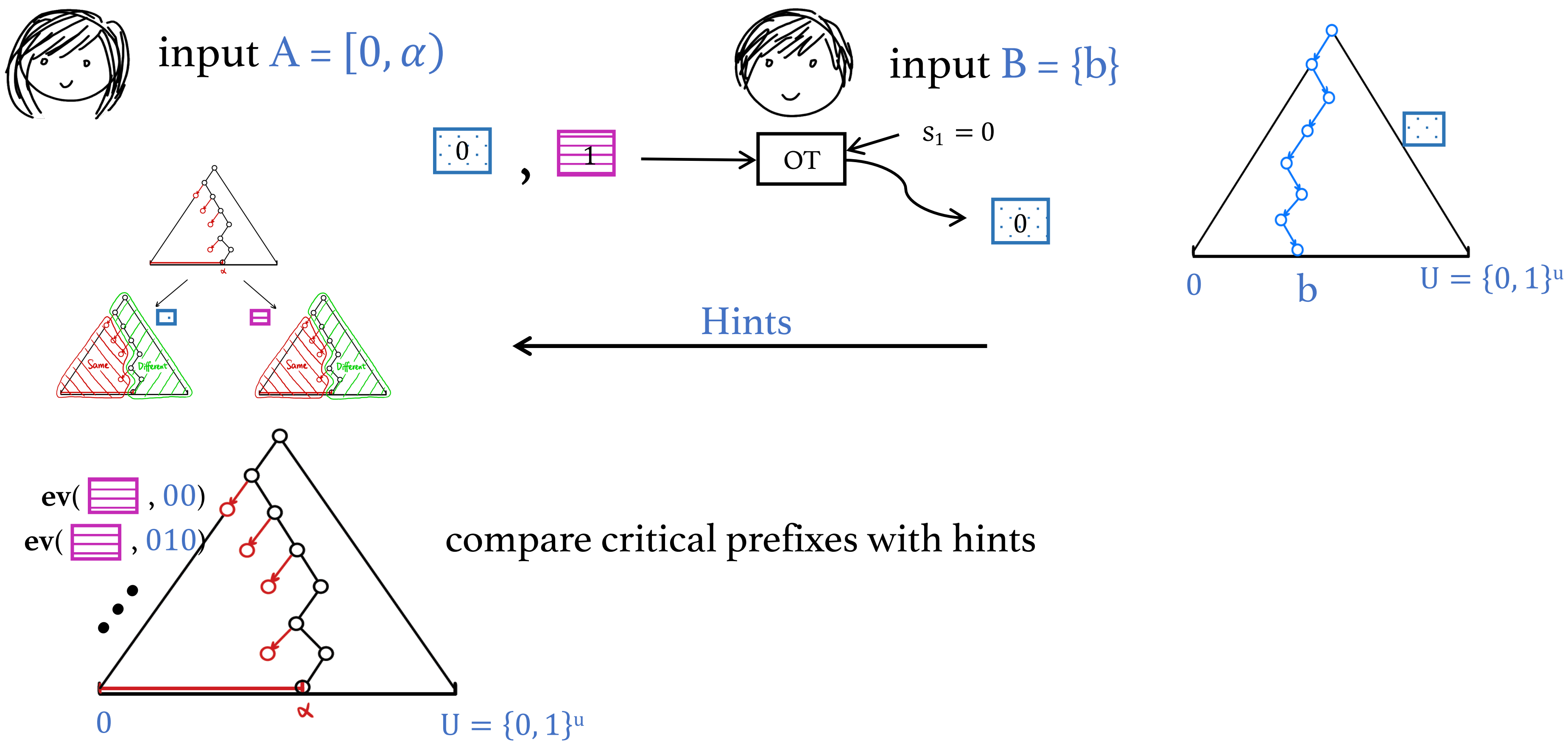
Hints: Bob evaluates incremental boolean FSS on every prefix of his input b

One-Sided Interval Single Point sa-PSI

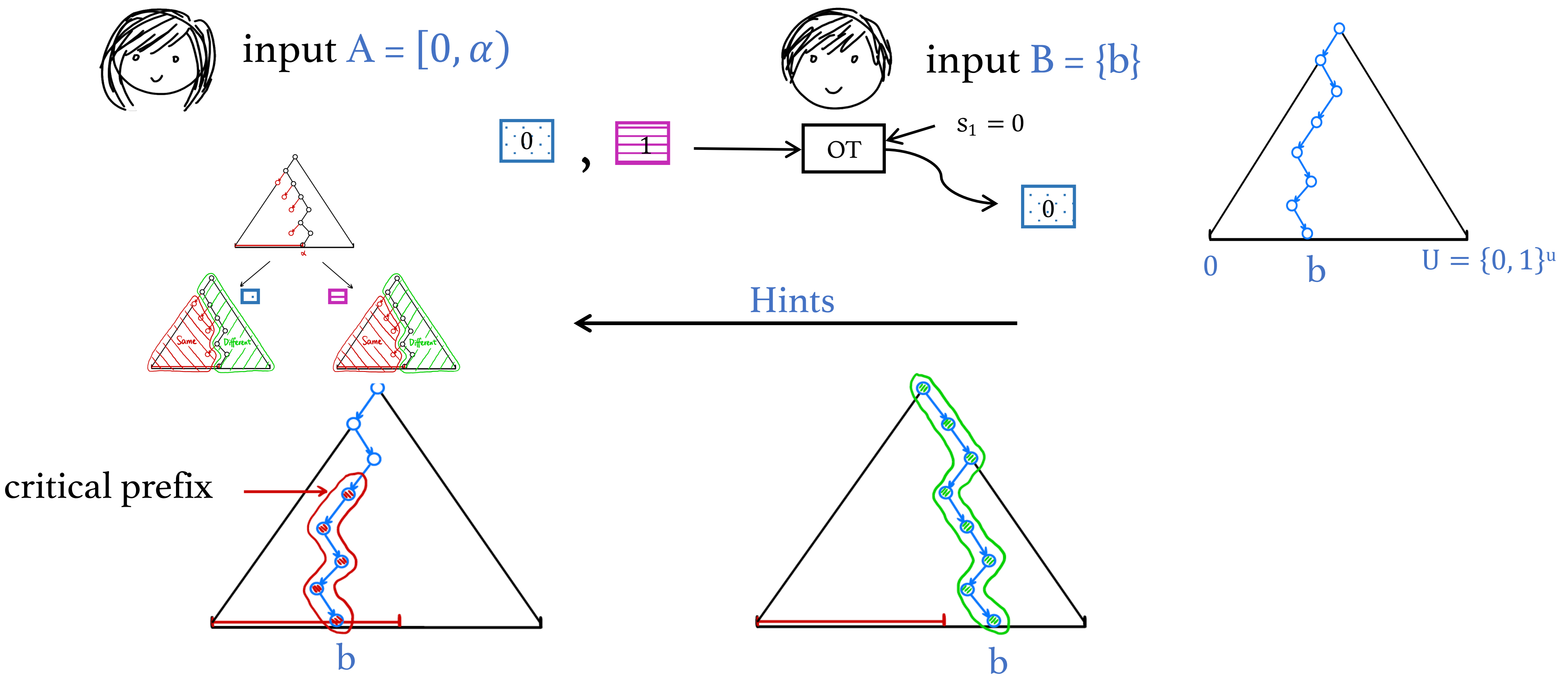


Hints: Bob evaluates incremental boolean FSS on every prefix of his input b

One-Sided Interval Single Point sa-PSI



One-Sided Interval Single Point sa-PSI



Case 1: computable by Alice

Case 2: looks random to Alice

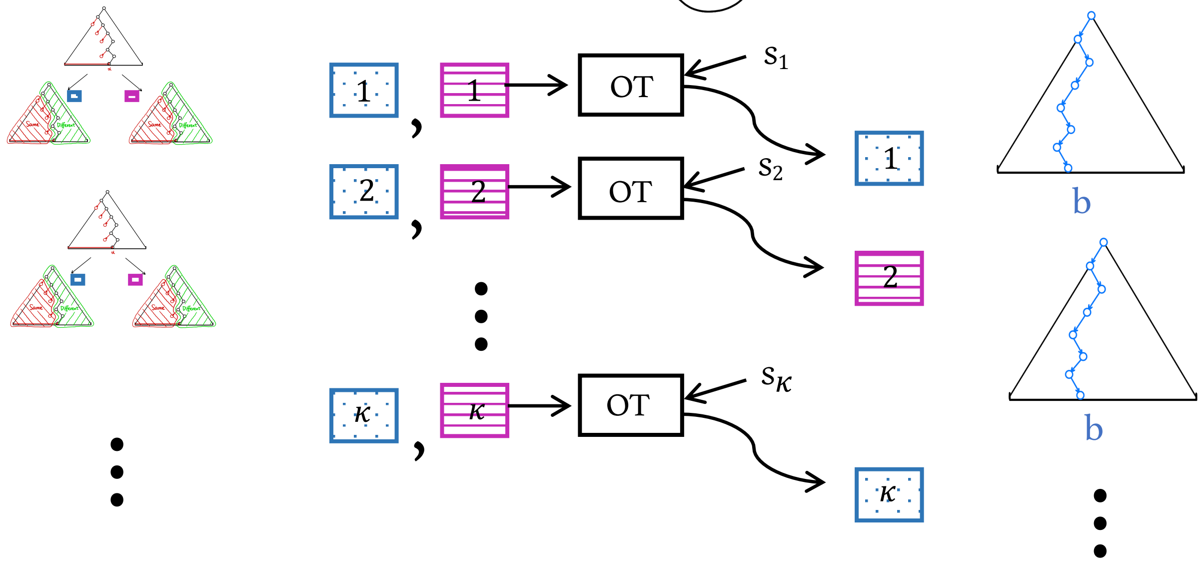
One-Sided Interval Single Point sa-PSI

input $A = [0, \alpha)$

1. generates κ incremental bFSS shares

input $B = \{b\}$

2. picks κ choice bits to learn or



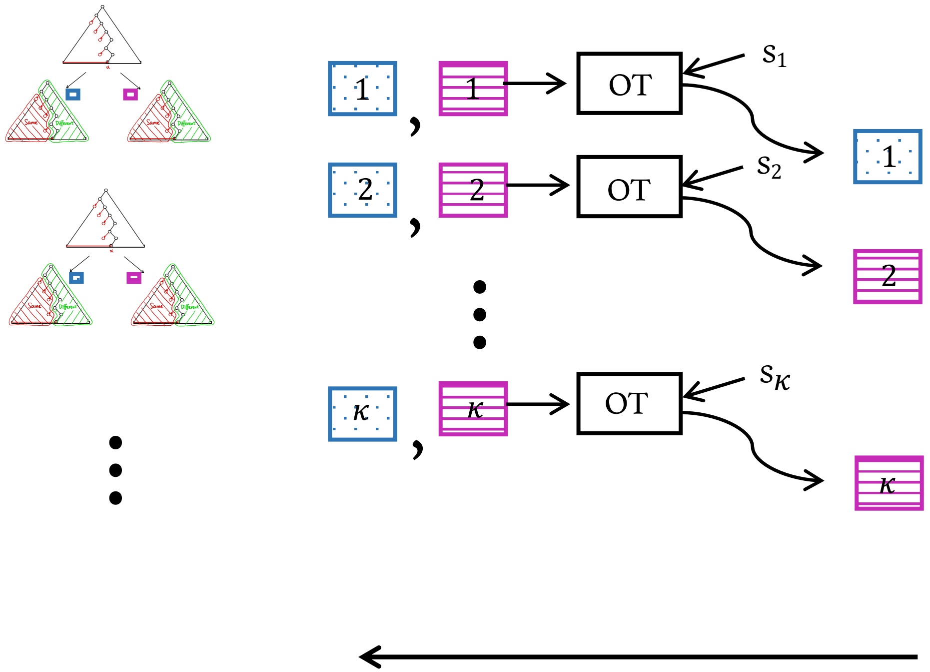
One-Sided Interval Single Point sa-PSI

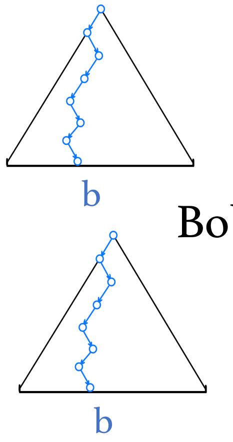
input $A = [0, \alpha)$

1. generates κ incremental bFSS shares

input $B = \{b\}$, $b = b_0 \ldots b_u$

2. picks κ choice bits to learn $\boxed{}$ or $\boxed{}$





Bob defines, computes $F(x)$ on all input prefixes

we have 'u' hints

$$F(b_0) = \mathbf{H}(\mathbf{ev}(\boxed{1}, b_0) \| \cdots \| \mathbf{ev}(\boxed{\kappa}, b_0))$$

$$F(b_0 b_1) = \mathbf{H}(\mathbf{ev}(\boxed{1}, b_0 b_1) \| \cdots \| \mathbf{ev}(\boxed{\kappa}, b_0 b_1))$$

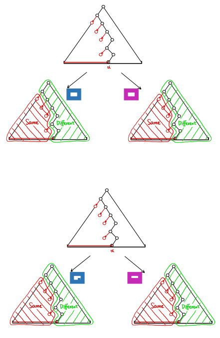
$\vdots$

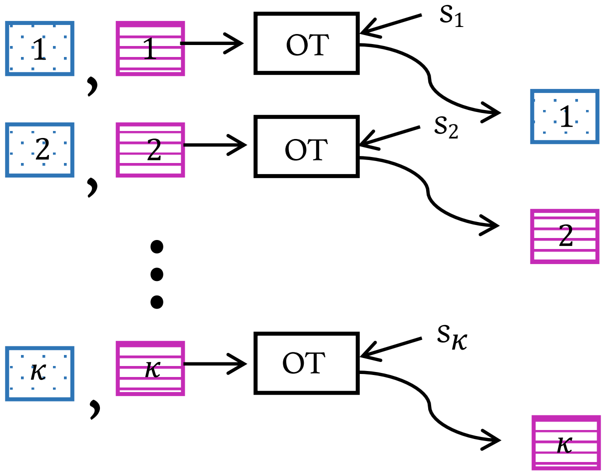
$$F(b_0 \ldots b_u) = \mathbf{H}(\mathbf{ev}(\boxed{1}, b_0 \ldots b_u) \| \cdots \| \mathbf{ev}(\boxed{\kappa}, b_0 \ldots b_u))$$

One-Sided Interval Single Point sa-PSI

input $A = [0, \alpha)$

1. generates κ incremental bFSS shares





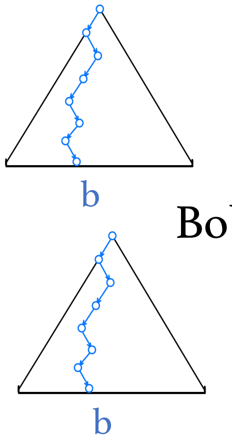
3. can compute $F(p_0)$,
if p_0 is a critical prefix of A

Hints ←

$$F(p_0) = \mathbf{H}(\mathbf{ev}(\boxed{1}, p_0) \| \cdots \| \mathbf{ev}(\boxed{\kappa}, p_0))$$

input $B = \{b\}$, $b = b_0 \ldots b_u$

2. picks κ choice bits to learn ⬚ or ▤



Bob defines, computes $F(x)$ on all input prefixes
we have 'u' hints

$$F(b_0) = \mathbf{H}(\mathbf{ev}(\boxed{1}, b_0) \| \cdots \| \mathbf{ev}(\boxed{\kappa}, b_0))$$

$$F(b_0 b_1) = \mathbf{H}(\mathbf{ev}(\boxed{1}, b_0 b_1) \| \cdots \| \mathbf{ev}(\boxed{\kappa}, b_0 b_1))$$

⋮

$$F(b_0 \ldots b_u) = \mathbf{H}(\mathbf{ev}(\boxed{1}, b_0 \ldots b_u) \| \cdots \| \mathbf{ev}(\boxed{\kappa}, b_0 \ldots b_u))$$

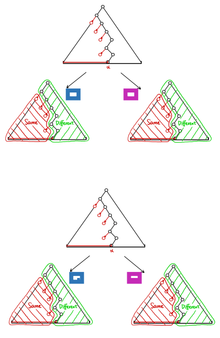
One-Sided Interval Single Point sa-PSI

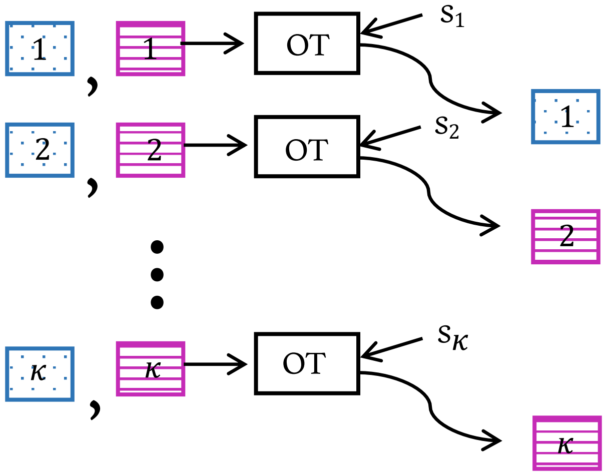
input $A = [0, \alpha)$

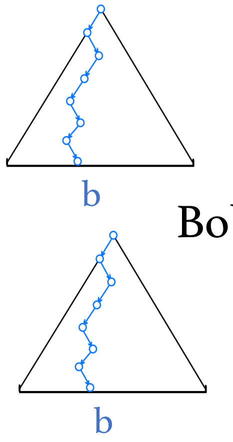
1. generates κ incremental bFSS shares

input $B = \{b\ (= b_0 \ldots b_u)\}$

2. picks κ choice bits to learn or







Bob defines, computes $F(x)$ on all input prefixes

we have 'u' hints

$F(b_0) = \mathbf{H}(\mathbf{ev}(\boxed{1}, b_0) \| \cdots \| \mathbf{ev}(\boxed{\kappa}, b_0))$

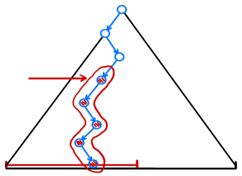
$F(b_0 b_1) = \mathbf{H}(\mathbf{ev}(\boxed{1}, b_0 b_1) \| \cdots \| \mathbf{ev}(\boxed{\kappa}, b_0 b_1))$

$\vdots$

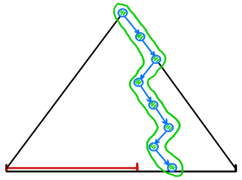
$F(b_0 \ldots b_u) = \mathbf{H}(\mathbf{ev}(\boxed{1}, b_0 \ldots b_u) \| \cdots \| \mathbf{ev}(\boxed{\kappa}, b_0 \ldots b_u))$

Hints

3. can compute $F(p_0)$,
if p_0 is a critical prefix of A
4. locally compare with hints



or



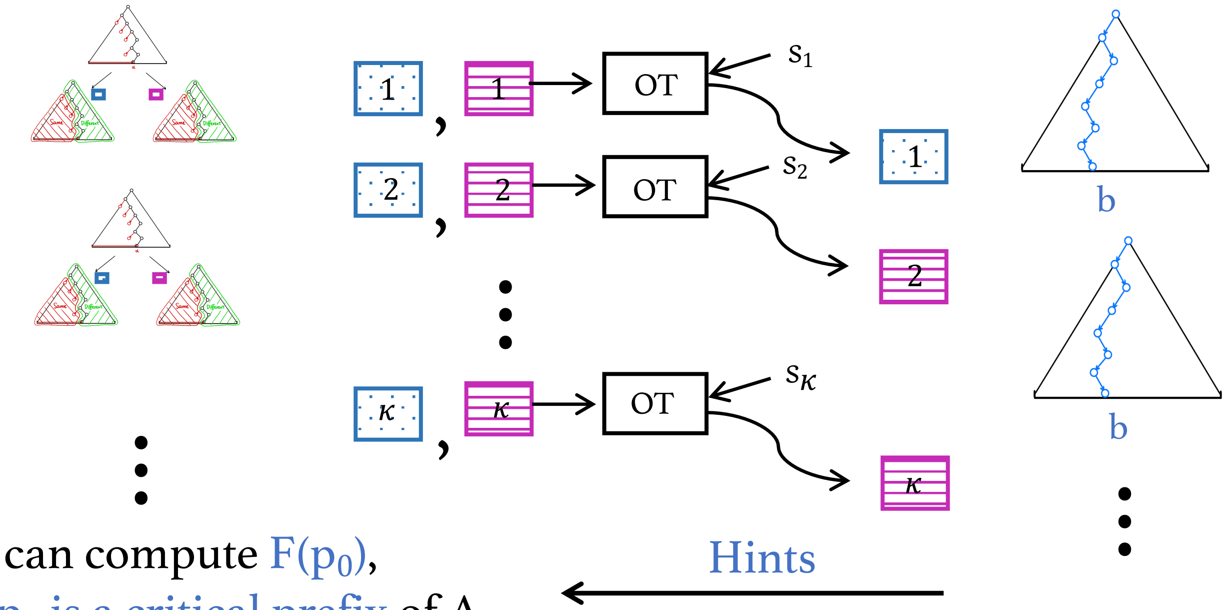
One-Sided Interval Single Point sa-PSI

input A = $[0, \alpha)$

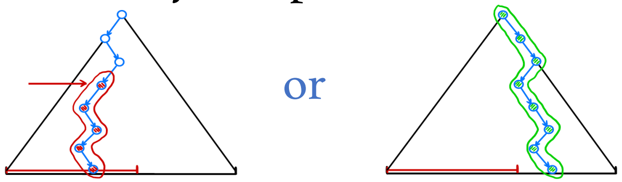
1. generates κ incremental bFSS shares

input B = {b}, b = $b_0 \ldots b_u$

2. picks κ choice bits to learn or



3. can compute $F(p_0)$,
if p_0 is a critical prefix of A
4. locally compare with hints



[see paper] how does Alice efficiently find 'b', if matching prefix is found?

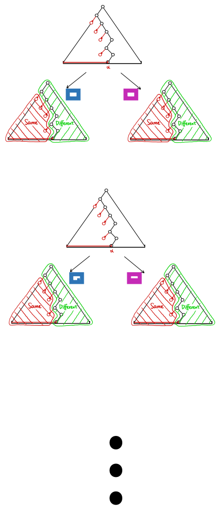
One-Sided Interval Single Point sa-PSI

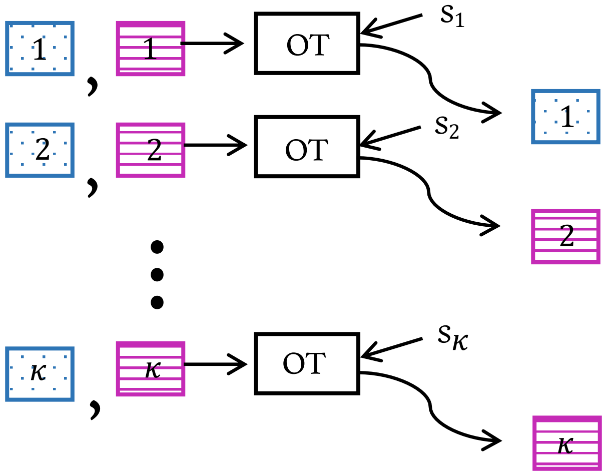
input A = $[0, \alpha)$

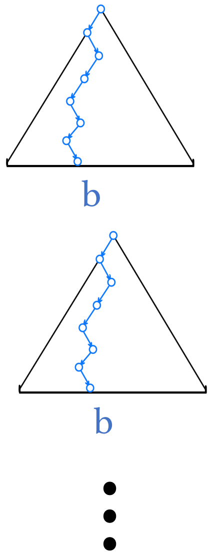
1. generates κ incremental bFSS shares

input B = {b}, b = $b_0 \ldots b_u$

2. picks κ choice bits to learn or







Cost comparison with [GRS22]

- Alice's computation reduces from $O(u \cdot \delta \cdot \kappa)$ to $O(u + u \cdot \kappa)$
- Bob's communication increases from $O(h_{out})$ to $O(u \cdot h_{out})$ bits

, where δ = interval length, κ = security parameter, h_{out} = hash output length

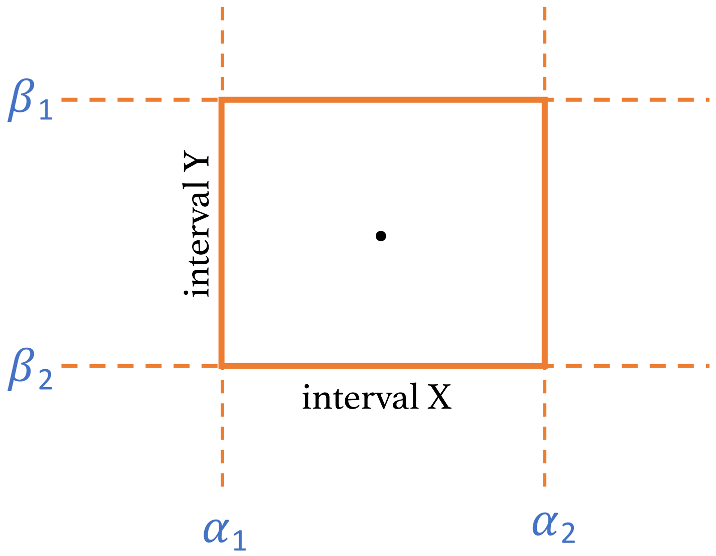
3. can compute $F(p_0)$, if p_0 is a critical prefix of A
4. locally compare with hints

Hints

In the paper, we also show…

ibFSS for d-dimensional interval

input A



L_∞ ball (of diameter δ)
intersection of 2.d intervals

[see paper]

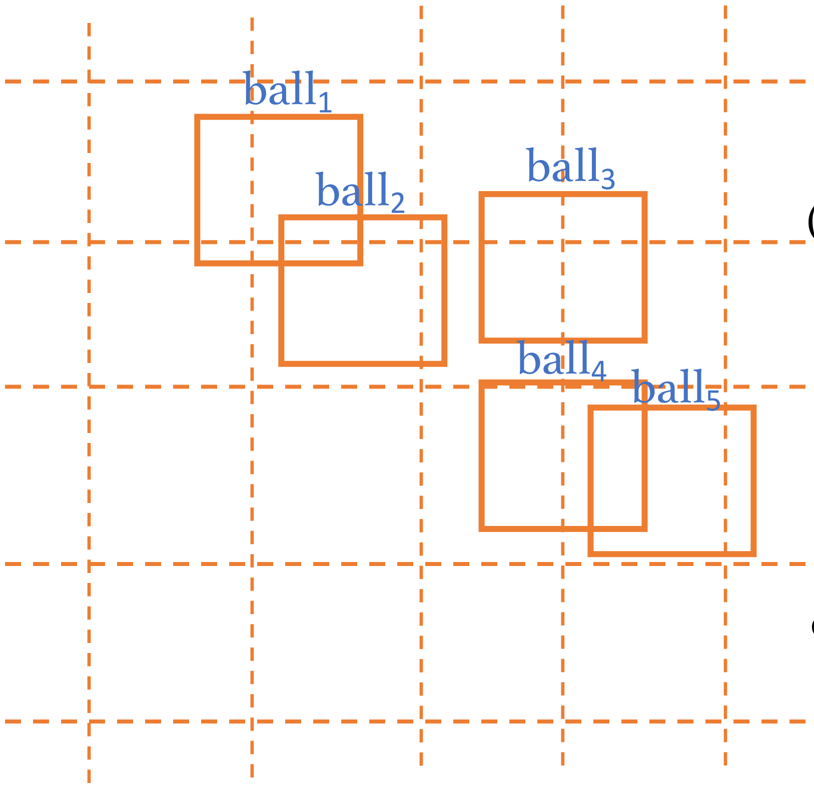
ibFSS scheme for Alice's structure

AND observation: [GRS22, GRS23]
$(b_0 > \alpha_1) \wedge (b_0 < \alpha_2) \wedge (b_1 > \beta_1) \wedge (b_1 < \alpha_1)$?

one-sided interval

Multi-ball Multi-point sa-PSI

union of L_∞ balls



[see paper]

ibFSS scheme for Alice's structure

OR observation: [our contribution]

$(b \in \text{ball}_1) \vee (b \in \text{ball}_2) \vee (b \in \text{ball}_3) \vee (b \in \text{ball}_4) \vee (b \in \text{ball}_5)$

d-dimensional interval

Spatial Hashing: [GRS22, GRS23]

can handle overlapping structures, improve concrete efficiency

Extending Functionality

How can Bob (with unstructured input) learn the intersection?

[see paper]

How can Alice (or Bob) learn PSI-Cardinality or PSI-Sum?

[see paper]

Future Directions

- Can we construct ibFSS for other structures (motivated by real world applications)?
- Can we extend our techniques to other distance metrics like L2 norm, Hamming distance metrics?
- Can we extend our ideas to the malicious setting? Can we improve the existing malicious framework?

Questions?

BACKUP slides

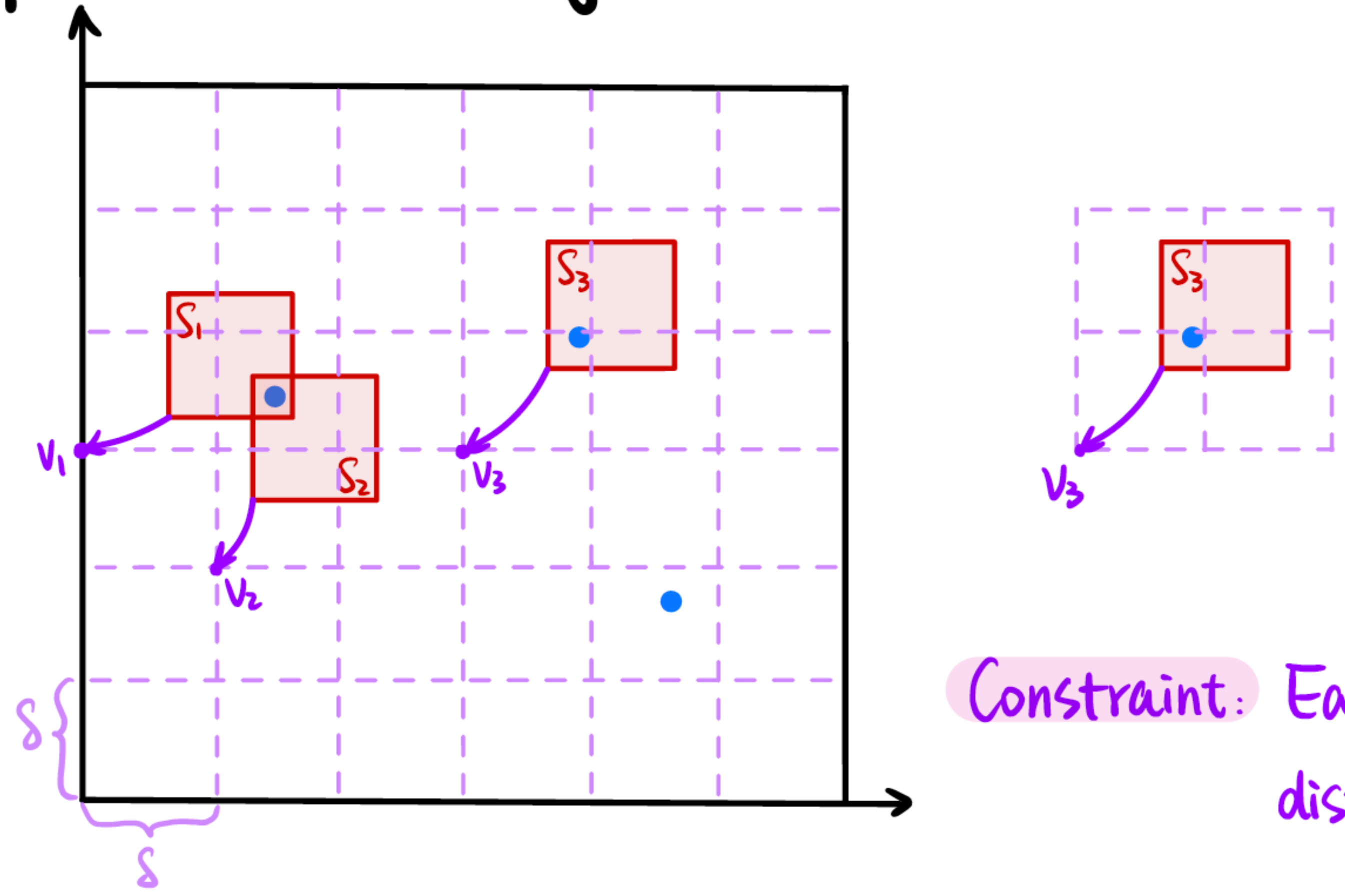
Comp. & Comm. Costs			[GRS22]	[GRS23]	Ours
Comp.	Alice	FSS	$\mathcal{O}(N_A \cdot (4\log\delta)^d \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(N_A \cdot u \cdot d \cdot (\log\delta)^d \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(N_A \cdot \log\delta \cdot d \cdot \ell_{\mathsf{OT}})$
		Intersection	$\mathcal{O}(\lvert S_A\rvert \cdot (2\log\delta)^d \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(\lvert S_A\rvert \cdot u \cdot d \cdot (2\log\delta)^d \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(N_A \cdot (\log\delta)^d + N_B \cdot \log\delta \cdot d \cdot \ell_{\mathsf{OT}})$
	Bob's Eval		$\mathcal{O}(N_B \cdot (2\log\delta)^d \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(N_B \cdot u \cdot d \cdot (2\log\delta)^d \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(N_B \cdot 2^d \cdot ((\log\delta)^d + \log\delta \cdot d \cdot \ell_{\mathsf{OT}})$
Comm. (bits)	OT		$\mathcal{O}(\kappa \cdot N_A \cdot (4\log\delta)^d \cdot \ell_{\mathsf{OT}})$	$O(\kappa \cdot N_A \cdot u \cdot d \cdot (\log\delta)^d \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(\kappa \cdot N_A \cdot \log\delta \cdot d \cdot \ell_{\mathsf{OT}})$
	Bob's Hashes		$\mathcal{O}(N_B \cdot h_{\mathsf{out}})$	$\mathcal{O}(N_B \cdot h_{\mathsf{out}})$	$\mathcal{O}(N_B \cdot (2\log\delta)^d \cdot h_{\mathsf{out}})$

Table 3: Summary of computation and communication costs for spatial hashing. $h_{\mathsf{out}} = \mathcal{O}(\lambda)$ is the output length of the final hash function. $\ell_{\mathsf{OT}} = \mathcal{O}(\kappa)$ is the number of OTs. $|S_A|$ is upper bounded by $\boldsymbol{2^{u\cdot d}}$. Prior work [GRS22, GRS23] only allows Alice to hold ***disjoint*** balls with ***fixed diameter*** δ while our protocol also supports ***overlapping*** balls with ***different diameters***.

Comp. & Comm. Costs			$d=2$		$d=3$		$d=5$	
			[GRS23]	Ours	[GRS23]	Ours	[GRS23]	Ours
Comp. (PRGs, SHA256)	Alice	FSS	86M	840K	516M	1.3M	13.8B	2.1M
		Intersection	352B	2.8B	135T	4.2B	1478Q	7.0B
	Bob's Eval		1.2T	11.3B	13.8T	34.6B	1468T	324B
Comm.	OT		1.3GB	12.5MB	7.7GB	18.8MB	205GB	31.3MB
	Bob's Hashes		4.77MB	477MB	4.77MB	4.77GB	4.77MB	477GB

Table 4: Suppose Alice has $N_A = 300$ balls with fixed diameter $\delta = 32$ and Bob has a collection of $N_B = 10^6$ points in $d = \{2, 3, 5\}$ dimensional space over $\mathcal{U} = \{0,1\}^u, u = 32$. Let computational security parameter $\kappa = 128$ and statistical security parameter $\lambda = 40$. We estimate the computation and communication cost of our new, spatial hashing technique and compare against the previous best construction [GRS23]. Our unit of computation cost is one PRG or hash operation. K, M, B, T, Q stand for thousand, million, billion, trillion, quadrillion respectively in computation units. For example, 1K means 1000 AES/SHA256 calls.

Spatial Hashing

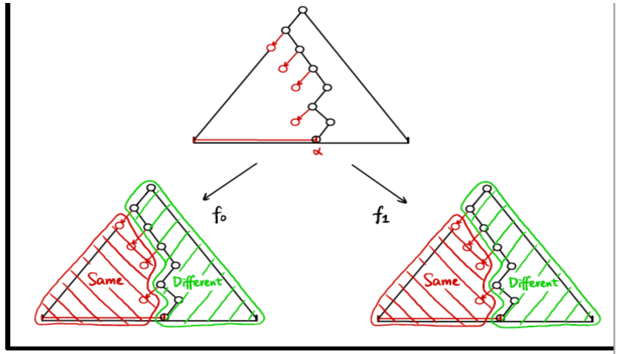


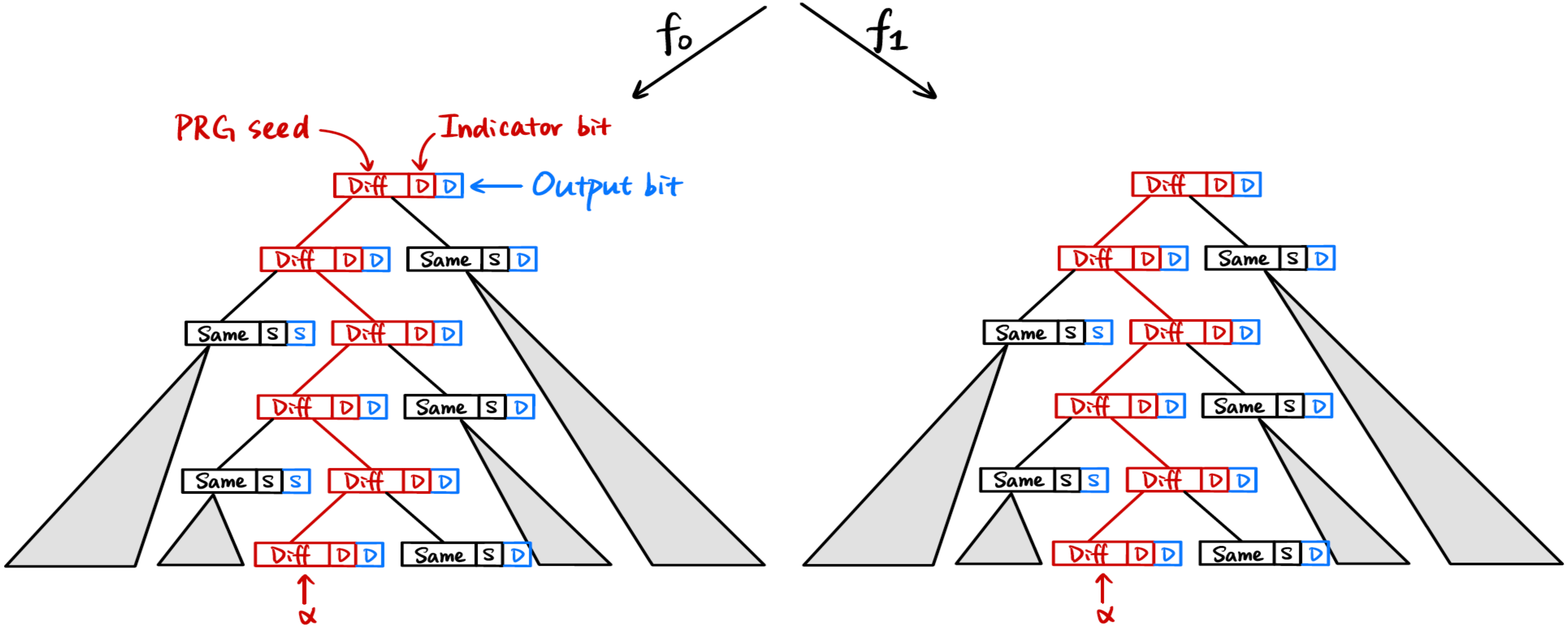
Constraint: Each ball can be assigned to a distinct origin/mini-universe

Incremental FSS

Incremental Distributed Comparison Function (iDCF)

$$f_\alpha(x) = \begin{cases} 0 & \text{if } x < \alpha \\ 1 & \text{otherwise} \end{cases}$$

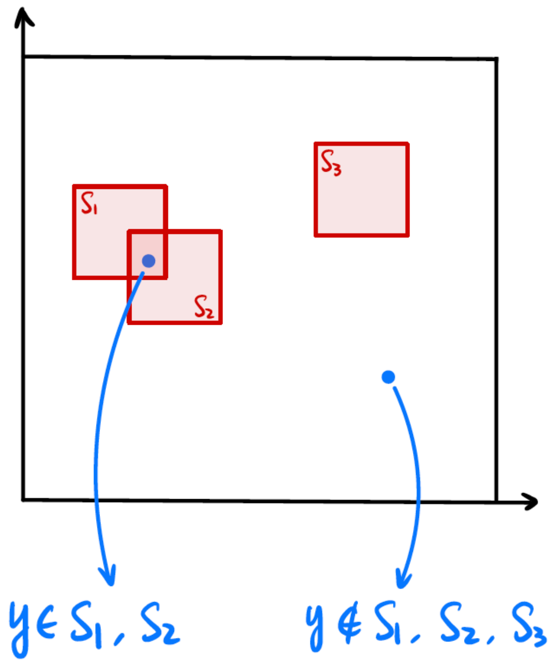




OR Observation

Alice

Bob



Input: $S = S_1 \cup S_2 \cup S_3$

Input: unstructed point y

Output: $S \cap \{y\}$

$y \in S_1$?

$y \in S_2$?

$y \in S_3$?

Sa-PSI-CA* $\Longrightarrow$ PSI-CA

*Only for disjoint balls

Alice | **Bob**

{Critical Prefixes} $\rightarrow$ PSI-CA $\leftarrow$ {All Prefixes}

Duplicates

Related work for sa-PSI

- Hamming distance – [FNP04, CH08, YSPW10] – polynomial evaluate using AHE
- Fuzzy PSI for hamming and l2 metric [IW06] – generic MPC, decryption circuit for homomorphic encryption
- Fuzzy PSI using homomorphic encryption[BCRT16]
- Chakroborti et al – [CFR21] fuzzy psi for 1-d integers and hamming distance, [UCK+21] uses FHE

Single ball vs single point

Comp. & Comm. Costs			[38]	Ours
Comp.	Alice	FSS	$\mathcal{O}(u \cdot d \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(u \cdot d \cdot \ell_{\mathsf{OT}})$
		Intersection	$\mathcal{O}(\min(\lvert \boldsymbol{S_A} \rvert \cdot u, 2^u) \cdot d \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(\boldsymbol{u^d} + u \cdot d \cdot \ell_{\mathsf{OT}})$
	Bob's Eval		$\mathcal{O}(u \cdot d \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(\boldsymbol{u^d} + u \cdot d \cdot \ell_{\mathsf{OT}})$
Comm. (bits)	OT		$\mathcal{O}(\kappa \cdot u \cdot d \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(\kappa \cdot u \cdot d \cdot \ell_{\mathsf{OT}})$
	Bob's Hashes		$\mathcal{O}(h_{\mathsf{out}})$	$\mathcal{O}(\boldsymbol{u^d} \cdot h_{\mathsf{out}})$

Table 1: Summary of computation and communication costs for the setting where Alice holds a single ℓ_∞ ball in the d-dimensional universe $(\{0,1\}^u)^d$ and Bob holds a single point in the universe. $h_{\mathsf{out}} = \mathcal{O}(\lambda)$ is the output length of the final hash function. $\ell_{\mathsf{OT}} = \mathcal{O}(\kappa)$ is the number of OTs. $\lvert S_A \rvert$ is upper bounded by $\boldsymbol{2^{u \cdot d}}$.

Multi ball vs multi point

Comp. & Comm. Costs			[38, 39]	Ours
Comp.	Alice	FSS	$\mathcal{O}(N_A \cdot \boldsymbol{u^d} \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(N_A \cdot \boldsymbol{u} \cdot \boldsymbol{d} \cdot \ell_{\mathsf{OT}})$
		Intersection	$\mathcal{O}(\lvert \boldsymbol{S_A} \rvert \cdot N_A \cdot \boldsymbol{u^d} \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(N_A \cdot \boldsymbol{u^d} + N_B \cdot \boldsymbol{u} \cdot \boldsymbol{d} \cdot \ell_{\mathsf{OT}})$
	Bob's Eval		$\mathcal{O}(N_A \cdot N_B \cdot \boldsymbol{u^d} \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(N_A \cdot N_B \cdot (\boldsymbol{u^d} + \boldsymbol{u} \cdot \boldsymbol{d} \cdot \ell_{\mathsf{OT}}))$
Comm. (bits)	OT		$\mathcal{O}(\kappa \cdot N_A \cdot \boldsymbol{u^d} \cdot \ell_{\mathsf{OT}})$	$\mathcal{O}(\kappa \cdot N_A \cdot \boldsymbol{u} \cdot \boldsymbol{d} \cdot \ell_{\mathsf{OT}})$
	Bob's Hashes		$\mathcal{O}(N_B \cdot h_{\mathsf{out}})$	$\mathcal{O}(N_B \cdot \boldsymbol{N_A} \cdot \boldsymbol{u^d} \cdot h_{\mathsf{out}})$

Table 2: Summary of computation and communication costs for the setting where Alice holds N_A number of ℓ_∞ balls in the d-dimensional universe $(\{0,1\}^u)^d$ and Bob holds N_B points in the universe. $h_{\mathsf{out}} = \mathcal{O}(\lambda)$ is the output length of the final hash function. $\ell_{\mathsf{OT}} = \mathcal{O}(\kappa)$ is the number of OTs. $|S_A|$ is upper bounded by $\boldsymbol{2^{u \cdot d}}$. Prior work [38, 39] only allows Alice to hold ***disjoint*** balls while our protocol also supports ***overlapping*** balls.