PSI from PaXoS:
Fast, Malicious Private Set Intersection

ia.cr/2020/193

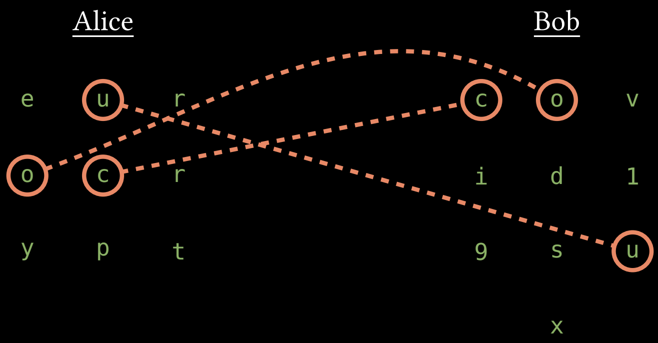
Benny Pinkas — Bar-Ilan University
Mike Rosulek — Oregon State University
Ni Trieu — UC Berkeley
Avishay Yanai — VMware

Eurocrypt 2020, COVID-19 edition

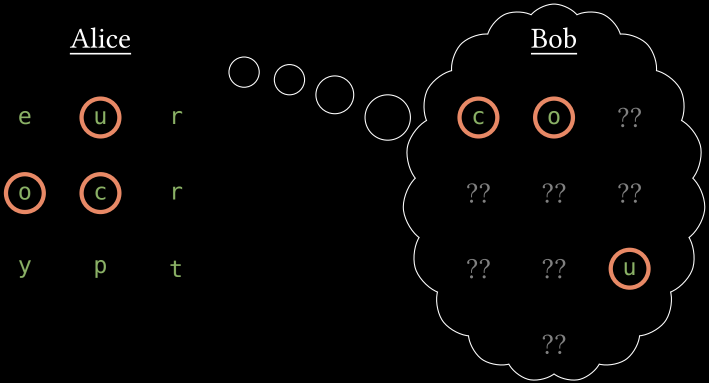
what is private set intersection (PSI)?

Alice				Bob		
e	u	r		c	o	v
o	c	r		i	d	1
y	p	t		9	s	u
					x	

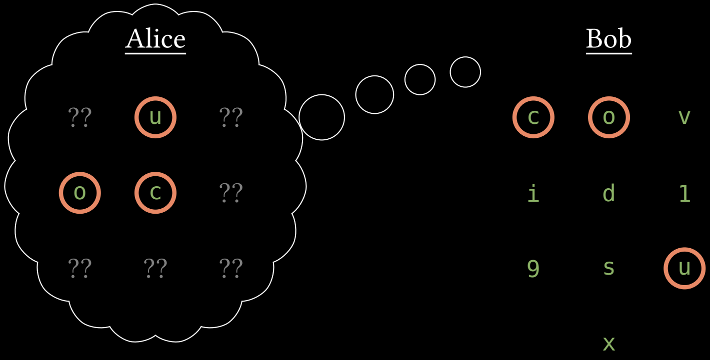
what is private set intersection (PSI)?



what is private set intersection (PSI)?



what is private set intersection (PSI)?



what is private set intersection (PSI)?

Signal

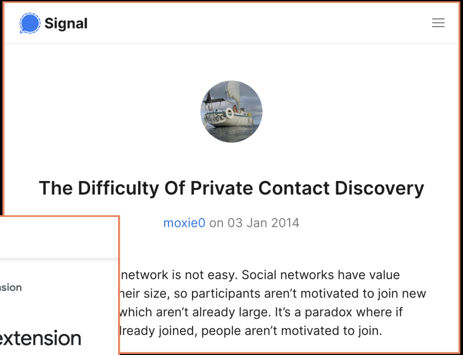
The Difficulty Of Private Contact Discovery

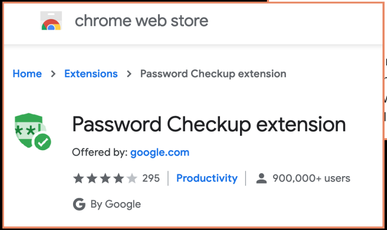
moxie0 on 03 Jan 2014

Building a social network is not easy. Social networks have value proportional to their size, so participants aren't motivated to join new social networks which aren't already large. It's a paradox where if people haven't already joined, people aren't motivated to join.

{my phone contacts} ∩ {users of your service}

what is private set intersection (PSI)?





{my passwords} ∩ {passwords found in breaches}

what is private set intersection (PSI)?

Signal

The Difficulty Of Private Contact Discovery

moxie0 on 03 Jan 2014

network is not easy. Social networks have value
heir size, so participants aren't motivated to join new
which aren't already large. It's a paradox where if
lready joined, people aren't motivated to join.

WIRED

BUSINESS CULTURE GEAR IDEAS SCIENCE SECURITY TRANSPORTATION SIGN IN SUBSCRIBE

LILY HAY NEWMAN SECURITY 06.19.2019 09:00 AM

Google Turns to Retro Cryptography to Keep Data Sets Private

Google's Private Join and Compute will let companies compare notes without divulging sensitive information.

chrome web store

Home > Extensions > Password Checkup extension

Password Checkup extension

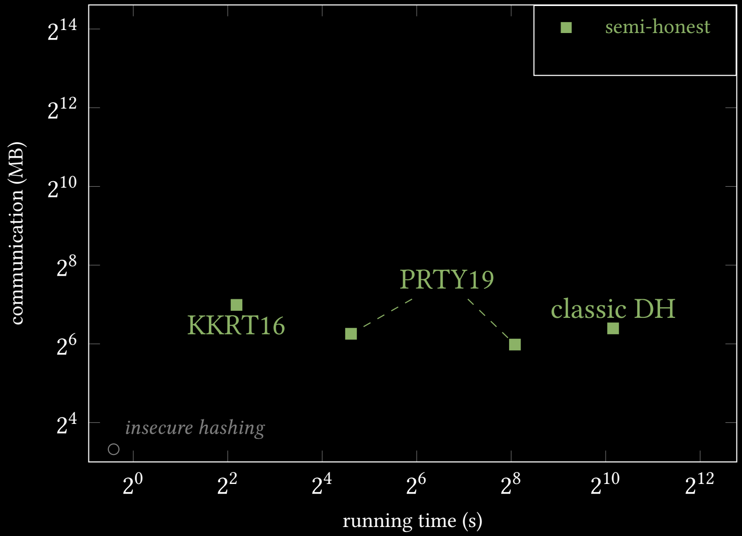
Offered by: google.com

★★★★★ 295 | Productivity | 900,000+ users

By Google

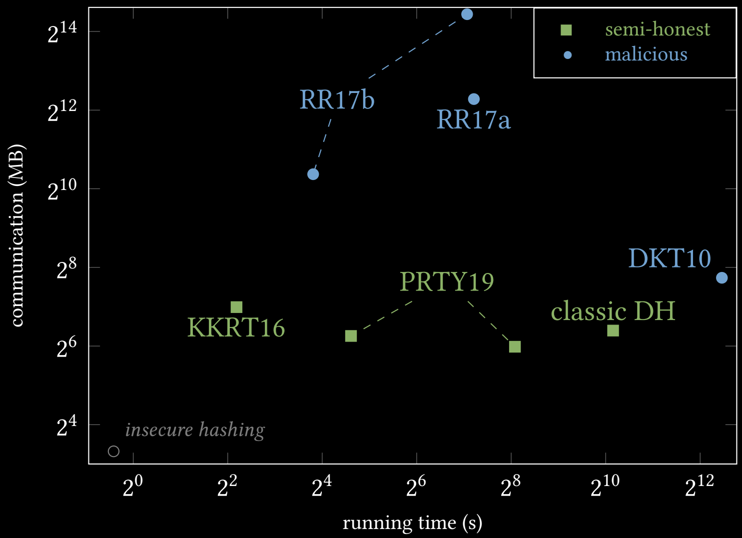
{people who saw ad} ∩ {customers who made purchases}

state of the art: PSI for 1 million items:



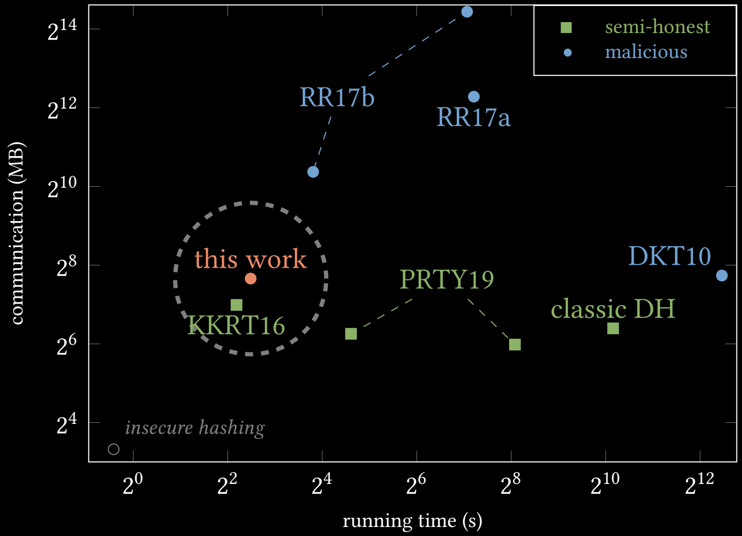
DKT10 = ia.cr/2010/469
KKRT16 = ia.cr/2016/799
RR17a = ia.cr/2016/746
RR17b = ia.cr/2017/769
PRTY19 = ia.cr/2019/634

state of the art: PSI for 1 million items:



DKT10 = ia.cr/2010/469
KKRT16 = ia.cr/2016/799
RR17a = ia.cr/2016/746
RR17b = ia.cr/2017/769
PRTY19 = ia.cr/2019/634

state of the art: PSI for 1 million items:



vs prior malicious:

- ~3× faster
- 8× less comm

vs semi-honest:

- 25% slower
- 60–150% more comm

asymptotically:

- first $O(n)$ malicious from OT extension

DKT10 = ia.cr/2010/469
KKRT16 = ia.cr/2016/799
RR17a = ia.cr/2016/746
RR17b = ia.cr/2017/769
PRTY19 = ia.cr/2019/634

1. *why is existing semi-honest PSI so efficient?*
2. *why is malicious security harder?*
3. *how do we overcome this limitation?*

1. *why is existing semi-honest PSI so efficient?*

2. *why is malicious security harder?*

3. *how do we overcome this limitation?*

what does "PaXoS" mean?

batch oblivious PRF (OPRF)

Alice

Bob

1

2

3

4

5

6

7

8

9

⋮

batch oblivious PRF (OPRF)

Alice

Bob

x_1	1
x_2	2
x_3	3
x_4	4
x_5	5
x_6	6
x_7	7
x_8	8
x_9	9
	$\vdots$

batch oblivious PRF (OPRF)

Alice		Bob
$F_1(x_1)$	1	$F_1(\cdot)$
$F_2(x_2)$	2	$F_2(\cdot)$
$F_3(x_3)$	3	$F_3(\cdot)$
$F_4(x_4)$	4	$F_4(\cdot)$
$F_5(x_5)$	5	$F_5(\cdot)$
$F_6(x_6)$	6	$F_6(\cdot)$
$F_7(x_7)$	7	$F_7(\cdot)$
$F_8(x_8)$	8	$F_8(\cdot)$
$F_9(x_9)$	9	$F_9(\cdot)$
	$\vdots$	

batch oblivious PRF (OPRF)

<u>Alice</u>

<u>Bob</u>

$F_1(x_1)$	1	$F_1(\cdot)$
$F_2(x_2)$	2	$F_2(\cdot)$
$F_3(x_3)$	3	$F_3(\cdot)$
$F_4(x_4)$	4	$F_4(\cdot)$
$F_5(x_5)$	5	$F_5(\cdot)$
$F_6(x_6)$	6	$F_6(\cdot)$
$F_7(x_7)$	7	$F_7(\cdot)$
$F_8(x_8)$	8	$F_8(\cdot)$
$F_9(x_9)$	9	$F_9(\cdot)$
	$\vdots$	

learns nothing about x_i's

batch oblivious PRF (OPRF)

Alice				Bob
	$F_1(x_1)$	1	$F_1(\cdot)$	
	$F_2(x_2)$	2	$F_2(\cdot)$	
	$F_3(x_3)$	3	$F_3(\cdot)$	
	$F_4(x_4)$	4	$F_4(\cdot)$	
all other $F_i(x^*)$ look random	$F_5(x_5)$	5	$F_5(\cdot)$	learns nothing about x_i's
	$F_6(x_6)$	6	$F_6(\cdot)$	
	$F_7(x_7)$	7	$F_7(\cdot)$	
	$F_8(x_8)$	8	$F_8(\cdot)$	
	$F_9(x_9)$	9	$F_9(\cdot)$	
		$\vdots$		

batch oblivious PRF (OPRF)

Alice				Bob
	$F_1(x_1)$	1	$F_1(\cdot)$	
	$F_2(x_2)$	2	$F_2(\cdot)$	
	$F_3(x_3)$	3	$F_3(\cdot)$	
	$F_4(x_4)$	4	$F_4(\cdot)$	
all other $F_i(x^*)$ look random	$F_5(x_5)$	5	$F_5(\cdot)$	learns nothing about x_i's
	$F_6(x_6)$	6	$F_6(\cdot)$	
	$F_7(x_7)$	7	$F_7(\cdot)$	
	$F_8(x_8)$	8	$F_8(\cdot)$	
	$F_9(x_9)$	9	$F_9(\cdot)$	
		⋮		

achieved very efficiently from OT extension

the KKRT16 (PSZ14) protocol

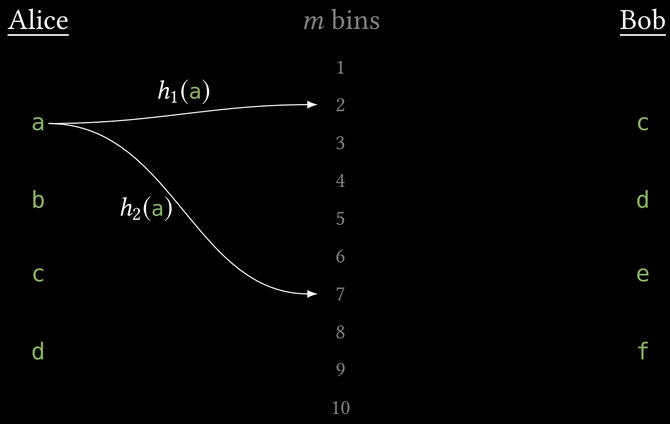
Alice	Bob
a	c
b	d
c	e
d	f

the KKRT16 (PSZ14) protocol

Alice	m bins	Bob
	1	
	2	
a	3	c
	4	
b	5	d
	6	
c	7	e
	8	
d	9	f
	10	

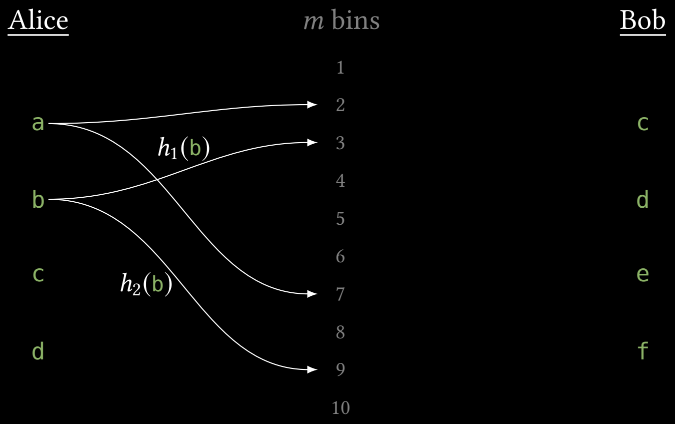
1. Agree on random $h_1, h_2 : \{0,1\}^* \to [m]$

the KKRT16 (PSZ14) protocol



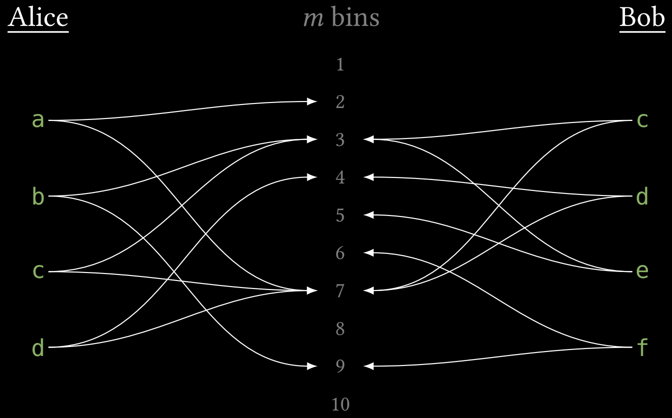
1. Agree on random $h_1, h_2 : \{0,1\}^* \to [m]$

the KKRT16 (PSZ14) protocol



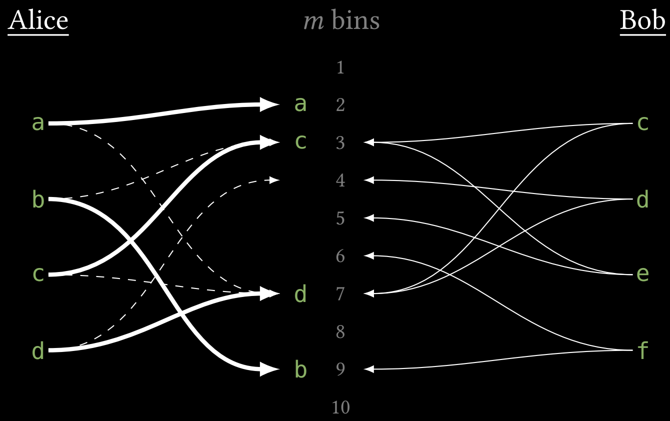
1. Agree on random $h_1, h_2 : \{0,1\}^* \to [m]$

the KKRT16 (PSZ14) protocol



1. Agree on random $h_1, h_2 : \{0,1\}^* \to [m]$

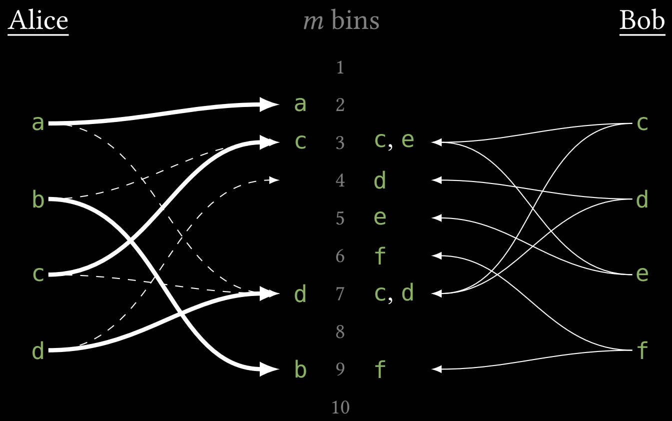
the KKRT16 (PSZ14) protocol



1. Agree on random $h_1, h_2 : \{0,1\}^* \to [m]$

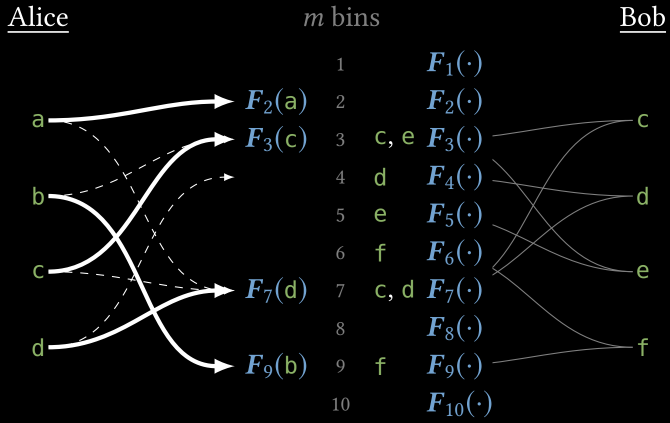
2. Alice places each x into bin $h_1(x)$ or $h_2(x)$

the KKRT16 (PSZ14) protocol



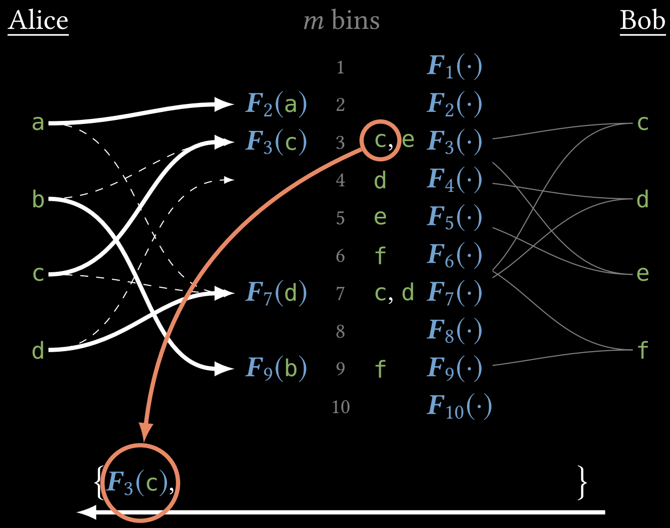
1. Agree on random $h_1, h_2 : \{0,1\}^* \to [m]$
2. Alice places each x into bin $h_1(x)$ or $h_2(x)$
3. Bob places each x into bins $h_1(x)$ and $h_2(x)$

the KKRT16 (PSZ14) protocol



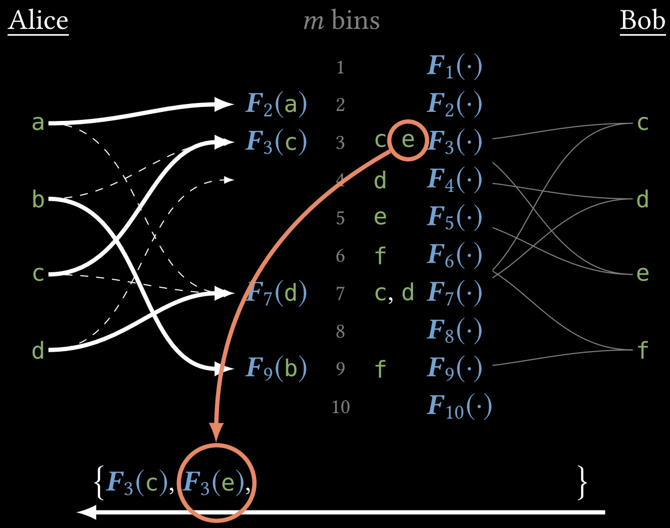
1. Agree on random $h_1, h_2 : \{0,1\}^* \longrightarrow [m]$
2. Alice places each x into bin $h_1(x)$ or $h_2(x)$
3. Bob places each x into bins $h_1(x)$ and $h_2(x)$
4. OPRF in each bin:
   Alice learns one $\boldsymbol{F}_i(x)$;
   Bob learns entire $\boldsymbol{F}_i(\cdot)$

the KKRT16 (PSZ14) protocol



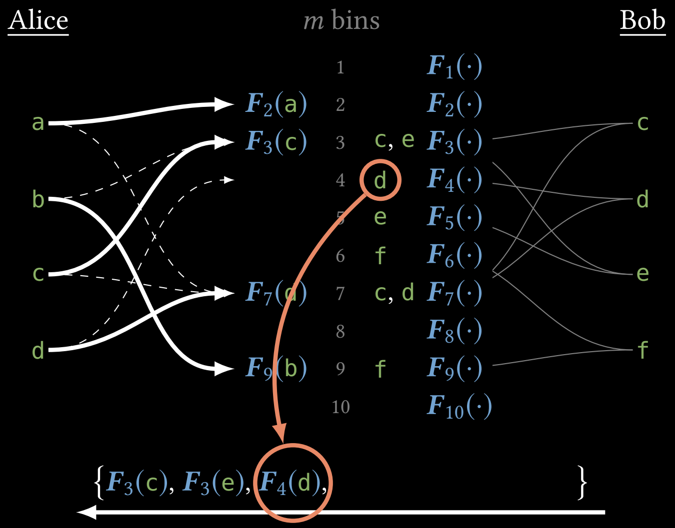
1. Agree on random $h_1, h_2 : \{0,1\}^* \to [m]$
2. Alice places each x into bin $h_1(x)$ or $h_2(x)$
3. Bob places each x into bins $h_1(x)$ and $h_2(x)$
4. OPRF in each bin: Alice learns one $F_i(x)$; Bob learns entire $F_i(\cdot)$
5. Bob sends all $F_i(x)$ values

the KKRT16 (PSZ14) protocol



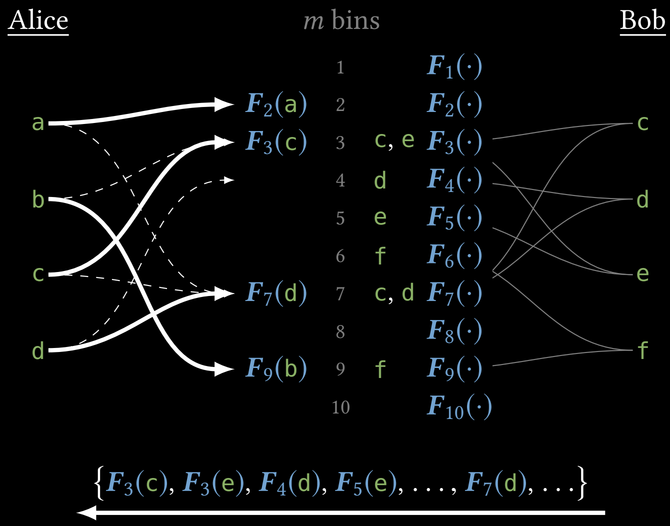
1. Agree on random $h_1, h_2 : \{0,1\}^* \to [m]$
2. Alice places each x into bin $h_1(x)$ or $h_2(x)$
3. Bob places each x into bins $h_1(x)$ and $h_2(x)$
4. OPRF in each bin: Alice learns one $F_i(x)$; Bob learns entire $F_i(\cdot)$
5. Bob sends all $F_i(x)$ values

the KKRT16 (PSZ14) protocol



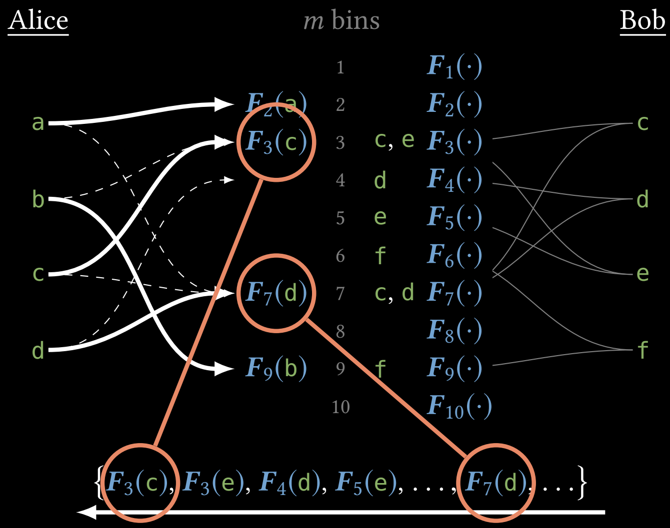
1. Agree on random $h_1, h_2 : \{0,1\}^* \to [m]$
2. Alice places each x into bin $h_1(x)$ or $h_2(x)$
3. Bob places each x into bins $h_1(x)$ and $h_2(x)$
4. OPRF in each bin: Alice learns one $F_i(x)$; Bob learns entire $F_i(\cdot)$
5. Bob sends all $F_i(x)$ values

the KKRT16 (PSZ14) protocol



1. Agree on random $h_1, h_2 : \{0, 1\}^* \rightarrow [m]$
2. Alice places each x into bin $h_1(x)$ or $h_2(x)$
3. Bob places each x into bins $h_1(x)$ and $h_2(x)$
4. OPRF in each bin: Alice learns one $F_i(x)$; Bob learns entire $F_i(\cdot)$
5. Bob sends all $F_i(x)$ values

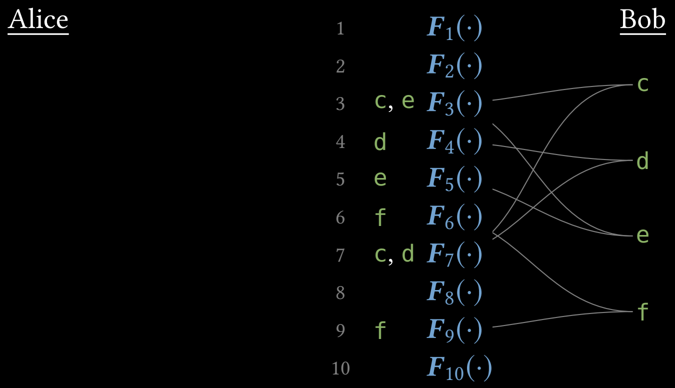
the KKRT16 (PSZ14) protocol



1. Agree on random $h_1, h_2 : \{0,1\}^* \to [m]$
2. Alice places each x into bin $h_1(x)$ or $h_2(x)$
3. Bob places each x into bins $h_1(x)$ and $h_2(x)$
4. OPRF in each bin: Alice learns one $F_i(x)$; Bob learns entire $F_i(\cdot)$
5. Bob sends all $F_i(x)$ values

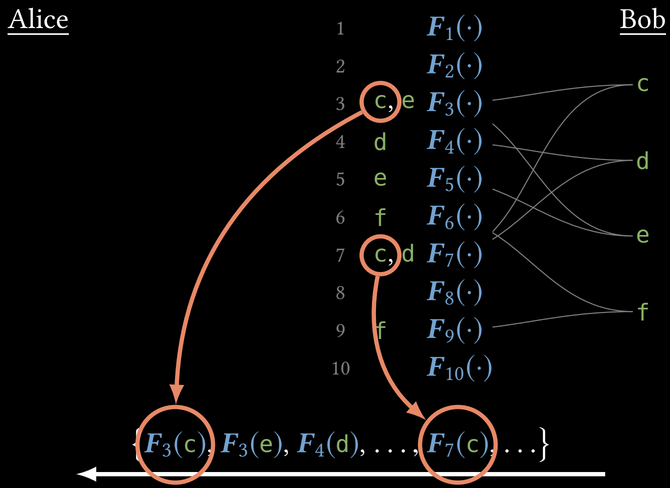
why isn’t it secure against malicious parties?

why isn't it secure against malicious parties?



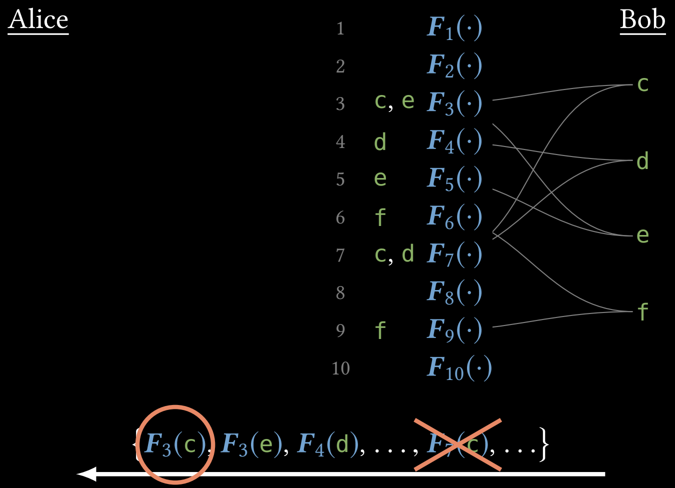
$$\{F_3(\mathrm{c}), F_3(\mathrm{e}), F_4(\mathrm{d}), \ldots, F_7(\mathrm{c}), \ldots\}$$

why isn't it secure against malicious parties?



Bob should send two F-values per item

why isn't it secure against malicious parties?

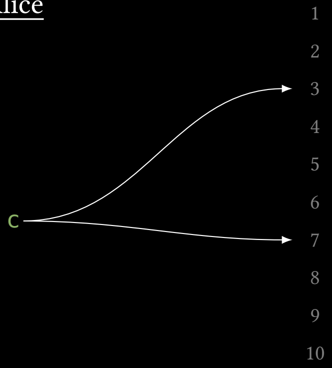


Bob should send two F-values per item , what if he sends only one?

why isn't it secure against malicious parties?

Alice

Bob

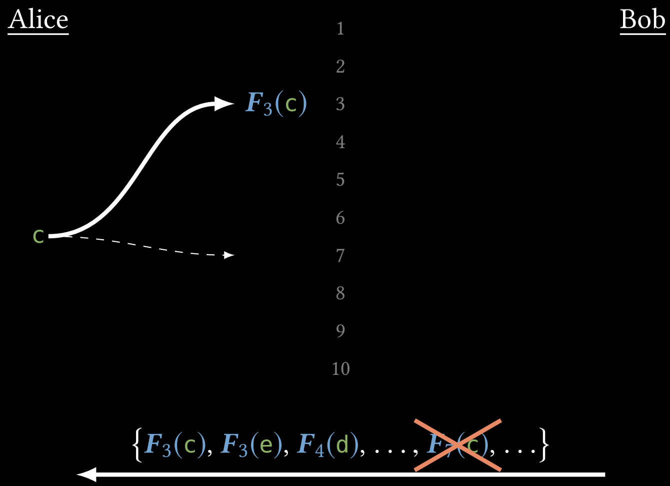


Bob should send two $\boldsymbol{F}$-values per item , what if he sends only one?

Alice has c; does she include it in output?

$$\{F_3(\mathtt{c}), F_3(\mathtt{e}), F_4(\mathtt{d}), \ldots, \cancel{F_7(\mathtt{c})}, \ldots\}$$

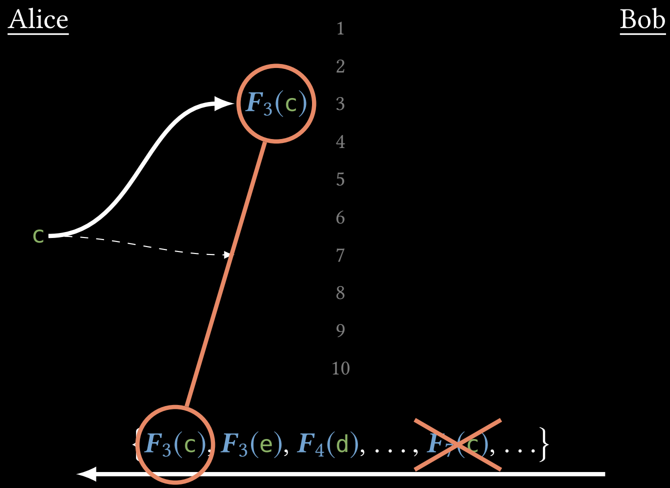
why isn't it secure against malicious parties?



Bob should send two ***F***-values per item , what if he sends only one?

Alice has c; does she include it in output?

why isn't it secure against malicious parties?



Bob should send two F-values per item , what if he sends only one?

Alice has c; does she include it in output?

why isn't it secure against malicious parties?

Alice

Bob

1
2
3
4
5
6
7
8
9
10

c

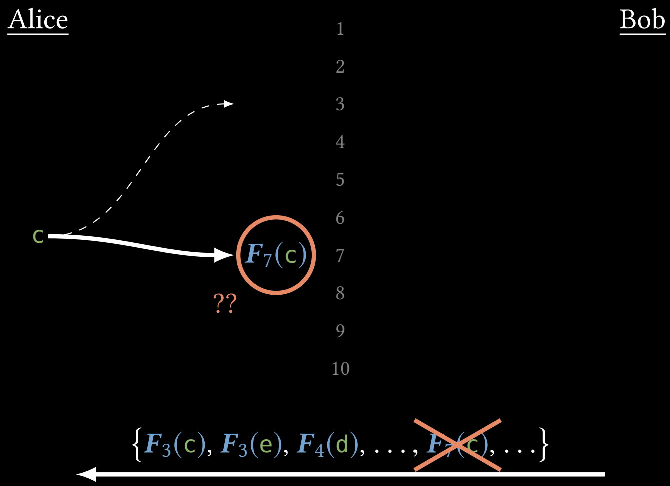
$F_7(c)$

Bob should send two F-values per item , what if he sends only one?

Alice has c; does she include it in output?

$\{F_3(c), F_3(e), F_4(d), \ldots, F_7(c), \ldots\}$

why isn't it secure against malicious parties?

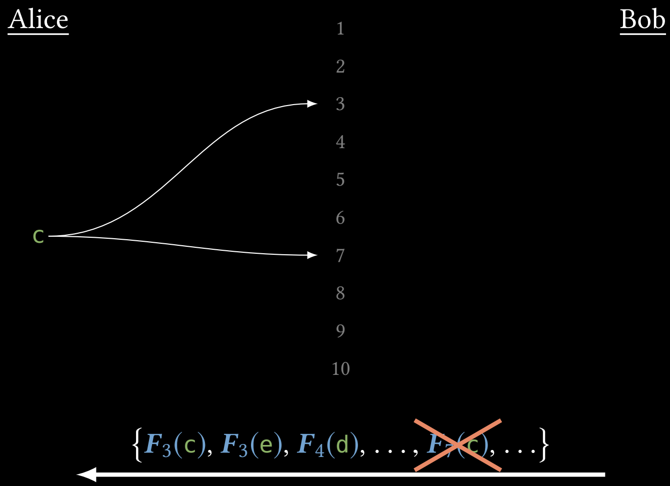


Bob should send two F-values per item , what if he sends only one?

Alice has c; does she include it in output?

Only if c placed in bin 3!

why isn't it secure against malicious parties?



Bob should send two F-values per item , what if he sends only one?

Alice has c; does she include it in output?

Only if c placed in bin 3!

- Depends on Alice's **entire input**!

$\Rightarrow$ can't simulate!

how do we overcome this problem?

batch OPRF for malicious PSI

Alice		Bob
$F_1(x_1)$	1	$F_1(\cdot)$
$F_2(x_2)$	2	$F_2(\cdot)$
$F_3(x_3)$	3	$F_3(\cdot)$
$F_4(x_4)$	4	$F_4(\cdot)$
$F_5(x_5)$	5	$F_5(\cdot)$
$F_6(x_6)$	6	$F_6(\cdot)$
$F_7(x_7)$	7	$F_7(\cdot)$
$F_8(x_8)$	8	$F_8(\cdot)$
$F_9(x_9)$	9	$F_9(\cdot)$
	$\vdots$	

batch OPRF for malicious PSI

Alice		Bob
$F_1(x_1)$	1	$F_1(\cdot)$
$F_2(x_2)$	2	$F_2(\cdot)$
$F_3(x_3)$	3	$F_3(\cdot)$
$F_4(x_4)$	4	$F_4(\cdot)$
$F_5(x_5)$	5	$F_5(\cdot)$
$F_6(x_6)$	6	$F_6(\cdot)$
$F_7(x_7)$	7	$F_7(\cdot)$
$F_8(x_8)$	8	$F_8(\cdot)$
$F_9(x_9)$	9	$F_9(\cdot)$
	$\vdots$	

State of the art malicious batch OPRF [OOS17]

- essentially same cost as semi-honest

batch OPRF for malicious PSI

Alice		Bob
$F_1(x_1)$	1	$F_1(\cdot)$
$F_2(x_2)$	2	$F_2(\cdot)$
$F_3(x_3)$	3	$F_3(\cdot)$
$F_4(x_4)$	4	$F_4(\cdot)$
$F_5(x_5)$	5	$F_5(\cdot)$
$F_6(x_6)$	6	$F_6(\cdot)$
$F_7(x_7)$	7	$F_7(\cdot)$
$F_8(x_8)$	8	$F_8(\cdot)$
$F_9(x_9)$	9	$F_9(\cdot)$
	$\vdots$	

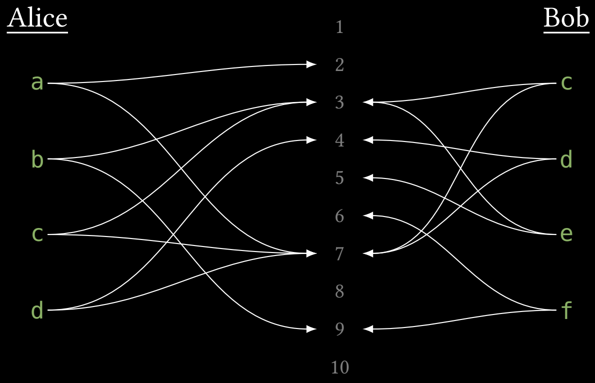
State of the art malicious batch OPRF [OOS17]

- essentially same cost as semi-honest
- consistency check relies on an additive homomorphism:

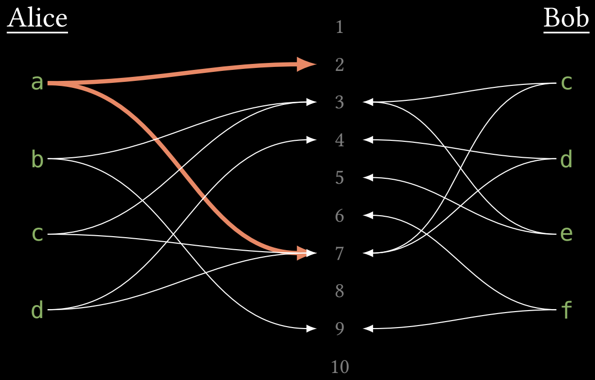
$$F_i(x) \oplus F_j(y) = F_{ij}(x \oplus y)$$

*: *a gross oversimplification*

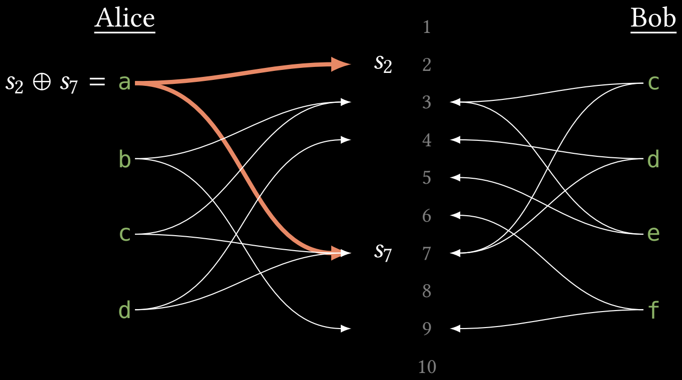
our protocol main idea:



our protocol main idea:

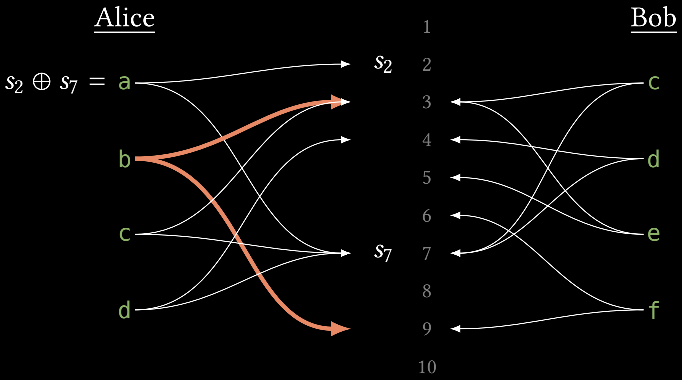


our protocol main idea:



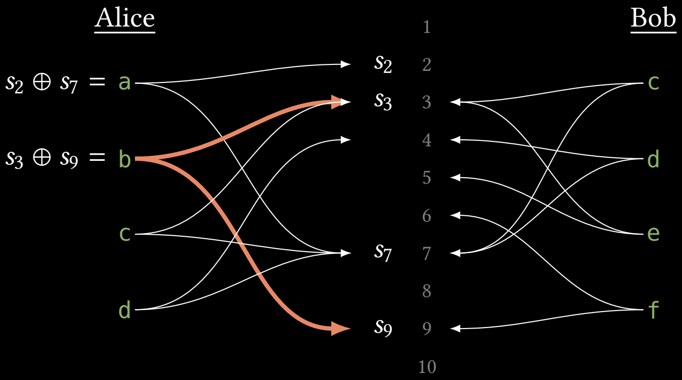
Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

our protocol main idea:



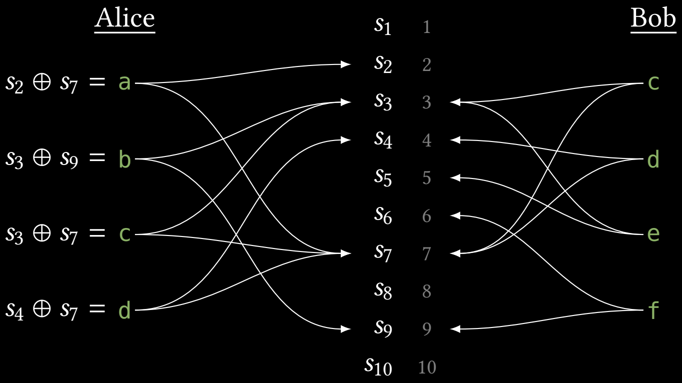
Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

our protocol main idea:



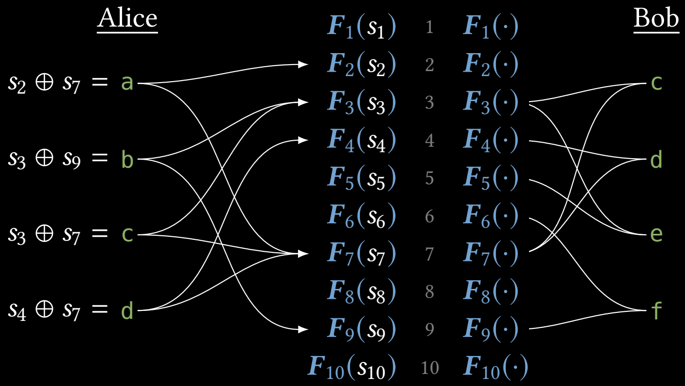
Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

our protocol main idea:



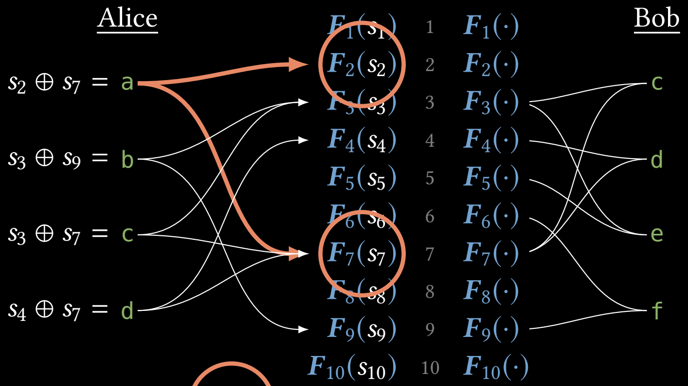
Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

our protocol main idea:



Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

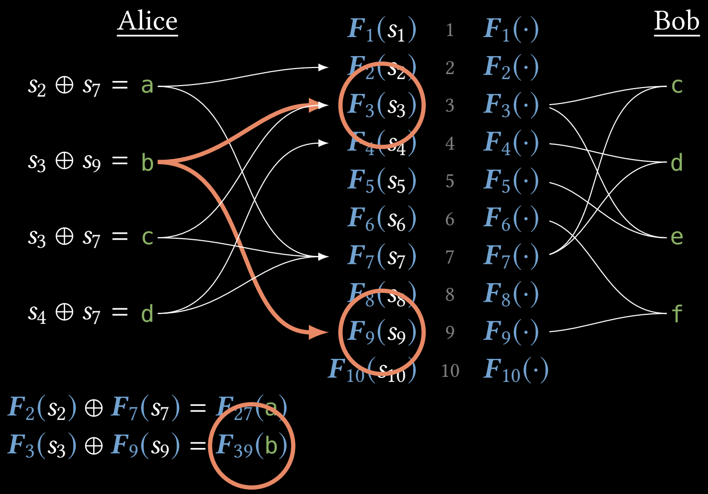
our protocol main idea:



$$F_2(s_2) \oplus F_7(s_7) = F_{27}(a)$$

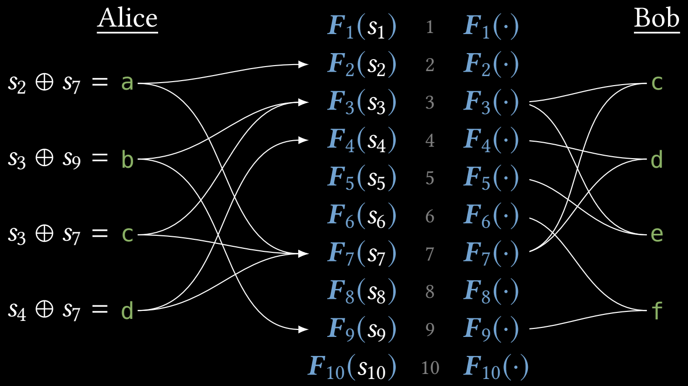
Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

our protocol main idea:



Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

our protocol main idea:



Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

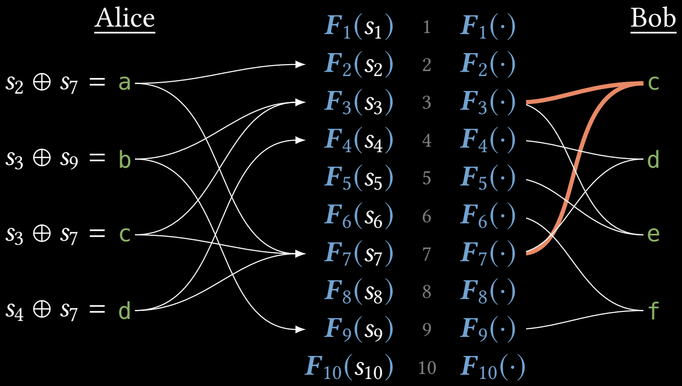
$F_2(s_2) \oplus F_7(s_7) = F_{27}(\mathsf{a})$

$F_3(s_3) \oplus F_9(s_9) = F_{39}(\mathsf{b})$

$F_3(s_3) \oplus F_7(s_7) = F_{37}(\mathsf{c})$

$F_4(s_4) \oplus F_7(s_7) = F_{47}(\mathsf{d})$

our protocol main idea:



$F_2(s_2) \oplus F_7(s_7) = F_{27}(\mathsf{a})$

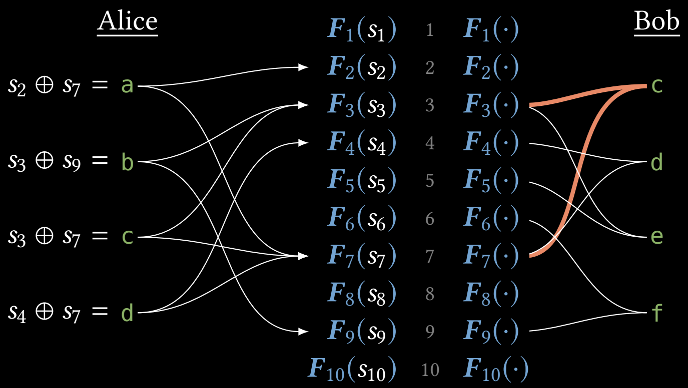
$F_3(s_3) \oplus F_9(s_9) = F_{39}(\mathsf{b})$

$F_3(s_3) \oplus F_7(s_7) = F_{37}(\mathsf{c})$

$F_4(s_4) \oplus F_7(s_7) = F_{47}(\mathsf{d})$

Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

our protocol main idea:

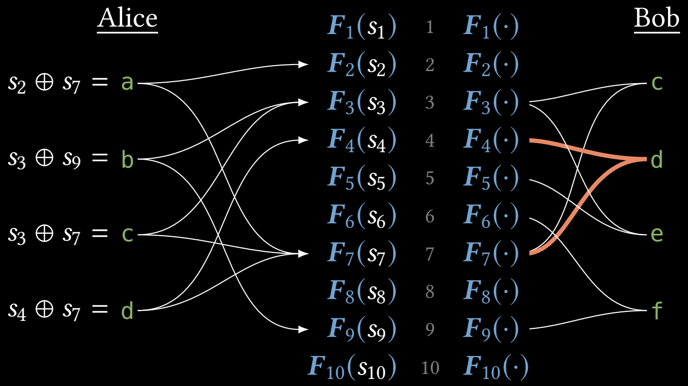


$F_2(s_2) \oplus F_7(s_7) = F_{27}(a)$
$F_3(s_3) \oplus F_9(s_9) = F_{39}(b)$
$F_3(s_3) \oplus F_7(s_7) = F_{37}(c)$
$F_4(s_4) \oplus F_7(s_7) = F_{47}(d)$

$\{F_{37}(c), \qquad \}$

Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

our protocol main idea:

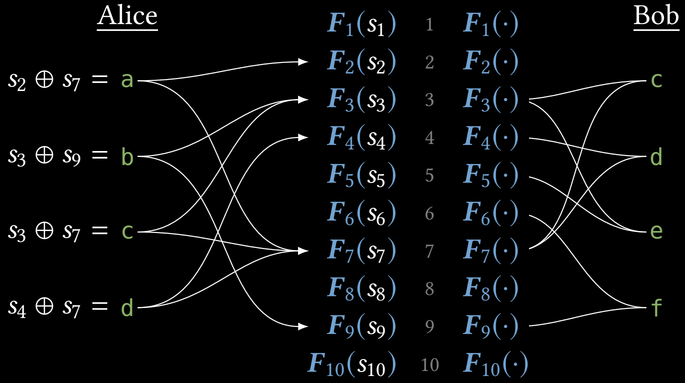


Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

$$F_2(s_2) \oplus F_7(s_7) = F_{27}(a)$$
$$F_3(s_3) \oplus F_9(s_9) = F_{39}(b)$$
$$F_3(s_3) \oplus F_7(s_7) = F_{37}(c)$$
$$F_4(s_4) \oplus F_7(s_7) = F_{47}(d)$$

$$\{F_{37}(c), F_{47}(d), \quad \}$$

our protocol main idea:



Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

$F_2(s_2) \oplus F_7(s_7) = F_{27}(\mathsf{a})$

$F_3(s_3) \oplus F_9(s_9) = F_{39}(\mathsf{b})$

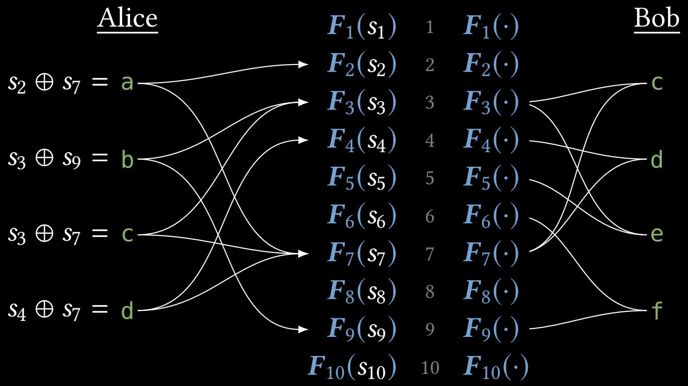
$F_3(s_3) \oplus F_7(s_7) = F_{37}(\mathsf{c})$

$F_4(s_4) \oplus F_7(s_7) = F_{47}(\mathsf{d})$

$\{F_{37}(\mathsf{c}), F_{47}(\mathsf{d}), F_{35}(\mathsf{e}), F_{69}(\mathsf{f})\}$ ←

Bob sends only one F-value per item

our protocol main idea:



$F_2(s_2) \oplus F_7(s_7) = F_{27}(\mathsf{a})$

$F_3(s_3) \oplus F_9(s_9) = F_{39}(\mathsf{b})$

$F_3(s_3) \oplus F_7(s_7) = F_{37}(\mathsf{c})$

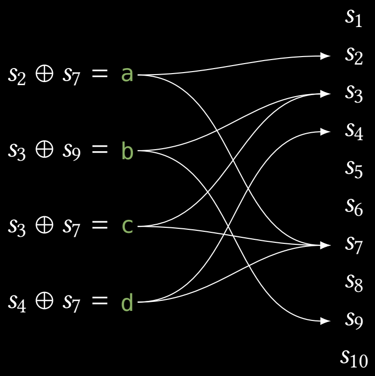
$F_4(s_4) \oplus F_7(s_7) = F_{47}(\mathsf{d})$

$\{F_{37}(\mathsf{c}), F_{47}(\mathsf{d}), F_{35}(\mathsf{e}), F_{69}(\mathsf{f})\}$ ←

Alice secret-shares x into bins $h_1(x)$ and $h_2(x)$

Bob sends only one F-value per item

[how] can Alice secret-share all items?



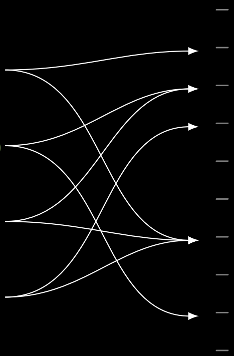
[how] can Alice secret-share all items?

$s_2 \oplus s_7 = \mathtt{a}$

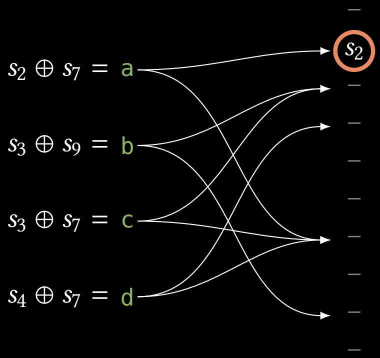
$s_3 \oplus s_9 = \mathtt{b}$

$s_3 \oplus s_7 = \mathtt{c}$

$s_4 \oplus s_7 = \mathtt{d}$



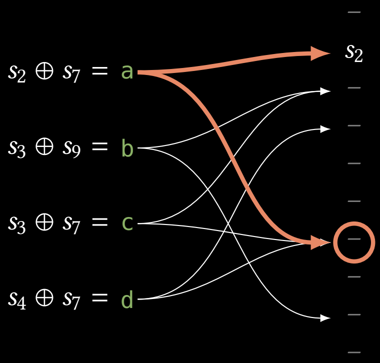
[how] can Alice secret-share all items?



ALGORITHM:

set one location arbitrarily

[how] can Alice secret-share all items?

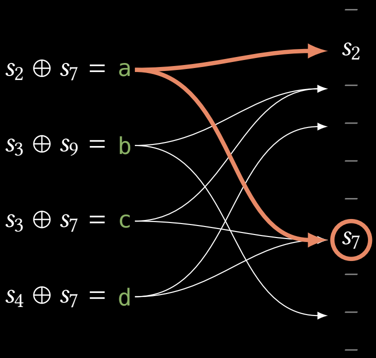


ALGORITHM:

set one location arbitrarily

find item with one unset location

[how] can Alice secret-share all items?

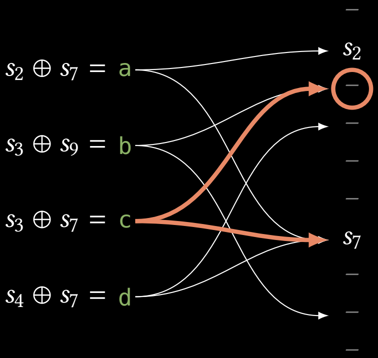


ALGORITHM:

set one location arbitrarily

find item with one unset location
solve for that unset location

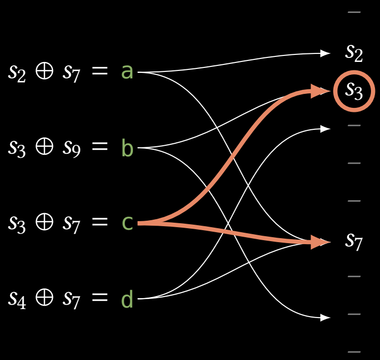
[how] can Alice secret-share all items?



ALGORITHM:

set one location arbitrarily
repeat:
    find item with one unset location
    solve for that unset location

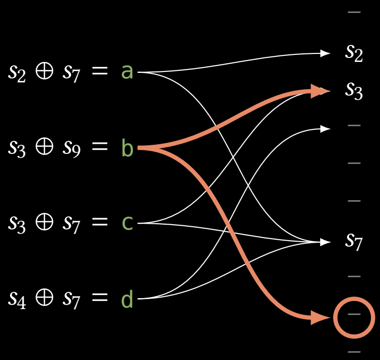
[how] can Alice secret-share all items?



ALGORITHM:

set one location arbitrarily
repeat:
    find item with one unset location
    solve for that unset location

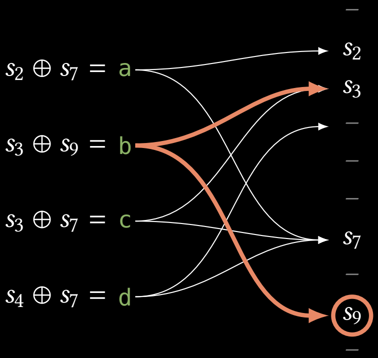
[how] can Alice secret-share all items?



ALGORITHM:

set one location arbitrarily
repeat:
    find item with one unset location
    solve for that unset location

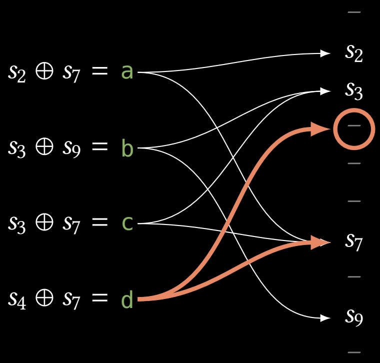
[how] can Alice secret-share all items?



ALGORITHM:

set one location arbitrarily
repeat:
    find item with one unset location
    solve for that unset location

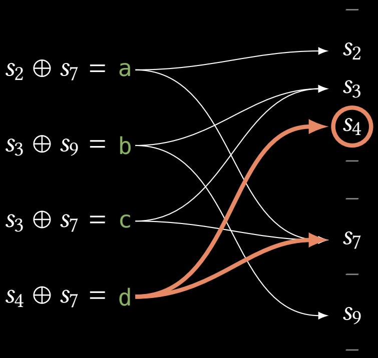
[how] can Alice secret-share all items?



ALGORITHM:

set one location arbitrarily
repeat:
    find item with one unset location
    solve for that unset location

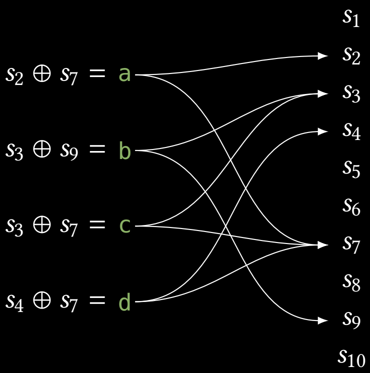
[how] can Alice secret-share all items?



ALGORITHM:

set one location arbitrarily
repeat:
    find item with one unset location
    solve for that unset location

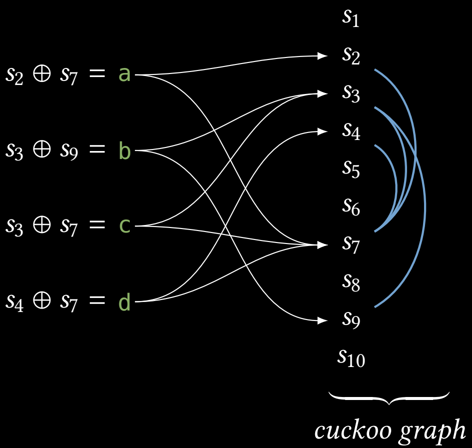
[how] can Alice secret-share all items?



ALGORITHM:

```
set one location arbitrarily
repeat:
    find item with one unset location
    solve for that unset location
```

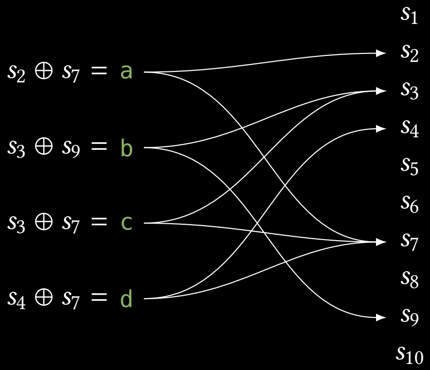
[how] can Alice secret-share all items?

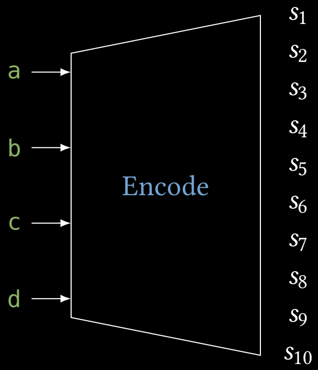


ALGORITHM:

```
set one location arbitrarily
repeat:
    find item with one unset location
    solve for that unset location
```

only works if cuckoo graph acyclic

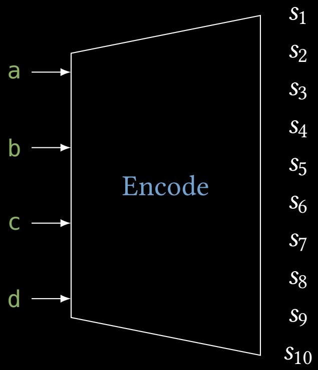

$s_2 \oplus s_7 = a$
$s_3 \oplus s_9 = b$
$s_3 \oplus s_7 = c$
$s_4 \oplus s_7 = d$
s_1
s_2
s_3
s_4
s_5
s_6
s_7
s_8
s_9
s_{10}



encode so that for all x:

$$s_{h_1(x)} \oplus s_{h_2(x)} = x$$

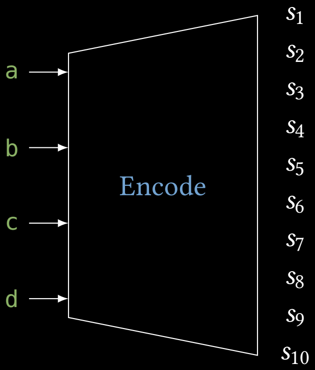
probe-and-XOR-of-strings (PaXoS)



encode so that for all x:

$$\bigoplus_{i \in P(x)} s_i = x$$

probe-and-XOR-of-strings (PaXoS)

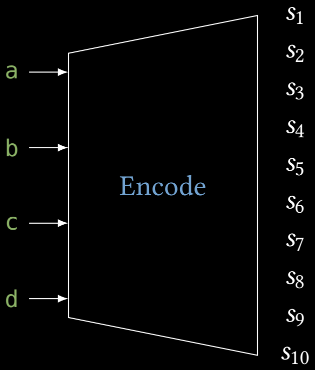


encode so that for all x:

$$\bigoplus_{i \in P(x)} s_i = x$$

1. system of linear constraints must be satisfiable with overwhelming probability

probe-and-XOR-of-strings (PaXoS)

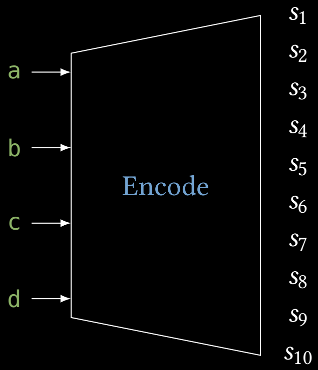


encode so that for all x:

$$\bigoplus_{i \in P(x)} s_i = x$$

1. system of linear constraints must be satisfiable with overwhelming probability
2. $|\vec{s}|$ = number of OPRFs = communication cost of PSI, ideally $O(\#\text{ items})$

probe-and-XOR-of-strings (PaXoS)



encode so that for all x:

$$\bigoplus_{i \in P(x)} s_i = x$$

1. system of linear constraints must be satisfiable with overwhelming probability
2. $|\vec{s}|$ = number of OPRFs = communication cost of PSI, ideally $O(\#\text{ items})$
3. ideally linear-time encoding of items into $\vec{s}$.

PaXoS constructions

secret-shared cuckoo idea:

- requires acyclic cuckoo graph
- failure probability too high

PaXoS constructions

secret-shared cuckoo idea:

- requires acyclic cuckoo graph
- failure probability too high

probe each position with probability 0.5:

- n items $\rightsquigarrow$ vector of size $n + \lambda$
- expensive $O(n^3)$ encoding

PaXoS constructions

secret-shared cuckoo idea:

- requires acyclic cuckoo graph
- failure probability too high

probe each position with probability 0.5:

- n items $\rightsquigarrow$ vector of size $n + \lambda$
- expensive $O(n^3)$ encoding

garbled bloom filter [DCW13]:

- n items $\rightsquigarrow$ vector of size λn

PaXoS constructions

secret-shared cuckoo idea:

- requires acyclic cuckoo graph
- failure probability too high

probe each position with probability 0.5:

- n items $\rightsquigarrow$ vector of size $n + \lambda$
- expensive $O(n^3)$ encoding

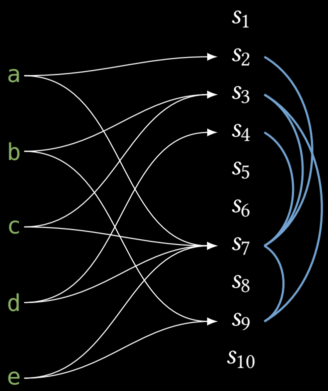
garbled bloom filter [DCW13]:

- n items $\rightsquigarrow$ vector of size λn

new garbled cuckoo PaXoS:

- n items $\rightsquigarrow$ vector of size $\sim 2.4n$
- fast encoding: $O(n\lambda)$

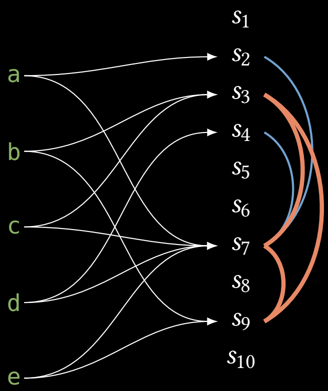
new garbled cuckoo PaXoS



for each item x:

- probe positions $h_1(x)$ and $h_2(x)$

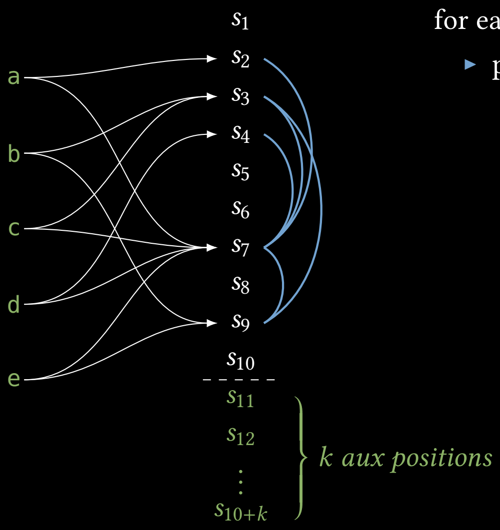
new garbled cuckoo PaXoS



for each item x:

- probe positions $h_1(x)$ and $h_2(x)$

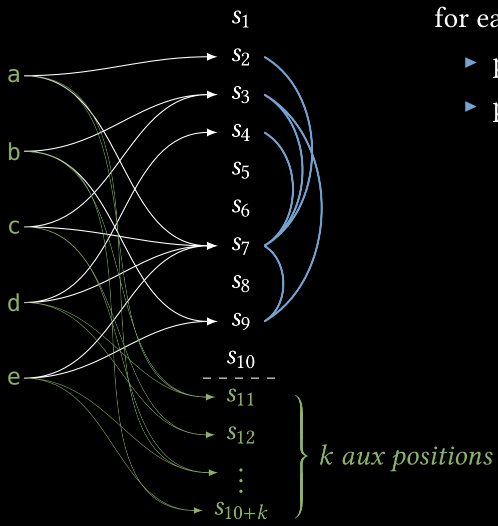
new garbled cuckoo PaXoS



for each item x:

- probe positions $h_1(x)$ and $h_2(x)$

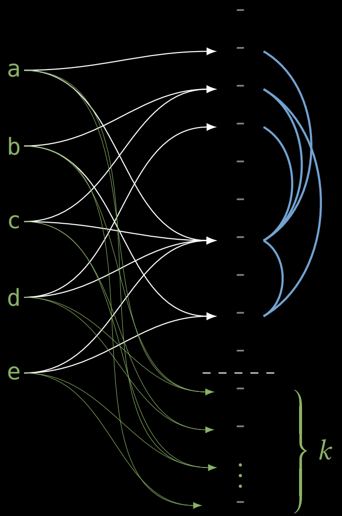
new garbled cuckoo PaXoS



for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

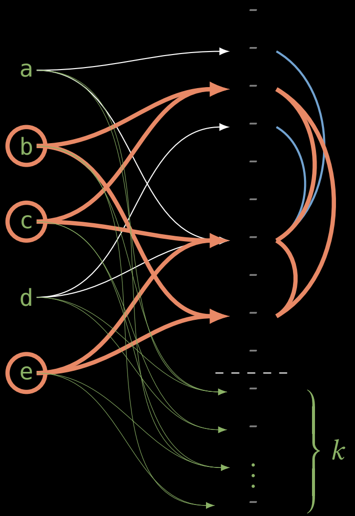
new garbled cuckoo PaXoS



for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

new garbled cuckoo PaXoS

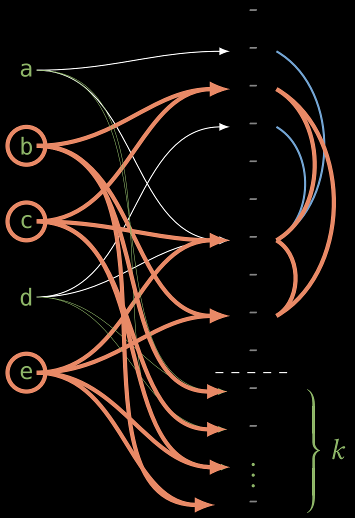


for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles

new garbled cuckoo PaXoS

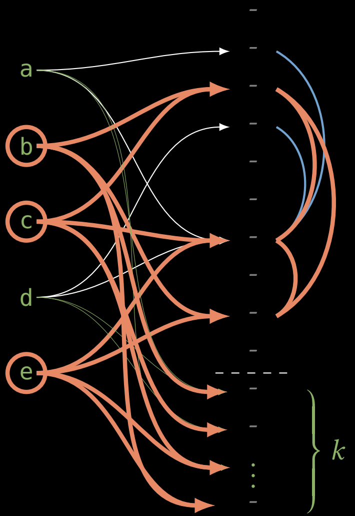


for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items

new garbled cuckoo PaXoS

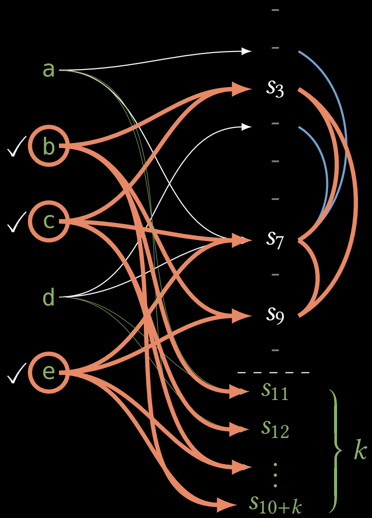


for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k >$ [# cycle items] $+ \lambda$

new garbled cuckoo PaXoS

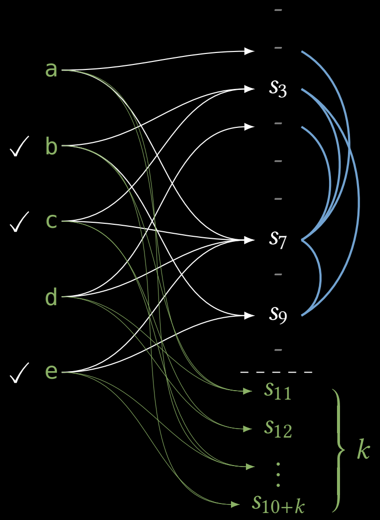


for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k >$ [# cycle items] $+ \lambda$
   - can be found in [# cycle items]^3 time

new garbled cuckoo PaXoS

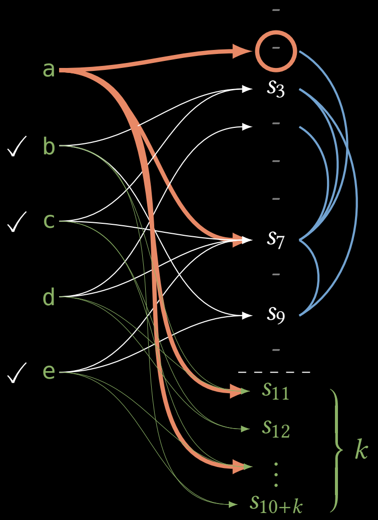


for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k >$ [# cycle items] $+ \lambda$
   - can be found in [# cycle items]^3 time
2. solve for remaining items **iteratively** (linear time)

new garbled cuckoo PaXoS

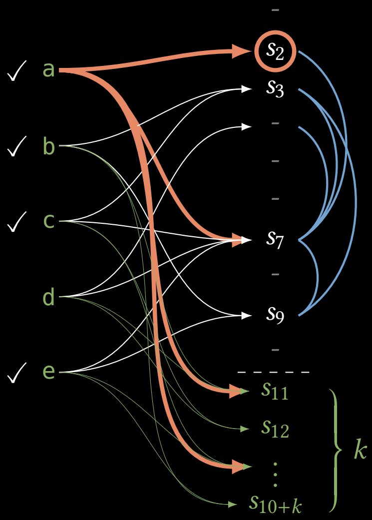


for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k >$ [# cycle items] $+ \lambda$
   - can be found in [# cycle items]^3 time
2. solve for remaining items **iteratively** (linear time)

new garbled cuckoo PaXoS

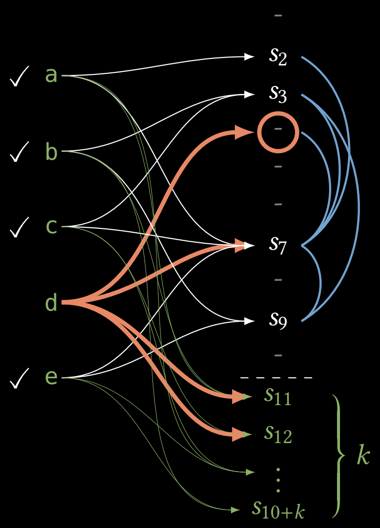


for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k >$ [# cycle items] + λ
   - can be found in [# cycle items]^3 time
2. solve for remaining items **iteratively** (linear time)
   - remaining cuckoo graph is acyclic

new garbled cuckoo PaXoS

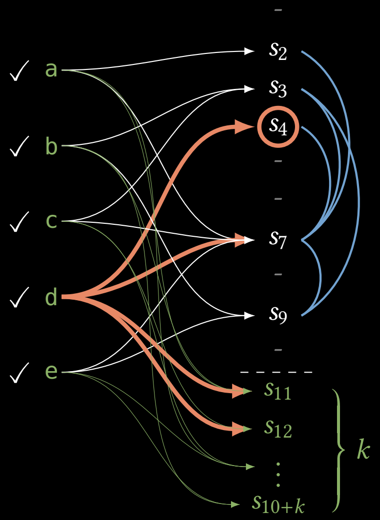


for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k >$ [# cycle items] + λ
   - can be found in [# cycle items]^3 time
2. solve for remaining items **iteratively** (linear time)
   - remaining cuckoo graph is acyclic

new garbled cuckoo PaXoS

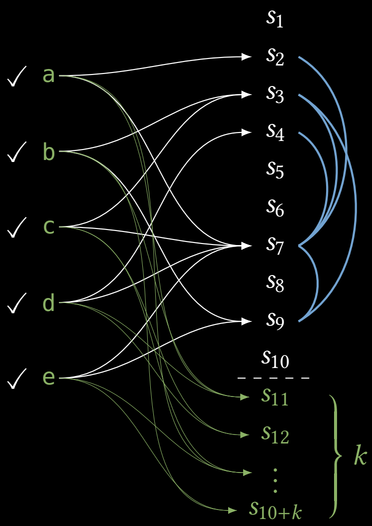


for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k >$ [# cycle items] $+ \lambda$
   - can be found in [# cycle items]^3 time
2. solve for remaining items **iteratively** (linear time)
   - remaining cuckoo graph is acyclic

new garbled cuckoo PaXoS

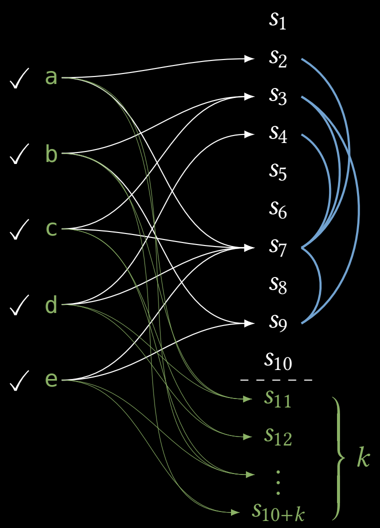


for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k >$ [# cycle items] + λ
   - can be found in [# cycle items]^3 time
2. solve for remaining items **iteratively** (linear time)
   - remaining cuckoo graph is acyclic

new garbled cuckoo PaXoS



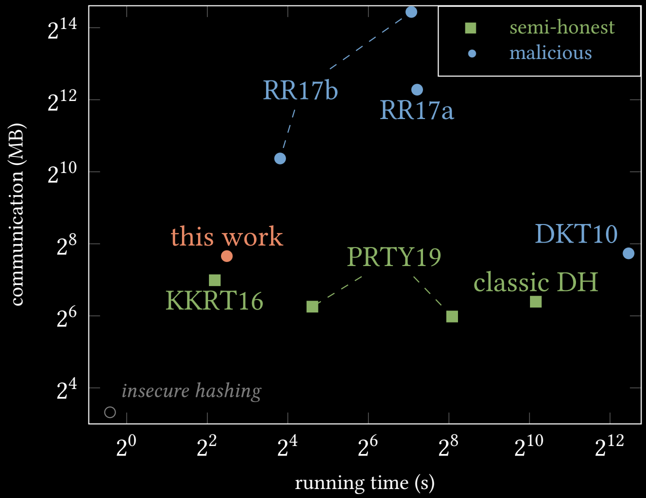
for each item x:

- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k >$ [# cycle items] $+ \lambda$
   - can be found in [# cycle items]^3 time
2. solve for remaining items **iteratively** (linear time)
   - remaining cuckoo graph is acyclic

whp: [# cycle items] is $O(\log n)$

summary



new approach for malicious PSI:

- fastest, least communication
- first $O(n)$ from OT extension
- almost as efficient as semi-honest

PaXoS data structure:

- encode items into a vector
- lookup is XOR of some positions
- first linear-time, constant rate construction

thanks!