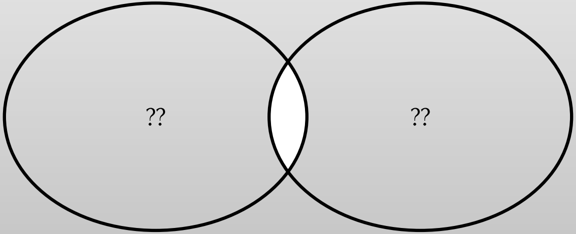
Private Set Intersection

Mike Rosulek
Oregon State University

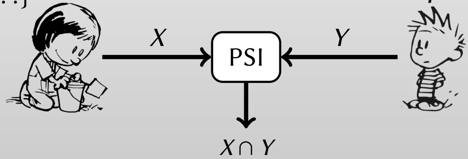
crypt@b-it 2018



Private set intersection (PSI)

Special case of secure 2-party computation:

$X = \{x_1, x_2, \ldots\}$ $\qquad$ $Y = \{y_1, y_2, \ldots\}$



PSI applications

Contact discovery, when signing up for WhatsApp

- $X =$ address book in my phone (phone numbers)
- $Y =$ WhatsApp user database

PSI applications

Contact discovery, when signing up for WhatsApp

- $X =$ address book in my phone (phone numbers)
- $Y =$ WhatsApp user database

Private scheduling

- $X =$ available timeslots on my calendar
- $Y =$ available timeslots on your calendar

PSI applications

Contact discovery, when signing up for WhatsApp

- $X =$ address book in my phone (phone numbers)
- $Y =$ WhatsApp user database

Private scheduling

- $X =$ available timeslots on my calendar
- $Y =$ available timeslots on your calendar

Ad conversion rate (PSI variant)

- $X =$ users who saw the advertisement
- $Y =$ customers who bought the product

PSI applications

Contact discovery, when signing up for WhatsApp

- $X =$ address book in my phone (phone numbers)
- $Y =$ WhatsApp user database

Private scheduling

- $X =$ available timeslots on my calendar
- $Y =$ available timeslots on your calendar

Ad conversion rate (PSI variant)

- $X =$ users who saw the advertisement
- $Y =$ customers who bought the product

No-fly list

- $X =$ passenger list of flight 123
- $Y =$ government no-fly list

“Obvious” protocol

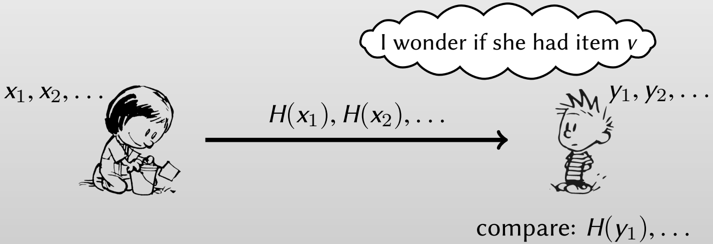
$x_1, x_2, \ldots$

$H(x_1), H(x_2), \ldots$ $\longrightarrow$

$y_1, y_2, \ldots$

compare: $H(y_1), \ldots$

“Obvious” protocol



INSECURE: Receiver can test *any* $v \stackrel{?}{\in} \{x_1, \ldots, x_n\}$, **offline**

- Problematic if items have low entropy (e.g., phone numbers)

Classical protocol [Meadows86,HubermanFranklinHogg99]

special case: each party has **just one** item

x

y

Classical protocol [Meadows86,HubermanFranklinHogg99]

special case: each party has **just one** item

x

$\xrightarrow{H(x)^{\alpha}}$

$\xleftarrow{H(y)^{\beta};\, H(x)^{\alpha\beta}}$

y

check: $H(x)^{\alpha\beta} \stackrel{?}{=} H(y)^{\beta\alpha}$

Idea:

- If $x = y$, then $H(x)^{\alpha\beta} = H(y)^{\beta\alpha}$
- If $x \neq y$, they are independently random (when H is random oracle)

Classical protocol [Meadows86,HubermanFranklinHogg99]

$x_1, x_2, \ldots$

$\xrightarrow{H(x_1)^\alpha, H(x_2)^\alpha, \ldots}$

$\xleftarrow{H(y_1)^\beta, H(x_2)^\beta, \ldots}$

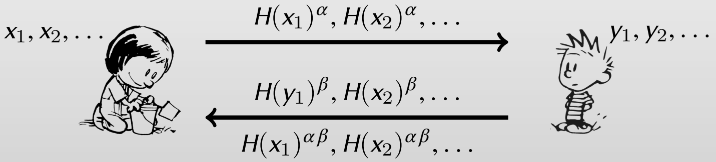
$H(x_1)^{\alpha\beta}, H(x_2)^{\alpha\beta}, \ldots$

$y_1, y_2, \ldots$

Idea:

- If $x = y$, then $H(x)^{\alpha\beta} = H(y)^{\beta\alpha}$
- If $x \neq y$, they are independently random (when H is random oracle)

Classical protocol [Meadows86,HubermanFranklinHogg99]



Idea:

- If $x = y$, then $H(x)^{\alpha\beta} = H(y)^{\beta\alpha}$
- If $x \neq y$, they are independently random (when H is random oracle)

Drawback: $O(n)$ **expensive** exponentiations

Roadmap

1

Crypto: Private equality tests:

- How to securely test whether **two strings** are identical
- Focus on building from OT (and similar primitives) in light of OT extension

2

Algorithmic: Hashing techniques

- How to reduce number of equality tests

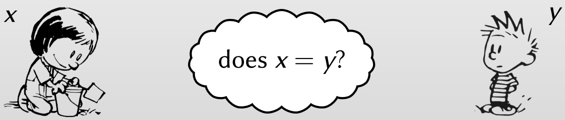
Simplest case: string equality

x

does $x = y$?

y

Simplest case: string equality



Using Yao's protocol: $(x, y \in \{0,1\}^\ell)$

- ℓ OTs
- Boolean circuit with $\ell - 1$ AND gates
- E.g.: $\ell = 64 \Rightarrow$ 48 Kbits

String equality from OT

$m_{1,0}$	$m_{1,1}$
$m_{2,0}$	$m_{2,1}$
$m_{3,0}$	$m_{3,1}$
$m_{4,0}$	$m_{4,1}$
$\vdots$	$\vdots$

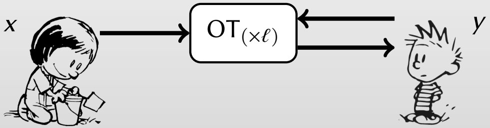
x

y

- Sender chooses 2ℓ **random** strings

String equality from OT

$m_{1,0}$	$m_{1,1}$
$m_{2,0}$	$m_{2,1}$
$m_{3,0}$	$m_{3,1}$
$m_{4,0}$	$m_{4,1}$
$\vdots$	$\vdots$



$y = 0110\cdots$

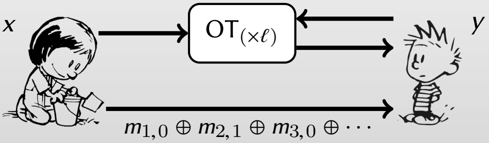
$m_{1,0}$	?
?	$m_{2,1}$
?	$m_{3,1}$
$m_{4,0}$	?
$\vdots$	$\vdots$

- Sender chooses 2ℓ **random** strings
- Receiver uses bits of y as OT choice bits

String equality from OT

$x = 0101\cdots$

$m_{1,0}$	$m_{1,1}$
$m_{2,0}$	$m_{2,1}$
$m_{3,0}$	$m_{3,1}$
$m_{4,0}$	$m_{4,1}$
$\vdots$	$\vdots$

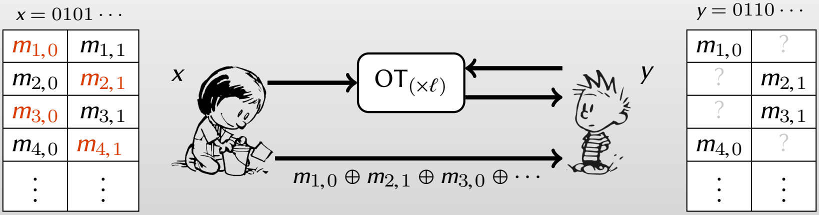


$y = 0110\cdots$

$m_{1,0}$	?
?	$m_{2,1}$
?	$m_{3,1}$
$m_{4,0}$	?
$\vdots$	$\vdots$

- Sender chooses 2ℓ **random** strings
- Receiver uses bits of y as OT choice bits
- **Summary value** of v defined as $\bigoplus_i m_{i,v_i}$
  - Sender can compute **any** summary value (in particular, for x)
  - Receiver can compute summary value **only** for y
  - Summary values other than y **look random** to receiver

String equality from OT



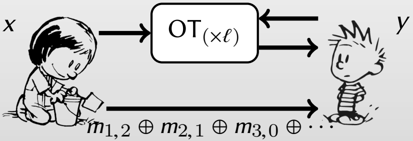
- Sender chooses 2ℓ **random** strings
- Receiver uses bits of y as OT choice bits
- **Summary value** of v defined as $\bigoplus_i m_{i,v_i}$
  - Sender can compute **any** summary value (in particular, for x)
  - Receiver can compute summary value **only** for y
  - Summary values other than y **look random** to receiver

Cost: just ℓ OTs

Improving equality tests [PinkasSchneiderZohner14]

$x = 2101\cdots$

$m_{1,0}$	$m_{1,1}$	$m_{1,2}$
$m_{2,0}$	$m_{2,1}$	$m_{2,2}$
$m_{3,0}$	$m_{3,1}$	$m_{3,2}$
$m_{4,0}$	$m_{4,1}$	$m_{4,3}$
$\vdots$	$\vdots$	$\vdots$



$y = 0122\cdots$

$m_{1,0}$	?	?
?	$m_{2,1}$	?
?	?	$m_{3,2}$
?	?	$m_{4,2}$
$\vdots$	$\vdots$	$\vdots$

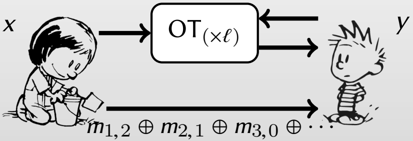
Idea: Instead of binary inputs, use **base-**k (base 3 in this example)

- Now only $\log_k \ell$ instances of 1-out-of-k OT

Improving equality tests [PinkasSchneiderZohner14]

$x = 2101\cdots$

$m_{1,0}$	$m_{1,1}$	$m_{1,2}$
$m_{2,0}$	$m_{2,1}$	$m_{2,2}$
$m_{3,0}$	$m_{3,1}$	$m_{3,2}$
$m_{4,0}$	$m_{4,1}$	$m_{4,3}$
$\vdots$	$\vdots$	$\vdots$



$y = 0122\cdots$

$m_{1,0}$	?	?
?	$m_{2,1}$	?
?	?	$m_{3,2}$
?	?	$m_{4,2}$
$\vdots$	$\vdots$	$\vdots$

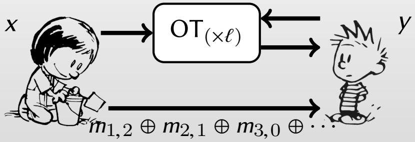
Idea: Instead of binary inputs, use **base-**k (base 3 in this example)

- Now only $\log_k \ell$ instances of 1-out-of-k OT
- **Note**: Only **random** OT required

Improving equality tests [PinkasSchneiderZohner14]

$x = 2101\cdots$

$m_{1,0}$	$m_{1,1}$	$m_{1,2}$
$m_{2,0}$	$m_{2,1}$	$m_{2,2}$
$m_{3,0}$	$m_{3,1}$	$m_{3,2}$
$m_{4,0}$	$m_{4,1}$	$m_{4,3}$
$\vdots$	$\vdots$	$\vdots$



$y = 0122\cdots$

$m_{1,0}$	?	?
?	$m_{2,1}$	?
?	?	$m_{3,2}$
?	?	$m_{4,2}$
$\vdots$	$\vdots$	$\vdots$

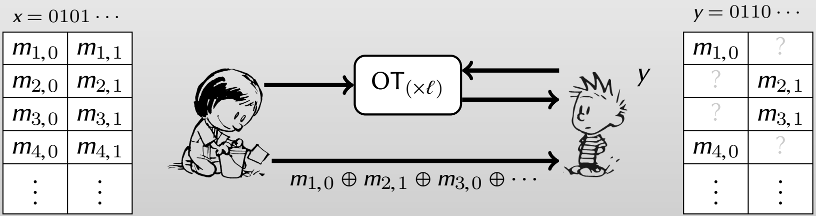
Idea: Instead of binary inputs, use **base-**k (base 3 in this example)

- Now only $\log_k \ell$ instances of 1-out-of-k OT
- **Note**: Only **random** OT required

Costs for different 1-out-of-k **random** OTs:

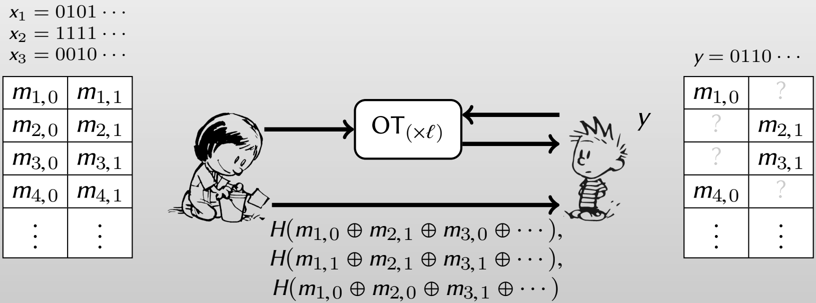
- Basic OT extension: $k = 2$: 128 bits/OT
- [KolesnikovKumaresan13]: $k = 2^8 \Rightarrow$ 3× fewer OTs @ 256 bits/OT
- [OrruOrsiniScholl16]: $k = 2^{76} \Rightarrow$ 76× fewer OTs @ 512 bits/OT
- [KolesnikovKumaresanRosulekTrieu16]: $k = \infty \Rightarrow$ $\sim$ 480 bits **total**

Another generalization



Private equality test: Alice has x, Bob has y, Bob learns $x \stackrel{?}{=} y$

Another generalization



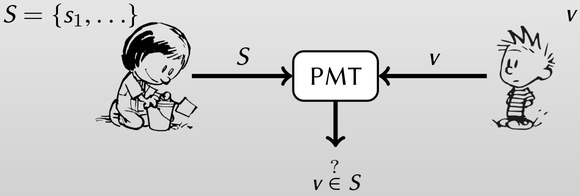
Private equality test: Alice has x, Bob has y, Bob learns $x \stackrel{?}{=} y$

Private set membership: Alice has set X, Bob has y, Bob learns $y \stackrel{?}{\in} X$

Roadmap

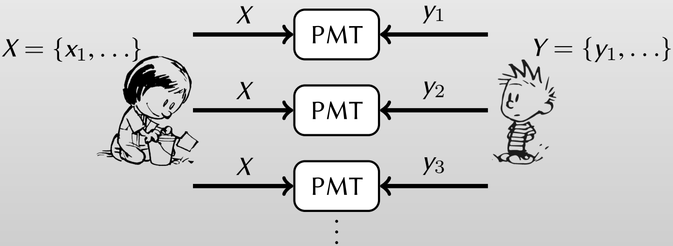
1. **Crypto:** Private equality tests:
   - How to securely test whether **two strings** are identical
2. **Algorithmic:** Hashing techniques
   - How to reduce number of equality tests

Building block

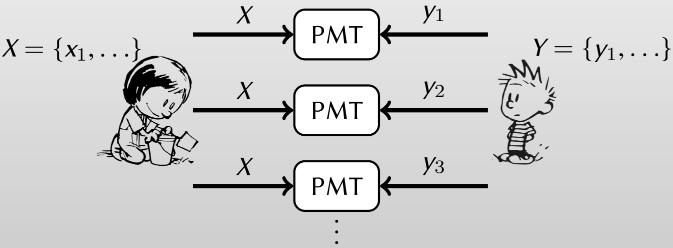


Cost: 1 OT primitive + sending *n* summary values

Dumb solution



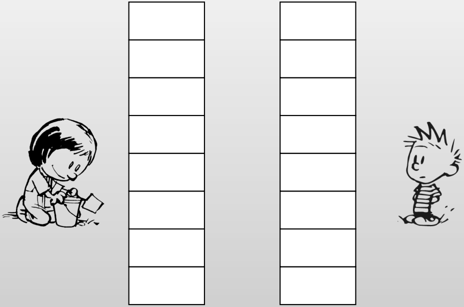
Dumb solution



Cost: $O(n^2)$

Better approach w/ hashing

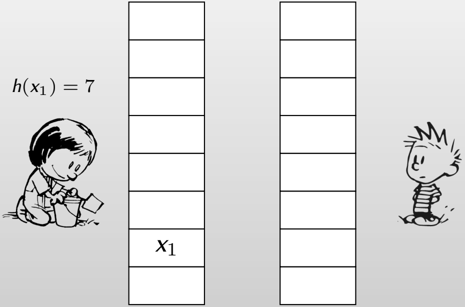
Agree on a random hash function $h : \{0,1\}^* \rightarrow [m]$



Better approach w/ hashing

Agree on a random hash function $h : \{0,1\}^* \rightarrow [m]$

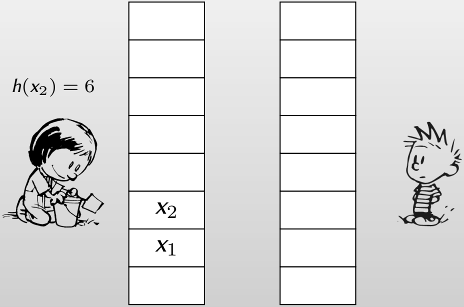
Assign item v to bin # $h(v)$



Better approach w/ hashing

Agree on a random hash function $h : \{0,1\}^* \to [m]$

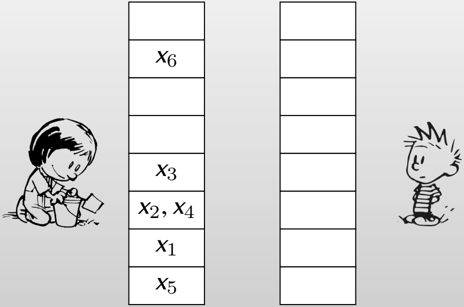
Assign item v to bin # $h(v)$



Better approach w/ hashing

Agree on a random hash function $h : \{0,1\}^* \to [m]$

Assign item v to bin # $h(v)$



Better approach w/ hashing

Agree on a random hash function $h : \{0,1\}^* \to [m]$

Assign item v to bin # $h(v)$

x_6	y_4
	y_1, y_6
x_3	y_3, y_5
x_2, x_4	
x_1	
x_5	y_2

Better approach w/ hashing

Agree on a random hash function $h : \{0,1\}^* \rightarrow [m]$

Assign item v to bin # $h(v)$

Do $\Theta(n^2)$ PSI **in each bin**

Idea: if both parties share an item v, **both** will put it in bin $h(v)$

	← PSI →	
x_6	← PSI →	y_4
	← PSI →	
	← PSI →	y_1, y_6
x_3	← PSI →	y_3, y_5
x_2, x_4	← PSI →	
x_1	← PSI →	
x_5	← PSI →	y_2

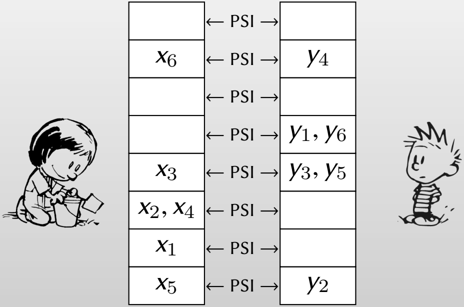
Better approach w/ hashing

Agree on a random hash function $h : \{0,1\}^* \to [m]$

Assign item v to bin # $h(v)$

Do $\Theta(n^2)$ PSI **in each bin**

Idea: if both parties share an item v, **both** will put it in bin $h(v)$



Cost: $\sum_i O(a_i b_i)$ where a_i, b_i = number of items in bin #i

- With n items into n bins, $E[\text{cost}] = O(n)$!

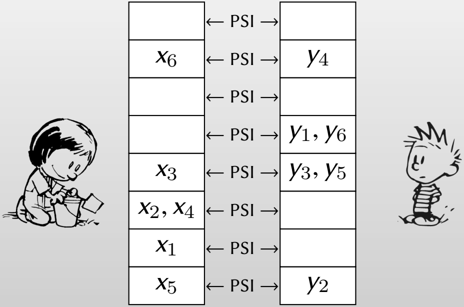
Better approach w/ hashing

Agree on a random hash function $h : \{0,1\}^* \to [m]$

Assign item v to bin # $h(v)$

Do $\Theta(n^2)$ PSI **in each bin**

Idea: if both parties share an item v, **both** will put it in bin $h(v)$



Cost: $\sum_i O(a_i b_i)$ where a_i, b_i = number of items in bin #i

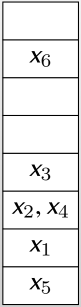
- With n items into n bins, $E[\text{cost}] = O(n)$!

Except, this is **completely insecure!** (why?)

Subtleties with hashing

"cost = $\sum_i O(a_i b_i)$" ??

- **only if a_i, b_i public**



Subtleties with hashing

"cost = $\sum_i O(a_i b_i)$" ??

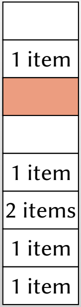
- **only if a_i, b_i public**

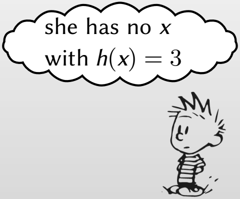
1 item
1 item
2 items
1 item
1 item

Subtleties with hashing

"cost = $\sum_i O(a_i b_i)$" ??

- **only if a_i, b_i public**





Subtleties with hashing

"cost = $\sum_i O(a_i b_i)$" ??

- **only if a_i, b_i public**

3 items
3 items
3 items
3 items
3 items
3 items
3 items
3 items

Solution:

1. Compute B such that $\Pr_h[\text{no bin has} > B \text{ items}] \leq 2^{-s}$ (balls in bins)
2. Add **dummy items** so that each bin has **exactly** B **items**

$\Rightarrow$ # (apparent) items per bin does not depend on input.

- (Protocol fails with probability 2^{-s})

Balls & bins questions

$$n \text{ balls} \overset{\text{randomly assign}}{\rightsquigarrow} m \text{ bins}$$

- Expected # balls per bin is n/m
- What is the **worst case** # balls in a bin (with high probability)?

Balls & bins questions

$$n \text{ balls} \overset{\text{randomly assign}}{\rightsquigarrow} m \text{ bins}$$

- Expected # balls per bin is n/m
- What is the **worst case** # balls in a bin (with high probability)?

Natural parameter choice: n items, n bins

- **Expected** balls per bin = 1
- **Worst-case** balls per bin = $O(\log n)$
- PSI cost = (# bins) × (worst-case load)^2 = $O(n \log^2 n)$

Balls & bins questions

$$n \text{ balls} \overset{\text{randomly assign}}{\rightsquigarrow} m \text{ bins}$$

- Expected # balls per bin is n/m
- What is the **worst case** # balls in a bin (with high probability)?

Natural parameter choice: n items, n bins

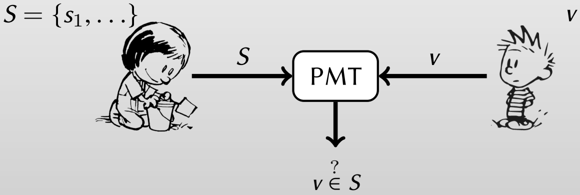
- **Expected** balls per bin = 1
- **Worst-case** balls per bin = $O(\log n)$
- PSI cost = (# bins) × (worst-case load)^2 = $O(n \log^2 n)$

Better parameter choice: n items, $O(n/\log n)$ bins [good to know!]

- **Expected** balls per bin = $O(\log n)$
- **Worst-case** balls per bin = $O(\log n)$
- PSI cost = $O(n \log n)$

Improved hashing

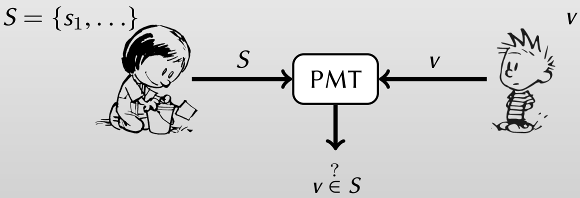
Remember:



Our basic building block naturally supports **one item** from Bob

Improved hashing

Remember:



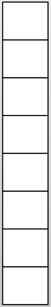
Our basic building block naturally supports **one item** from Bob

Idea: find hashing scheme that leaves only **1 item per bin**

- Only Bob needs to have 1 item per bin

Cuckoo hashing

Use **2** random hash functions h_1, h_2

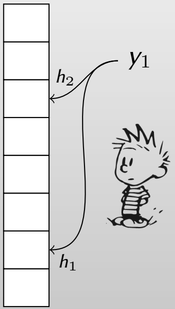


Cuckoo hashing

Use **2** random hash functions h_1, h_2

If either $h_1(y)$ or $h_2(y)$ is empty,

- put y in that bin

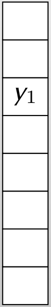


Cuckoo hashing

Use **2** random hash functions h_1, h_2

If either $h_1(y)$ or $h_2(y)$ is empty,

- put y in that bin

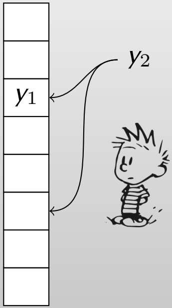


Cuckoo hashing

Use **2** random hash functions h_1, h_2

If either $h_1(y)$ or $h_2(y)$ is empty,

- put y in that bin

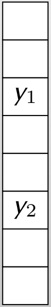


Cuckoo hashing

Use **2** random hash functions h_1, h_2

If either $h_1(y)$ or $h_2(y)$ is empty,

- put y in that bin



Cuckoo hashing

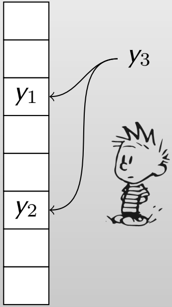
Use **2** random hash functions h_1, h_2

If either $h_1(y)$ or $h_2(y)$ is empty,

- put y in that bin

If $h_1(y)$ and $h_2(y)$ both occupied,

- **evict** someone y' and recurse on y'



Cuckoo hashing

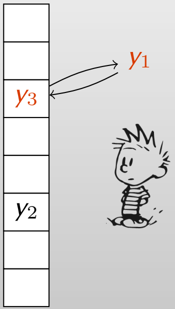
Use **2** random hash functions h_1, h_2

If either $h_1(y)$ or $h_2(y)$ is empty,

- put y in that bin

If $h_1(y)$ and $h_2(y)$ both occupied,

- **evict** someone y' and recurse on y'



Cuckoo hashing

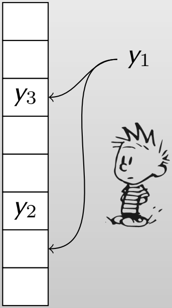
Use **2** random hash functions h_1, h_2

If either $h_1(y)$ or $h_2(y)$ is empty,

- put y in that bin

If $h_1(y)$ and $h_2(y)$ both occupied,

- **evict** someone y' and recurse on y'



Cuckoo hashing

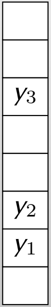
Use **2** random hash functions h_1, h_2

If either $h_1(y)$ or $h_2(y)$ is empty,

- put y in that bin

If $h_1(y)$ and $h_2(y)$ both occupied,

- **evict** someone y' and recurse on y'



Cuckoo hashing

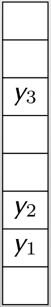
Use **2** random hash functions h_1, h_2

If either $h_1(y)$ or $h_2(y)$ is empty,

- put y in that bin

If $h_1(y)$ and $h_2(y)$ both occupied,

- **evict** someone y' and recurse on y'

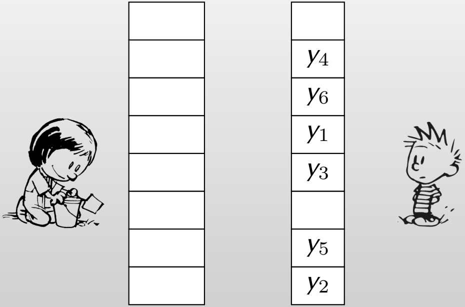


Claim: with sufficient bins, this process terminates with high probability

Cuckoo hashing for PSI [PinkasSchneiderZohner14]

Agree on h_1, h_2

Bob hashes with Cuckoo hashing

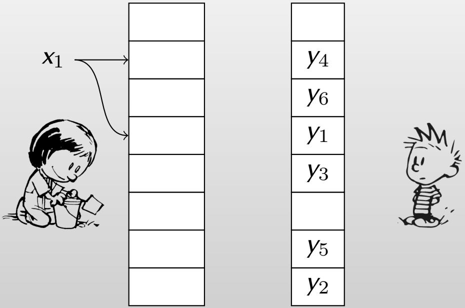


Cuckoo hashing for PSI [PinkasSchneiderZohner14]

Agree on h_1, h_2

Bob hashes with Cuckoo hashing

What about Alice?



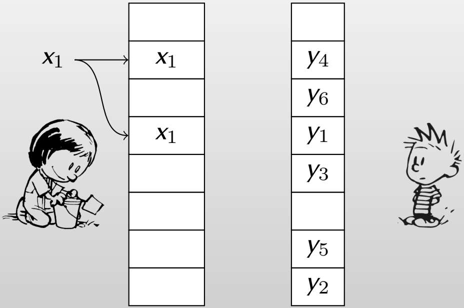
Cuckoo hashing for PSI [PinkasSchneiderZohner14]

Agree on h_1, h_2

Bob hashes with Cuckoo hashing

What about Alice?

- Place x in **both** $h_1(x)$ and $h_2(x)$



Cuckoo hashing for PSI [PinkasSchneiderZohner14]

Agree on h_1, h_2

Bob hashes with Cuckoo hashing

What about Alice?

- Place x in **both** $h_1(x)$ and $h_2(x)$

x_6, x_1
x_6
x_1, x_3
x_3, x_4
x_2, x_4
x_5
x_5, x_2

y_4
y_6
y_1
y_3
y_5
y_2

Cuckoo hashing for PSI [PinkasSchneiderZohner14]

Agree on h_1, h_2

Bob hashes with Cuckoo hashing

What about Alice?

- Place x in **both** $h_1(x)$ and $h_2(x)$

PMT in each bin

	$\leftarrow$ PMT $\rightarrow$	
x_6, x_1	$\leftarrow$ PMT $\rightarrow$	y_4
x_6	$\leftarrow$ PMT $\rightarrow$	y_6
x_1, x_3	$\leftarrow$ PMT $\rightarrow$	y_1
x_3, x_4	$\leftarrow$ PMT $\rightarrow$	y_3
x_2, x_4	$\leftarrow$ PMT $\rightarrow$	
x_5	$\leftarrow$ PMT $\rightarrow$	y_5
x_5, x_2	$\leftarrow$ PMT $\rightarrow$	y_2

Cuckoo hashing for PSI [PinkasSchneiderZohner14]

Agree on h_1, h_2

Bob hashes with Cuckoo hashing

What about Alice?

- Place x in **both** $h_1(x)$ and $h_2(x)$

PMT in each bin

	← PMT →	
x_6, x_1	← PMT →	y_4
x_6	← PMT →	y_6
x_1, x_3	← PMT →	y_1
x_3, x_4	← PMT →	y_3
x_2, x_4	← PMT →	
x_5	← PMT →	y_5
x_5, x_2	← PMT →	y_2

Idea: Only Bob gets output from PMT

- He places y in $h_?(y)$; if Alice also has y, it will also be here

Important: Alice cannot learn whether Bob placed y in $h_1(y)$ or $h_2(y)$

Cuckoo hashing PSI details

Don’t forget: Alice should pad with dummy items! ($2n$ balls in m bins)

	← PMT →	
x_6, x_1	← PMT →	y_4
x_6	← PMT →	y_6
x_1, x_3	← PMT →	y_1
x_3, x_4	← PMT →	y_3
x_2, x_4	← PMT →	
x_5	← PMT →	y_5
x_5, x_2	← PMT →	y_2

Cuckoo hashing PSI details

Don't forget: Alice should pad with dummy items! ($2n$ balls in m bins)

$\bot, \bot$	$\leftarrow$ PMT $\rightarrow$	
x_6, x_1	$\leftarrow$ PMT $\rightarrow$	y_4
$x_6, \bot$	$\leftarrow$ PMT $\rightarrow$	y_6
x_1, x_3	$\leftarrow$ PMT $\rightarrow$	y_1
x_3, x_4	$\leftarrow$ PMT $\rightarrow$	y_3
x_2, x_4	$\leftarrow$ PMT $\rightarrow$	
$x_5, \bot$	$\leftarrow$ PMT $\rightarrow$	y_5
x_5, x_2	$\leftarrow$ PMT $\rightarrow$	y_2

Cuckoo hashing PSI details

Don't forget: Alice should pad with dummy items! (2n balls in m bins)

- Bob too!

$\bot, \bot$	$\leftarrow$ PMT $\rightarrow$	$\bot'$
x_6, x_1	$\leftarrow$ PMT $\rightarrow$	y_4
$x_6, \bot$	$\leftarrow$ PMT $\rightarrow$	y_6
x_1, x_3	$\leftarrow$ PMT $\rightarrow$	y_1
x_3, x_4	$\leftarrow$ PMT $\rightarrow$	y_3
x_2, x_4	$\leftarrow$ PMT $\rightarrow$	$\bot'$
$x_5, \bot$	$\leftarrow$ PMT $\rightarrow$	y_5
x_5, x_2	$\leftarrow$ PMT $\rightarrow$	y_2

Cuckoo hashing PSI details

Don't forget: Alice should pad with dummy items! ($2n$ balls in m bins)

- Bob too!

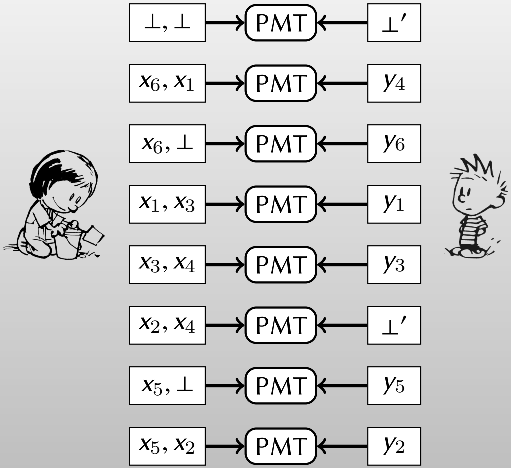
Cost:

- $\sim 1.5n$ bins for Cuckoo
- At most $O(\log n)$ items per bin for Alice

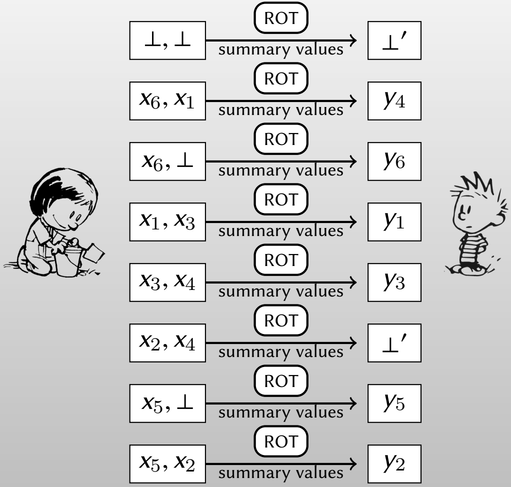
$\Rightarrow$ Still $O(n \log n)$ cost!

$\bot, \bot$	$\leftarrow$ PMT $\rightarrow$	$\bot'$
x_6, x_1	$\leftarrow$ PMT $\rightarrow$	y_4
$x_6, \bot$	$\leftarrow$ PMT $\rightarrow$	y_6
x_1, x_3	$\leftarrow$ PMT $\rightarrow$	y_1
x_3, x_4	$\leftarrow$ PMT $\rightarrow$	y_3
x_2, x_4	$\leftarrow$ PMT $\rightarrow$	$\bot'$
$x_5, \bot$	$\leftarrow$ PMT $\rightarrow$	y_5
x_5, x_2	$\leftarrow$ PMT $\rightarrow$	y_2

Avoiding dummy items

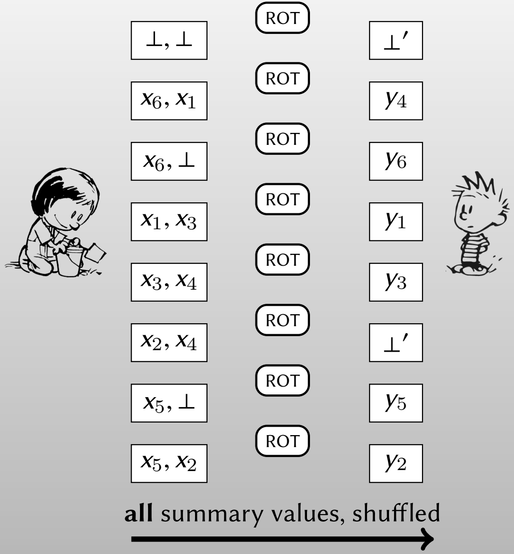


Avoiding dummy items



Avoiding dummy items

Summary values can be sent all together



Avoiding dummy items

Summary values can be sent all together

- No longer associated with bins

Previously:

- Can't leak # true items in a bin

Now:

- Everyone knows: n true items $\Rightarrow$ $2n$ true summary masks

$\Rightarrow$ Send only summary masks of true items

$\bot, \bot$	ROT	$\bot'$
x_6, x_1	ROT	y_4
$x_6, \bot$	ROT	y_6
x_1, x_3	ROT	y_1
x_3, x_4	ROT	y_3
x_2, x_4	ROT	$\bot'$
$x_5, \bot$	ROT	y_5
x_5, x_2	ROT	y_2

all summary values, shuffled →

Cuckoo PSI costs

Other details:

- Actually use Cuckoo hashing with **3 hash functions**

Costs:

- $\sim 1.5n$ Cuckoo bins
- $\sim 1.5n$ OT primitives
- $2n$ summary masks

$\Rightarrow$ total cost $O(n)$

Performance: [KolesnikovKumaresanRosulekTrieu16] = most efficient 1-out-of-∞ OT equality test

- PSI of 1 million items $\Rightarrow$ **3.8 seconds** @ 120 MB
- Insecure protocol (hash and send) $\Rightarrow$ **0.7 seconds** @ 10 MB