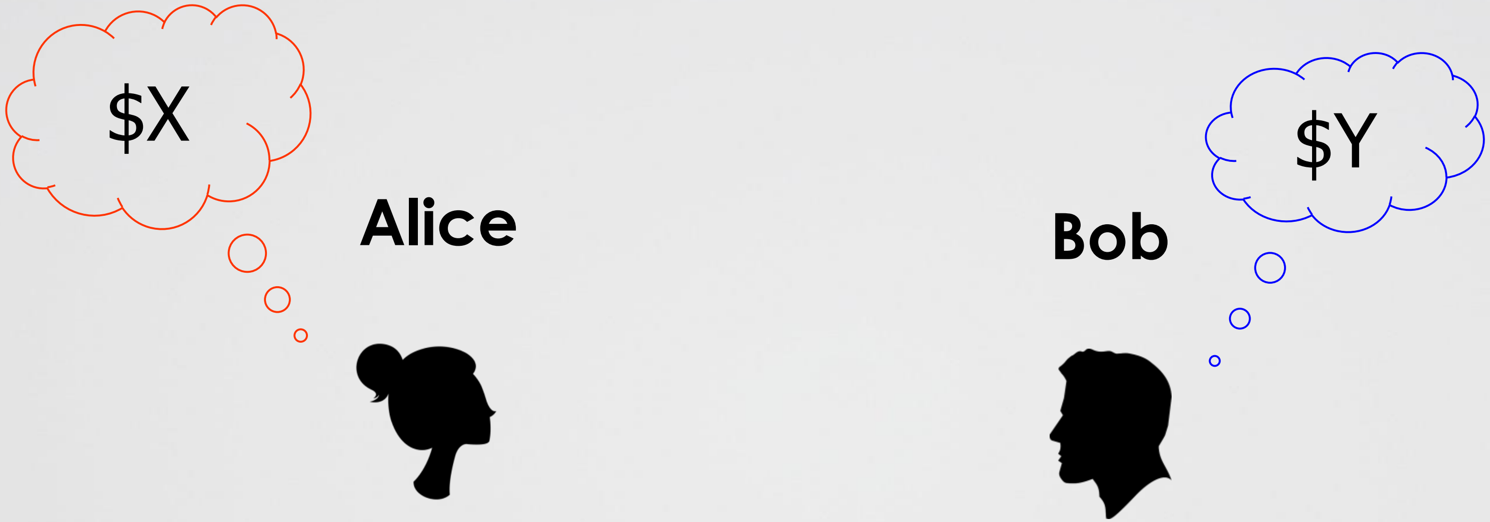
MPC beyond the Generic Model and Private Intersection Analytics

Benny Pinkas, Bar-Ilan University

(with many coauthors)

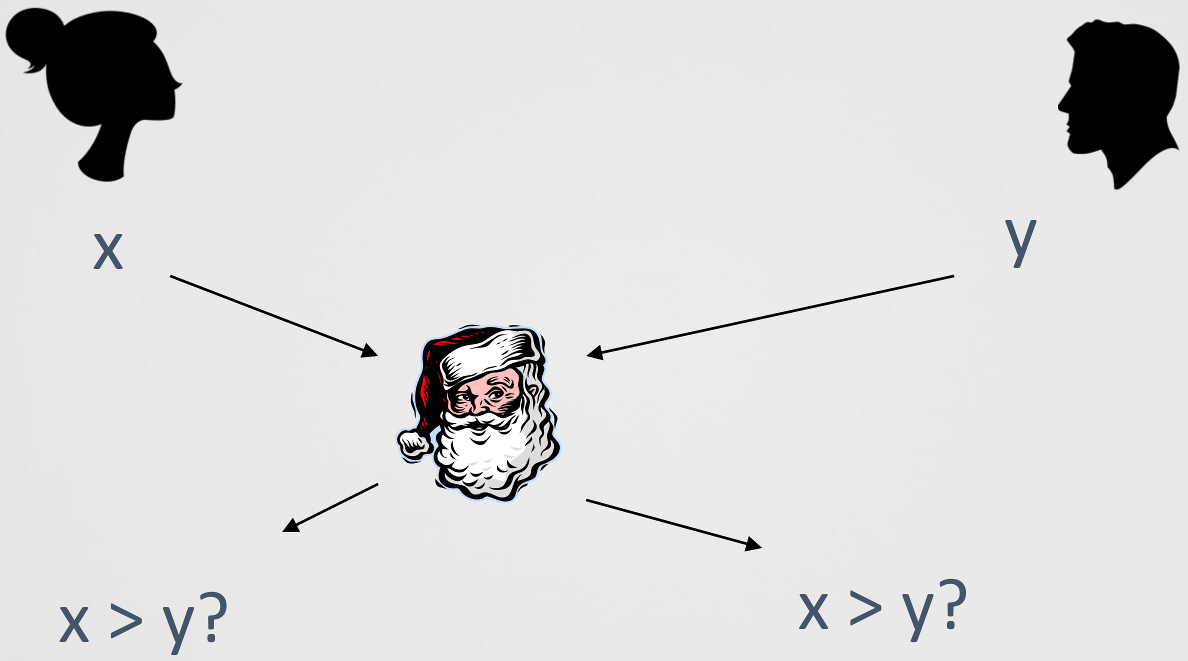
The millionaires' problem (Yao, 1982)



- Want to find out if X > Y
- But leak no other information! (even to each other)
- Standard crypto tools (encryption) do not help in this case!

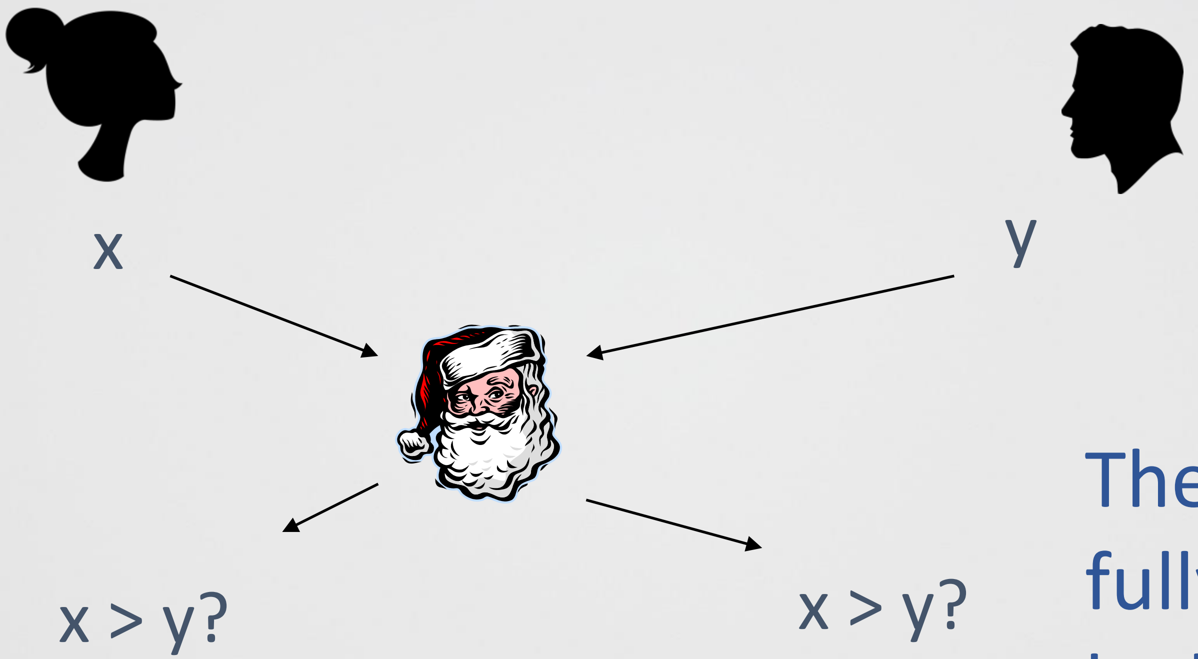
The millionaires' Problem

As if…



The millionaires' Problem

As if…



The trusted party is fully secure and leaks no information!

Research on MPC

- MPC started as a curious mental game/challenge
  - Millionaires problem, poker over the phone,…
- MPC research was theoretical for many years (1982 - 1998)
  - Focused on feasibility results and not on implementation
- This type of basic research is important

Applications of the millionaires problem?

Trading

- Alice: I want to sell x stocks, for a minimum price of P_{ASK}
- Bob: I want to buy y stocks, for a maximum price of P_{BID}

- Output:
  - If $P_{ASK} > P_{BID}$ then output “no deal”
  - Otherwise output “you can trade $\min(x,y)$ stocks”

- Auctions and bidding

Defining security

Desired Security of MPC

Input: x y

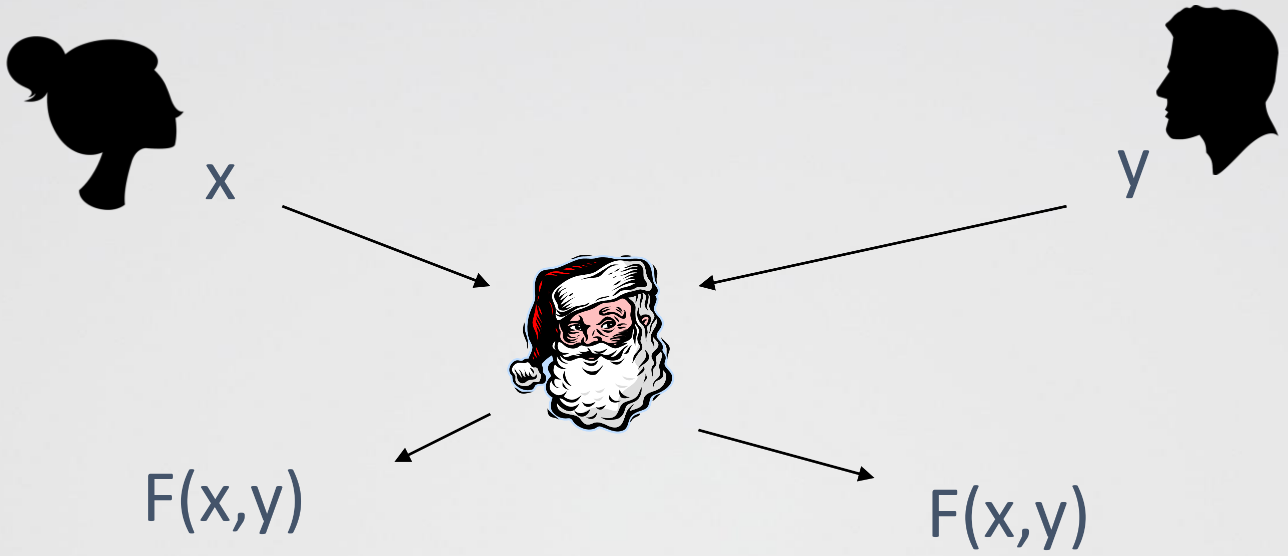
Output: F(x,y) *and nothing else*

As if…

x y

F(x,y) F(x,y)

Does this model make sense?



- We cannot hope for more privacy
- Does the trusted party scenario make sense?
  - Are the parties motivated to submit their true inputs?
  - Can they tolerate the disclosure of F(x,y)?
- If so, can implement the scenario without a trusted party

MPC is about a distributed implementation of a trusted party

Instead of

MPC does **not** tell you which function is safe to compute,

do

…

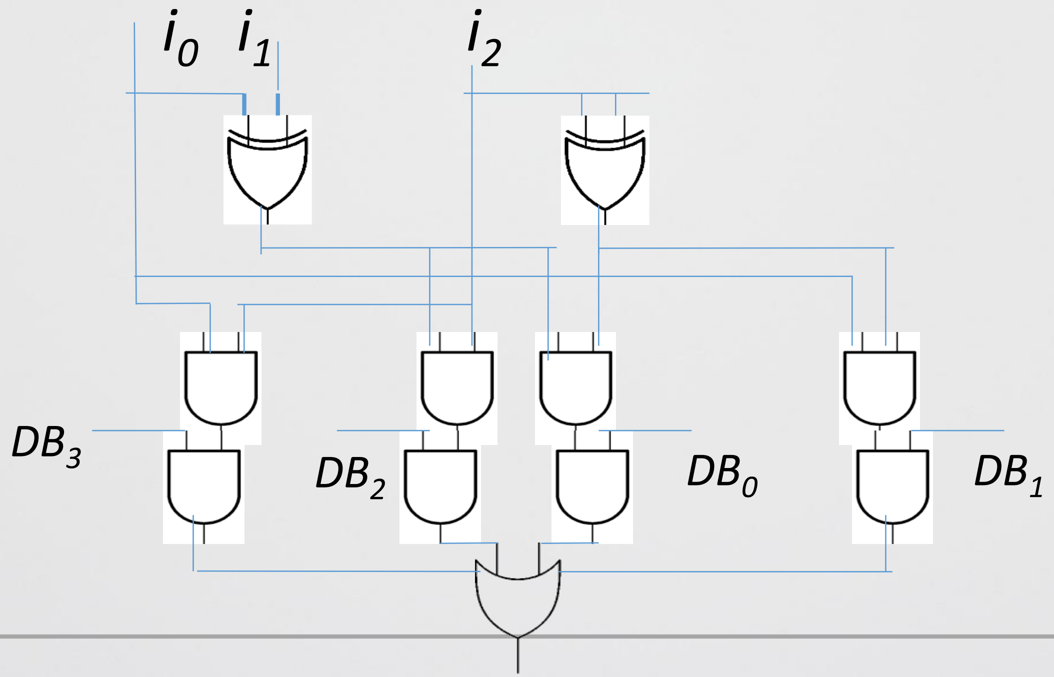
learn F(x,y) *and nothing else*

only how to compute it without trusted party

Implementing Secure Computation

Generic secure computation (Yao, 1982)

- Can be used to securely compute **any** function
- Considered theoretical, until the Fairplay system [MNPS04]
- Based on representing the function as a **Boolean circuit**



Implementing generic secure computation

- A lot of very smart optimizations in recent years
- Actual performance depends on circuit size, and on
    - setting
    - security requirements
    - preprocessing
    - engineering
  - Secure computation of AES: between 1μs to 1sec per block
  - We can easily handle circuits with 10^6 - 10^9 gates

MPC Beyond Generic Computation

What we know

1. Efficient circuit → efficient MPC protocol
2. Function with polynomial run time → circuit of polynomial size
3. (1) + (2) → if we can efficiently compute a function then we can also run an MPC computing it

- Overhead of MPC depends on the circuit representation

Examples

- Alice has integer x, Bob has integer y
  - Computing x+y, x-y
  - Computing whether x>y, max(x,y)
  - Computing $x \cdot y$, x^y
- X, Y are sets
  - Computing $X \cap Y$
  - Computing median(X,Y)
- X is an array, y is an index
  - Computing X[y]

Examples

- Alice has integer x, Bob has integer y
  - Computing $x+y$, $x-y$ (easy)
  - Computing whether $x>y$, $\max(x,y)$ (easy)
  - Computing $x \cdot y$, x^y (less easy)
- X, Y are sets
  - Computing $X \cap Y$ (less easy)
  - Computing median(X,Y) (less easy)
- X is an array, y is an index
  - Computing X[y] (not easy)

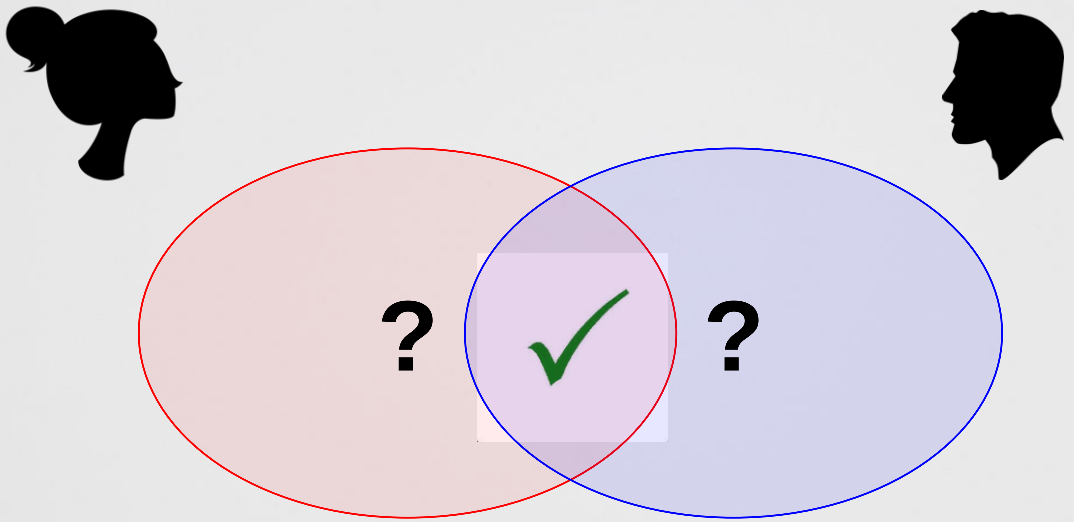
Specific vs. Generic Protocols

- Sometimes we can design a specific protocol for a specific problem, which will be more efficient than a generic, circuit-based protocol
- Still, it is preferable to use a circuit-based generic protocol
- We'll now show how to get the best of both worlds

Private Set Intersection (PSI) and Analytics

PSI Background, and Why Circuit-Based PSI?

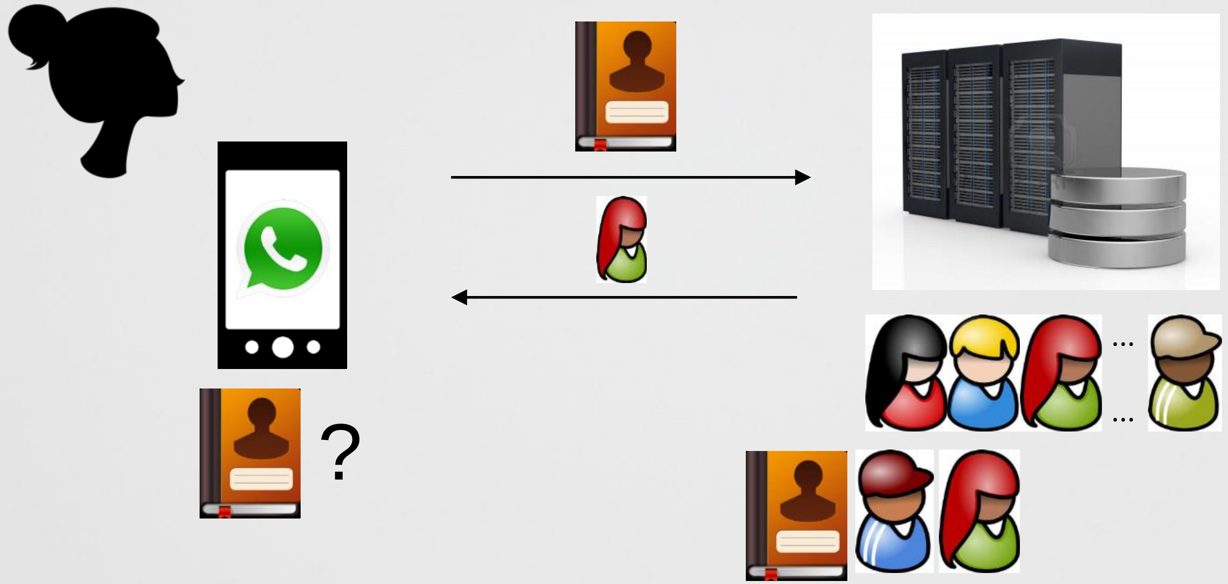
Private Set Intersection (PSI)



Applications of PSI

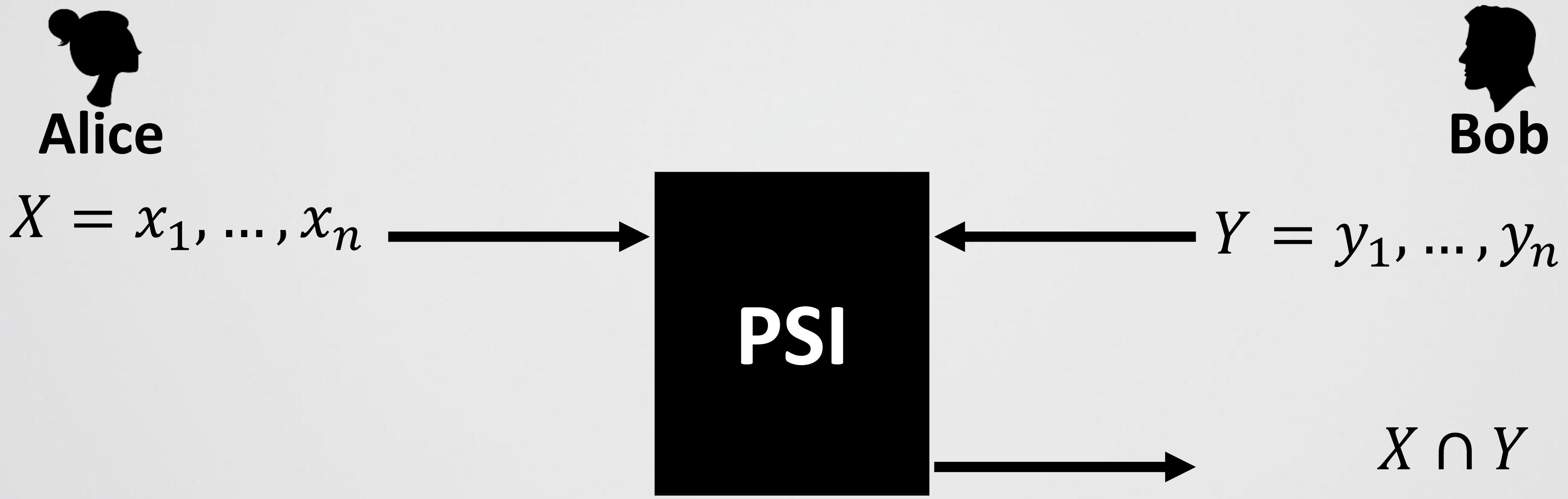
- **Information sharing**, e.g., intersection of threat information
- **Matching**, e.g., testing compatibility of different properties (preferences, genomes…)
- **Join** DB operations
- **Analytics:** $\Pr(A / B) = \Pr(A \cap B) / \Pr(B)$
- **Identifying mutual contacts** (Signal app)
- **Computing ad conversion rates** (Google)

Application: Common Contacts



Private Set Intersection (PSI)

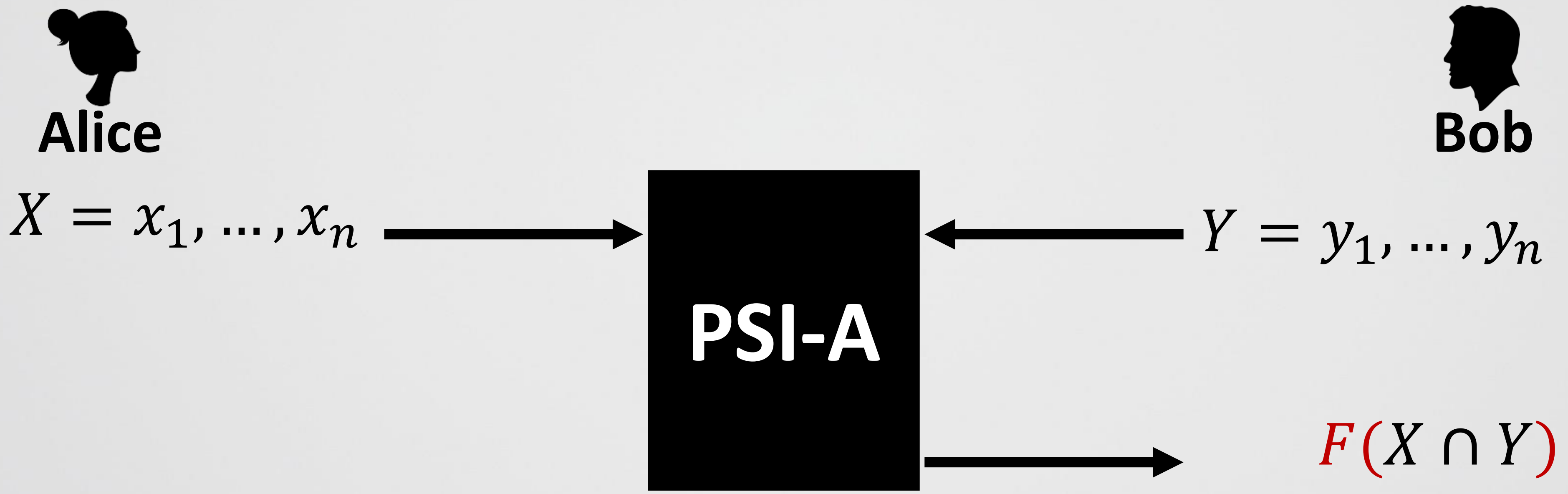
Definition



Private Set Intersection (PSI) - Analytics

Definition

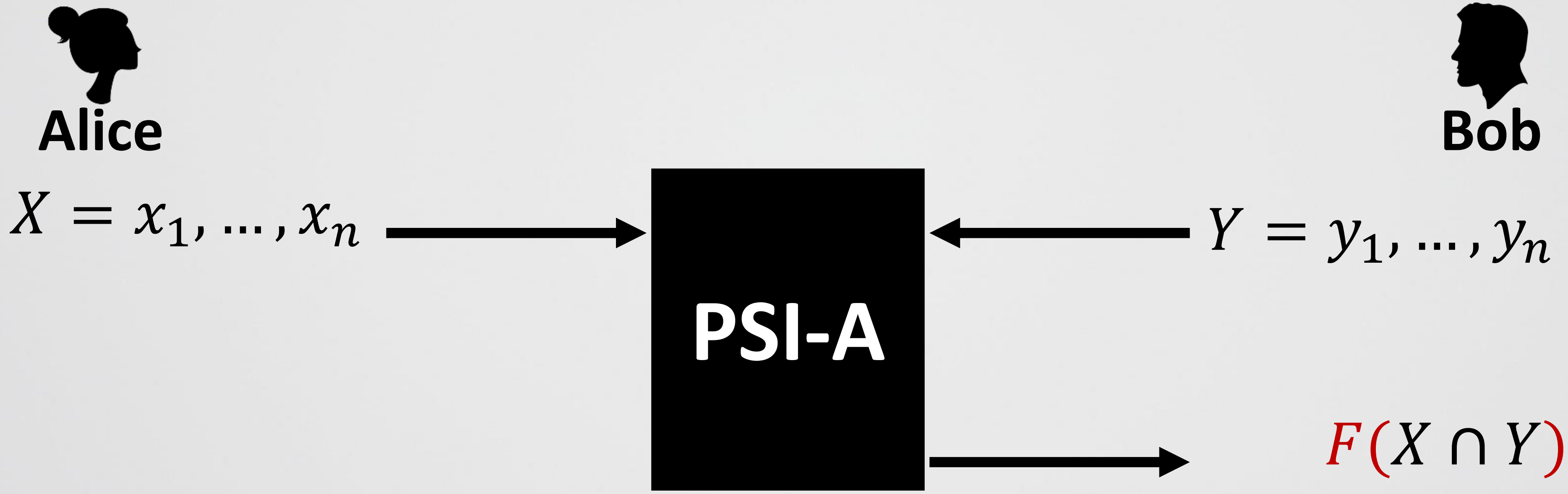
1. Post processing function F.   E.g. $F = |X \cap Y|$



Private Set Intersection (PSI) - Analytics

Definition

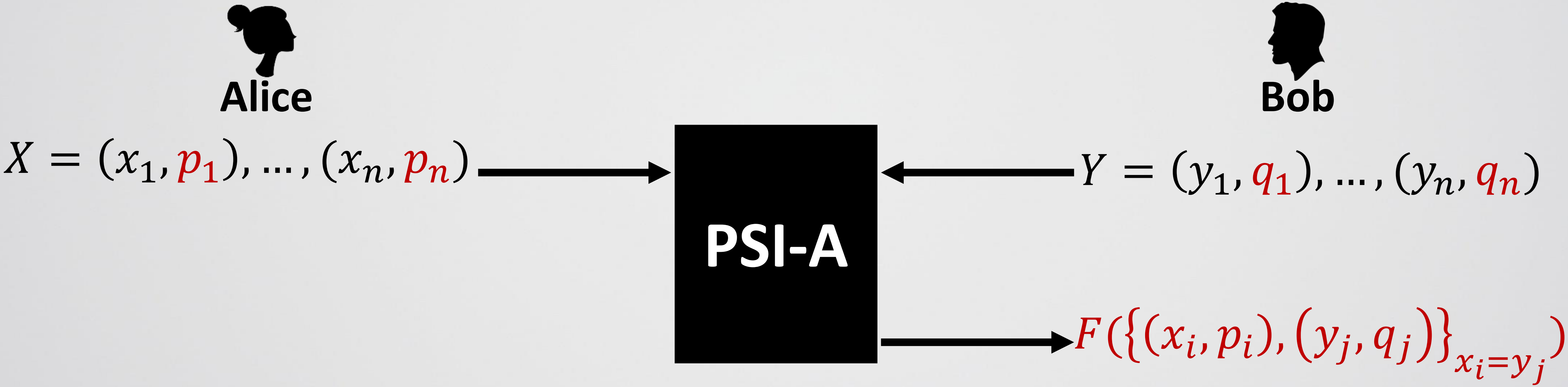
1. Post processing function F.   E.g. $F = |X \cap Y|$
2. Items are associated with payloads (aka. PSI with data-transfer)



Private Set Intersection (PSI) - Analytics

Definition

1. Post processing function F. E.g. $F = |X \cap Y|$
2. Items are associated with payloads (aka. PSI with data-transfer)



Private Set Intersection (PSI) - Analytics

Applications

Private Intersection-Sum Protocol with Applications to Attributing Aggregate Ad Conversions

Mihaela Ion[†], Ben Kreuter[†], Erhan Nergiz[†], Sarvar Patel[†], Shobhit Saxena[†], Karn Seth[†], David Shanahan[†] and Moti Yung[‡*]

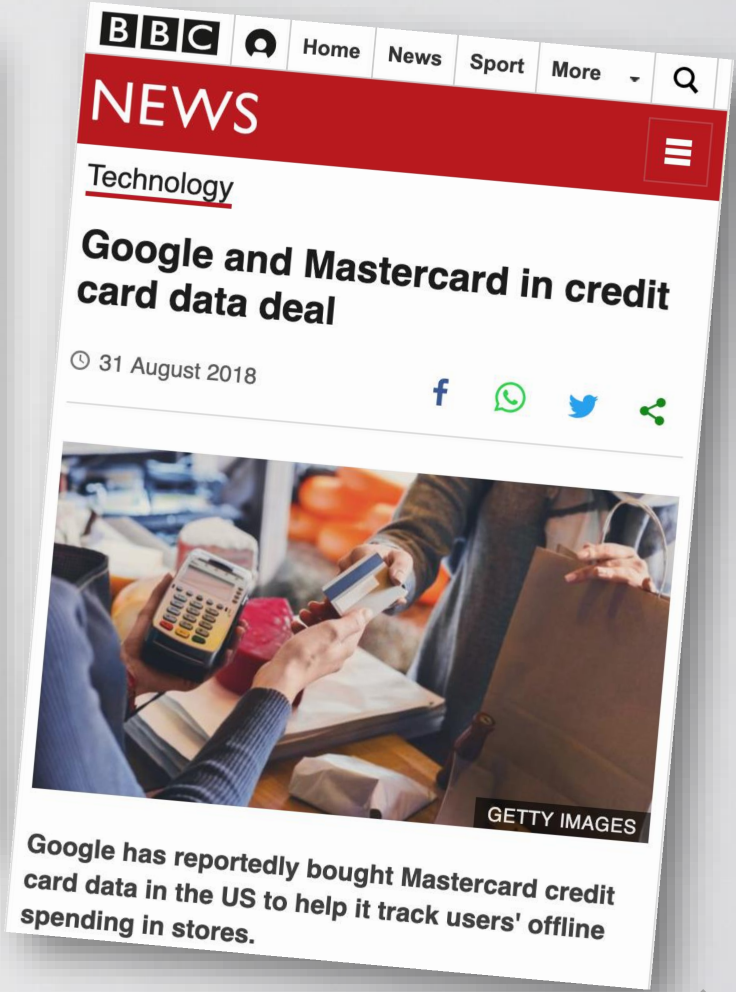
[†]{mion, benkreuter, anergiz, sarvar, shobhitsaxena, karn, dshanahan}@google.com

Google Inc.

[‡]moti@cs.columbia.edu

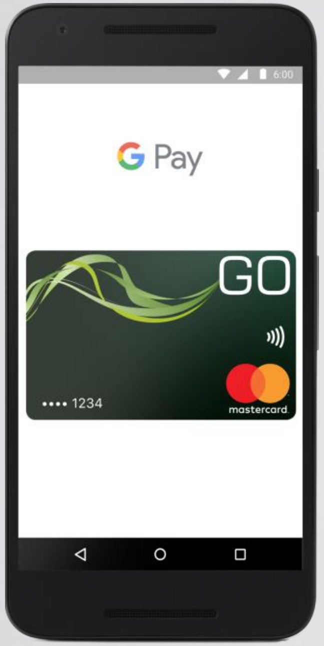
Columbia University and Snap Inc.

July 31, 2017



Private Set Intersection (PSI) - **Analytics**

Applications: **Online Ads to Offline Purchase Conversion**

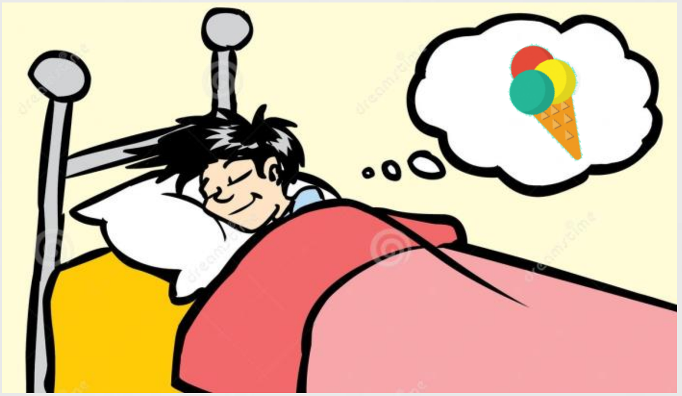


Register

1

Watch online ad

2



Dream about it

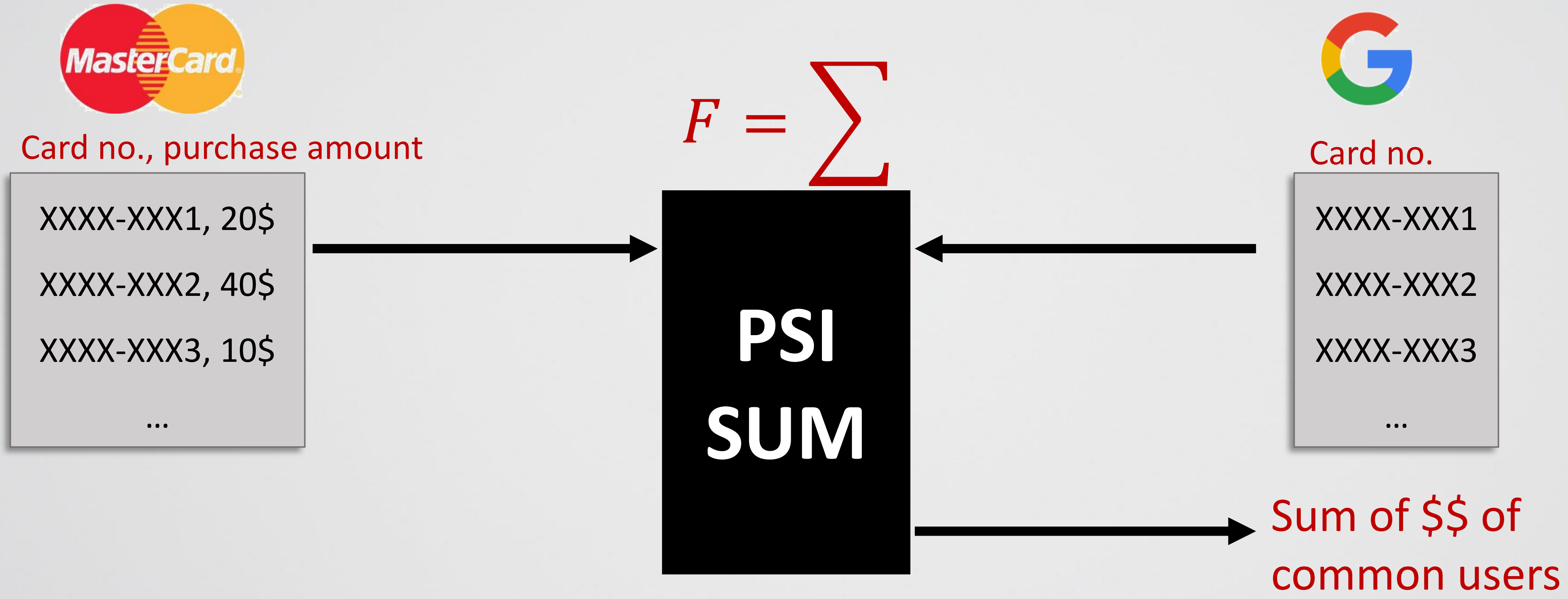
3

Buy offline

4

Private Set Intersection (PSI) - Analytics

Applications: Online Ads to Offline Purchase Conversion



Implementing PSI

Private Set Intersection (PSI)

Naïve solution

Insecure when items have low entropy

Alice

Bob

$H(x_1), \dots, H(x_n)$

Compare to
$H(y_1), \dots, H(y_n)$

Public-key based Protocols for PSI

(for example, based on the Diffie-Hellman assumption)

PSI based on Diffie-Hellman

- [S80, M86, HFH99, AES03]:

(ignore the details)

α 

$x_1,\ldots,x_n$

 β

$y_1,\ldots,y_n$

$(H(x_1))^\alpha,\ldots,(H(x_n))^\alpha \longrightarrow$

$\longleftarrow (H(y_1))^\beta,\ldots,(H(y_n))^\beta$

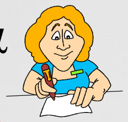
$\longleftarrow ((H(x_1))^\alpha)^\beta,\ldots,((H(x_n))^\alpha)^\beta$

$((H(y_1))^\beta)^\alpha,\ldots,((H(y_n))^\beta)^\alpha$

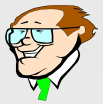
Compares the two lists

PSI based on Diffie-Hellman

- [S80, M86, HFH99, AES03]:

α 

$x_1,\ldots,x_n$

 β

$y_1,\ldots,y_n$

$(H(x_1))^\alpha,\ldots,(H(x_n))^\alpha \longrightarrow$

$\longleftarrow (H(y_1))^\beta,\ldots,(H(y_n))^\beta$

$\longleftarrow ((H(x_1))^\alpha)^\beta,\ldots,((H(x_n))^\alpha)^\beta$

$((H(y_1))^\beta)^\alpha,\ldots,((H(y_n))^\beta)^\alpha$

Compares the two lists

- Simple to understand ☺
- Simple to implement ☺
- Can be based on elliptic-curve crypto ☺
- Minimal communication ☺ but a lot of computation ☹

More recent PSI constructions [PSZ1,PSSZ15,KKRT16]

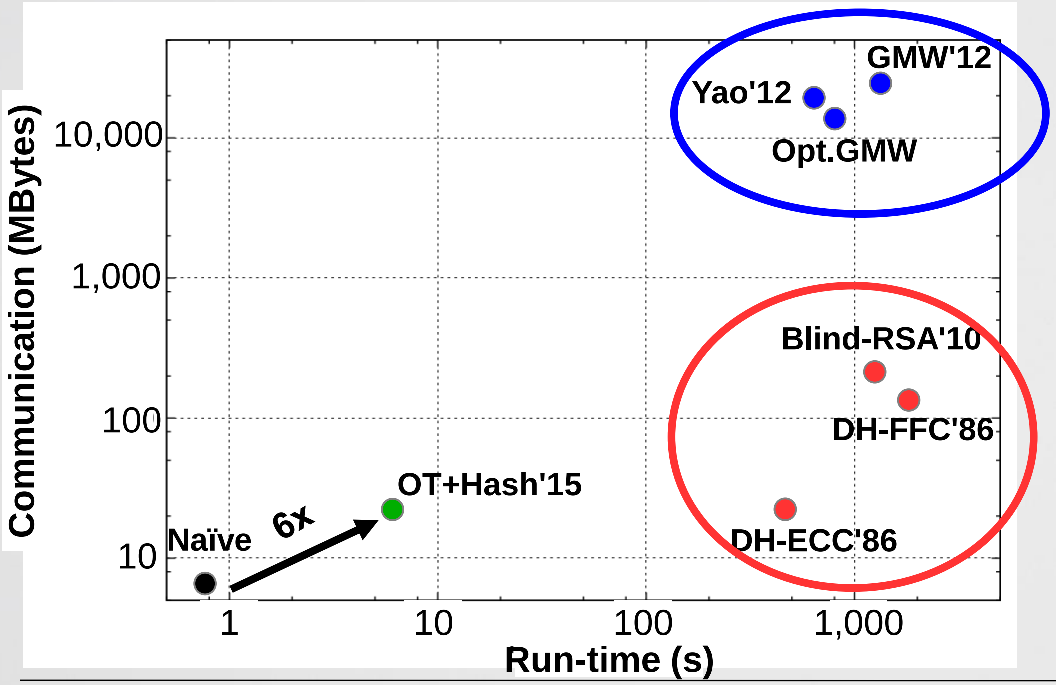
- PSI is “equivalent” to oblivious transfer

- Oblivious transfer extension [IKNP04] is very fast, and can enable very efficient PSI

- Used different hashing ideas to dramatically reduce the overhead of PSI

Performance Classification of PSI protocols [PSZ]

- PSI on $n = 2^{18}$ elements of $s=32$-bit length for 128-bit security on Gbit LAN

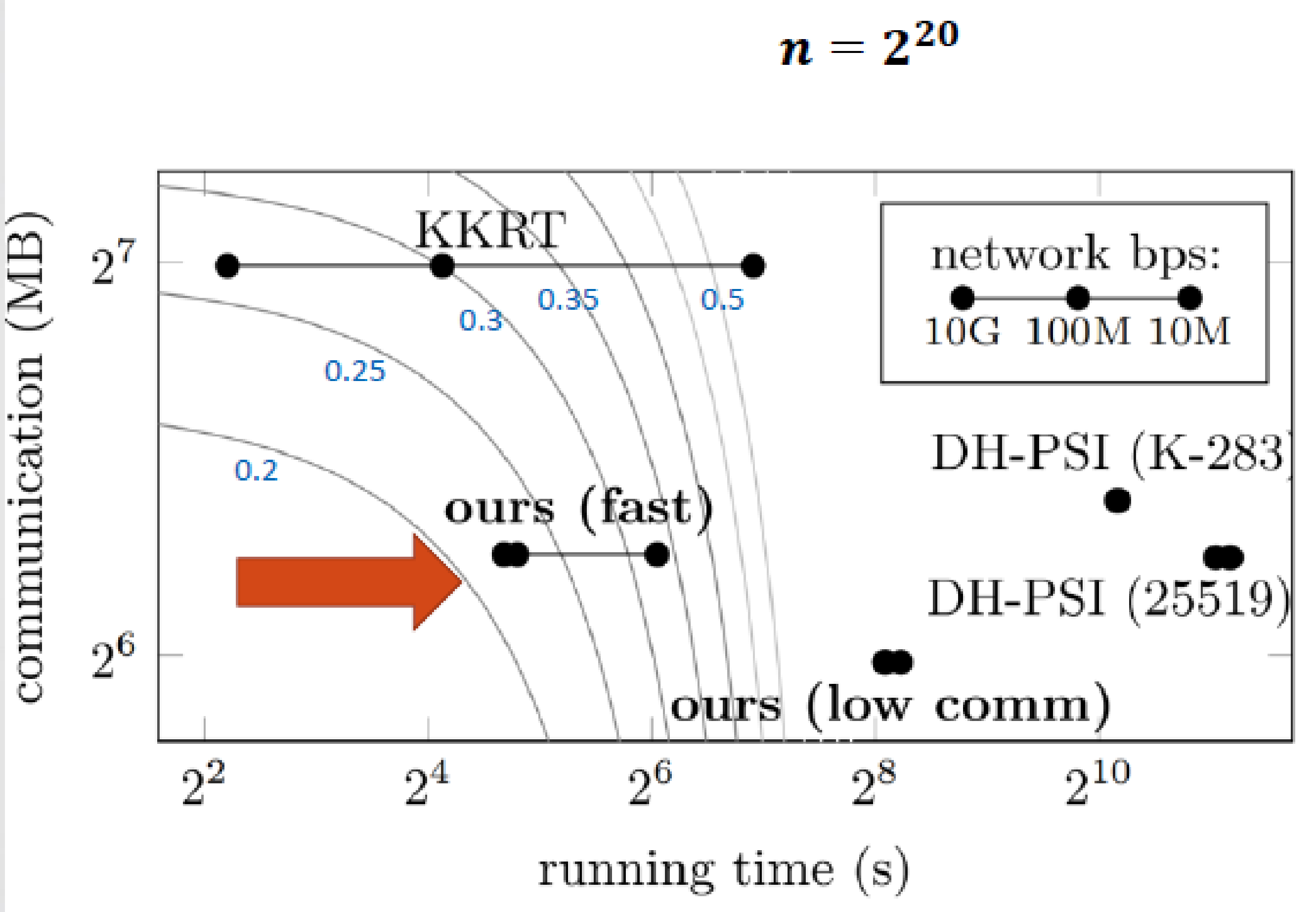


Circuit-Based (PSI analytics):
- high run-time & communication, but easily extensible to arbitrary functions

PK-Based: (starting from [S80,M86])
- high run-time
+ best communication

OT-Based:
[PSZ15,PSSZ16,KKRT16] good communication and run-time

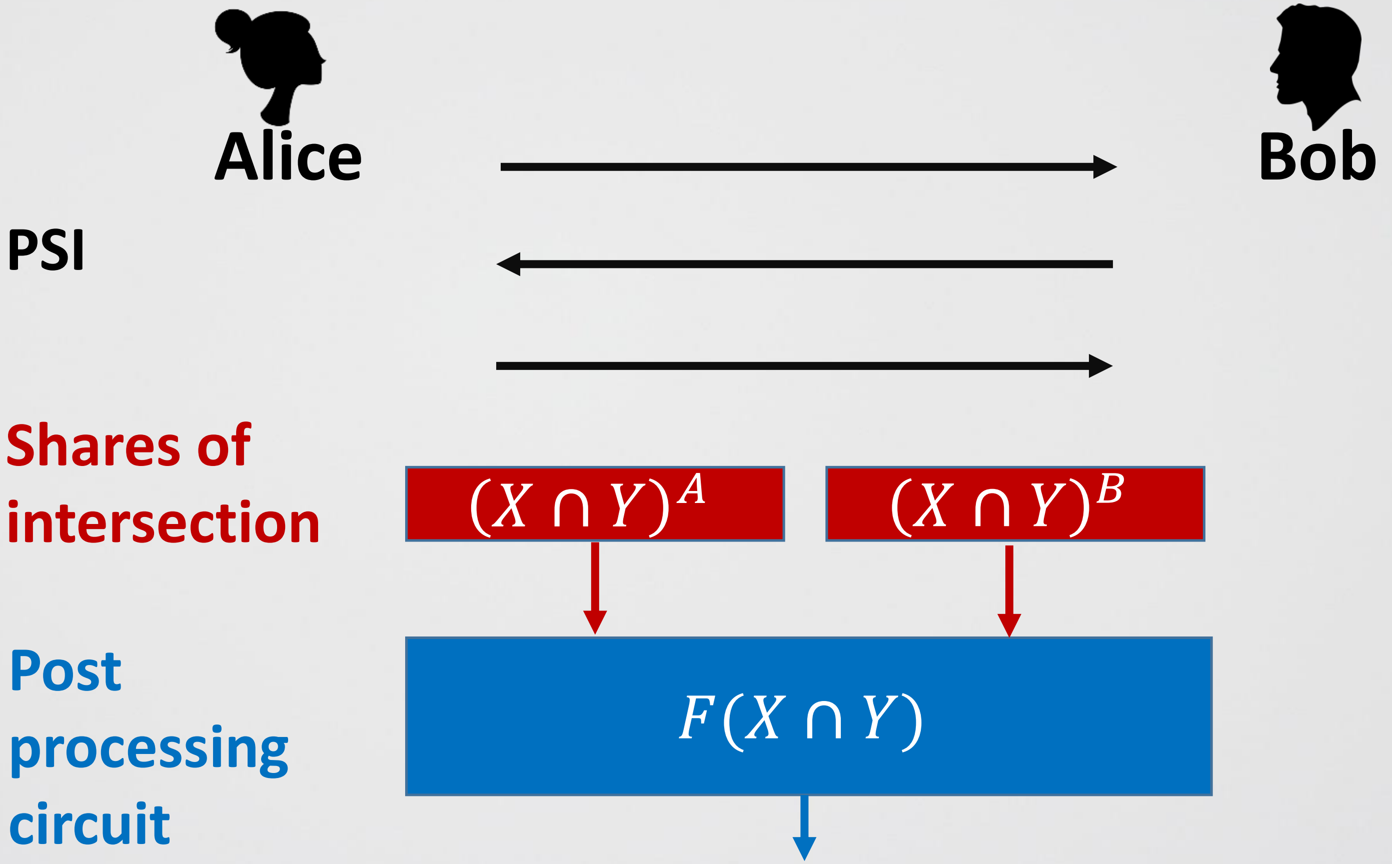
SpOT PSI (Crypto 2019 [PRTY])



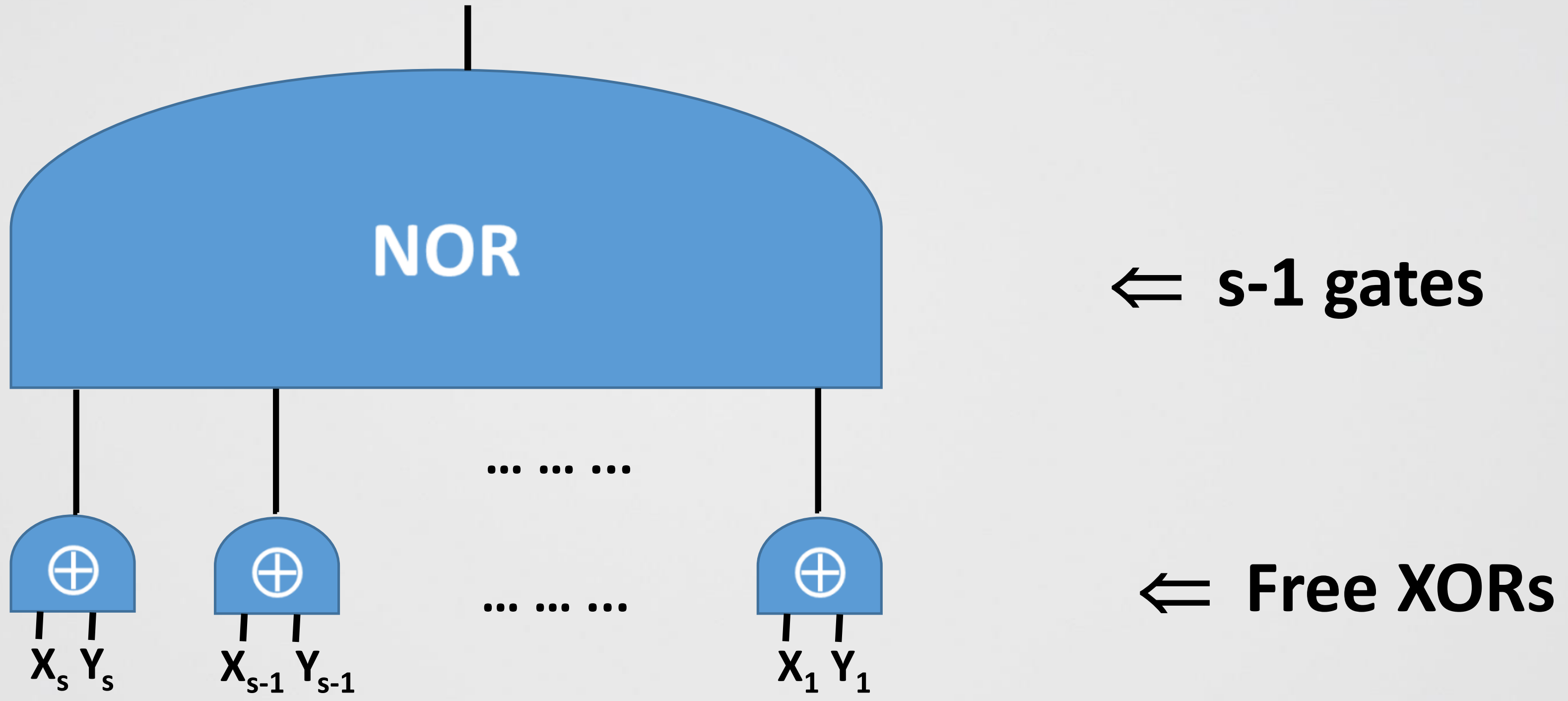
Private Set Intersection (PSI) - Analytics

Also known as: **Circuit-Based PSI**

- Immediate security for any F
- Modularity
- Existing code base



A circuit comparing two s-bit values (x=y?)



The Algorithmic Challenge

- Goal: Find the smallest circuit for computing the intersection
  - Alice and Bob **can prepare** their inputs
  - Circuit must **not** depend on data!

- Any symmetric function of the intersection could be then added
  - E.g., the **size** of the intersection, or whether size is greater than some **threshold**, potentially after adding noise to ensure **differential privacy**

Known circuit-based protocols for PSI

- A naïve circuit for PSI uses n^2 comparisons
- A protocol based on sorting networks – $O(n \log n)$ comparisons [HEK12]
- A protocol based on OT and hashing– $O(n \log n / \log\log n)$ comparisons [PSSZ16]

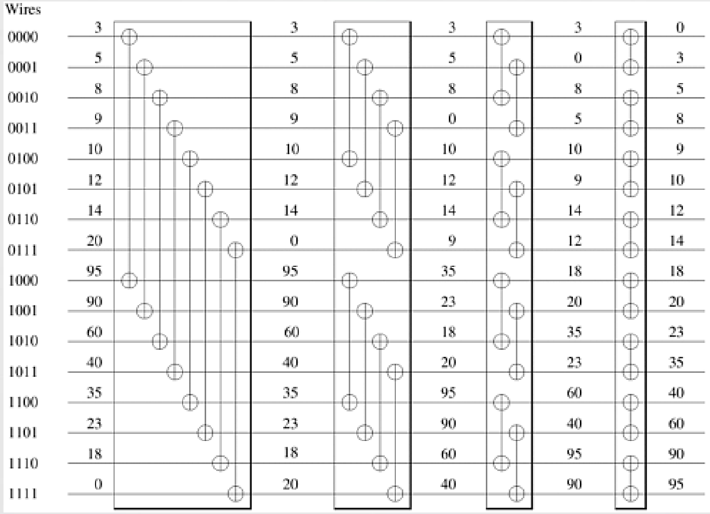
Known circuit-based protocols for PSI

- A naïve circuit for PSI uses n^2 comparisons
- A protocol based on sorting networks – $O(n \log n)$ comparisons [HEK12]
- A protocol based on OT and hashing– $O(n \log n / \log\log n)$ comparisons [PSSZ16]

- We reduced the overhead to $O(n)$ [PSTY19]

A circuit based PSI protocol [HEK12]

- A PSI circuit that has three steps
  - Sort: merge two sorted lists using a bitonic merging network [Bat68]. Uses $n\log(2n)$ comparisons.

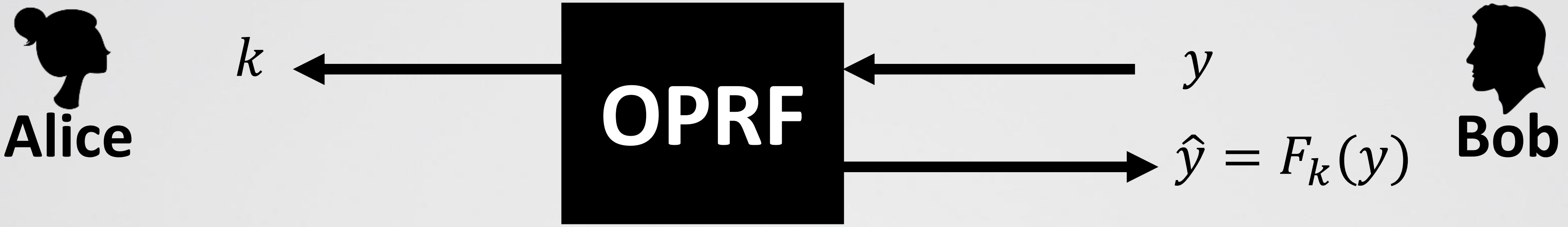


A circuit based PSI protocol [HEK12]

- A circuit that has three steps
  - Sort: Merge two sorted lists using a bitonic merging network [Bat68]. Computes the sorted union using $n\log(2n)$ comparisons.
  - Compare: Compare adjacent items. Uses $2n$ equality checks.
  - Shuffle: Randomly shuffle results using a Waxman permutation network [W68], using $\sim n\log(n)$ swappings.
  - Overall Computes $O(n\log n)$ comparisons.
    Uses $s\cdot(3n\log n + 4n)$ AND gates. (s is input length)

Private Set Intersection (PSI)

Main tool: **Oblivious PRF (OPRF) [FIPR05]**



- To compare $x_1, \dots, x_n$ to y Alice sends:

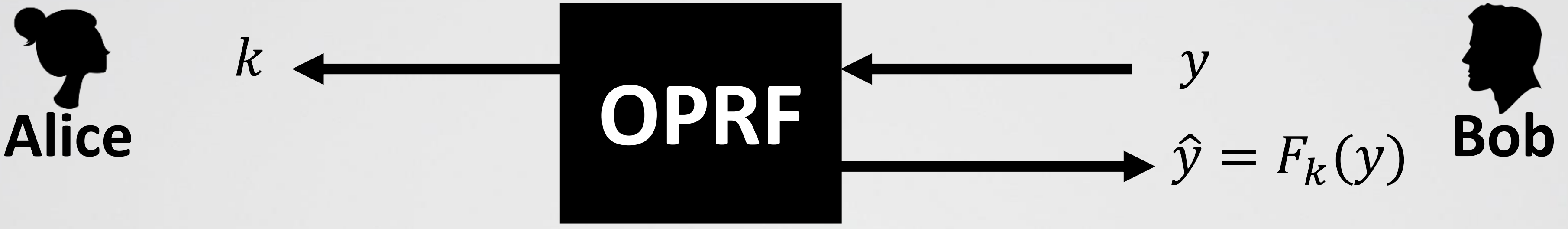
$$\hat{x}_1, \dots, \hat{x}_n = F_k(x_1), \dots F_k(x_n) \longrightarrow$$

If y $\notin \{x_1, \dots, x_n\}$ then it looks random to Bob

$$\hat{x}_1 =? \; \hat{y}$$
$$\hat{x}_2 =? \; \hat{y}$$
$$\dots$$
$$\hat{x}_n =? \; \hat{y}$$

Private Set Intersection (PSI) - Analytics

Main tool: **Oblivious PRF (OPRF) [FIPR05]**

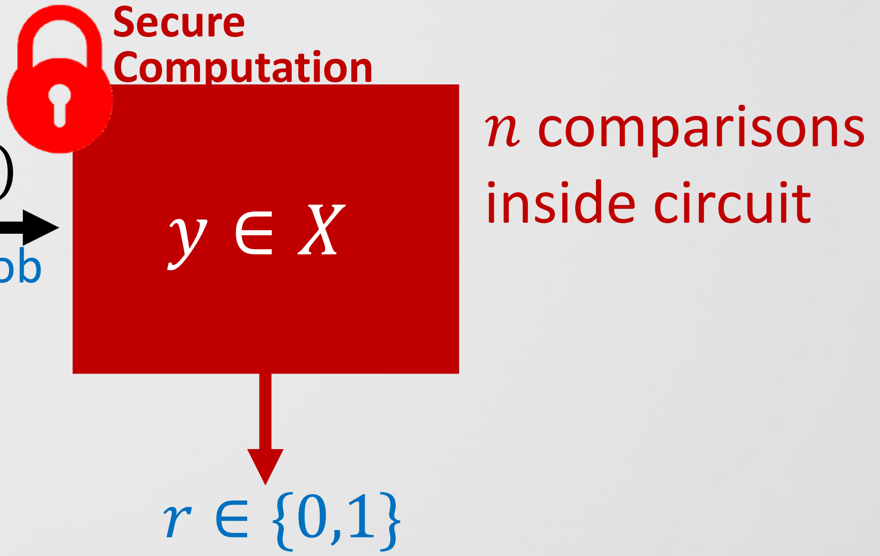


- To compare $x_1, \dots, x_n$ to y Alice sends:

$$\hat{x}_1, \dots, \hat{x}_n = F_k(x_1), \dots F_k(x_n)$$

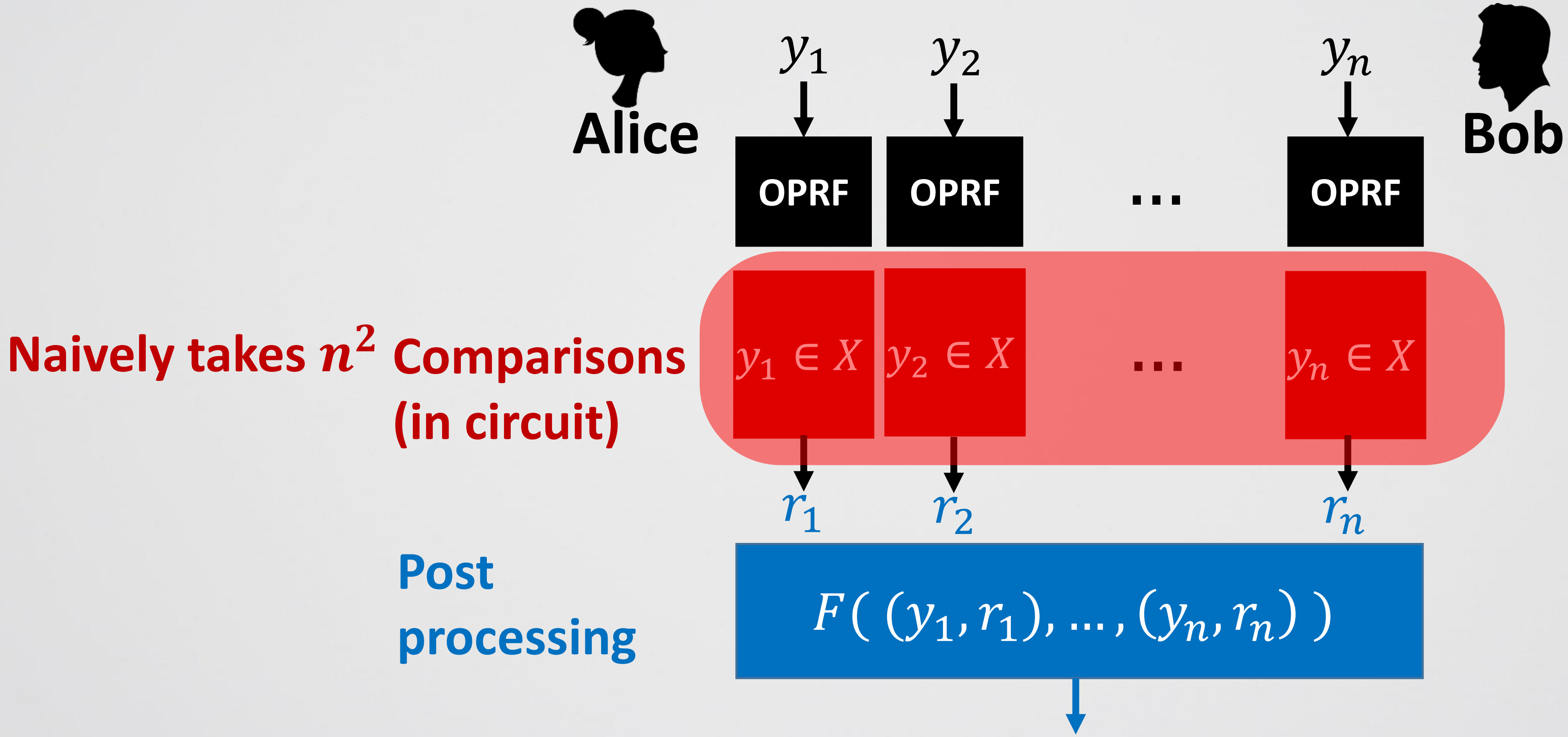
If y $\notin \{x_1, \dots, x_n\}$ then it looks random to Bob

- To compute a function – compare in a 2PC circuit:



Private Set Intersection (PSI) - Analytics

Protocol Overview



Results [PSTY19]

	PSI Analytics	PSI
Asymptotic	First linear-communication protocol (in **OT-hybrid** model and assuming **correlation-robust hash function**)	
Concrete	vs. [PSWW18] 10x less communication 3-6x faster	vs. DH-based 10-20% less communication 7x faster !
Functionality	Payload from both parties	vs. [KKRT16] 40-50% less communication 2-6x slower in LAN (10 Gbps) 2x faster in WAN (10 Mbps)

Cheapest in $ (always)

PSI + Hashing

Map to bins

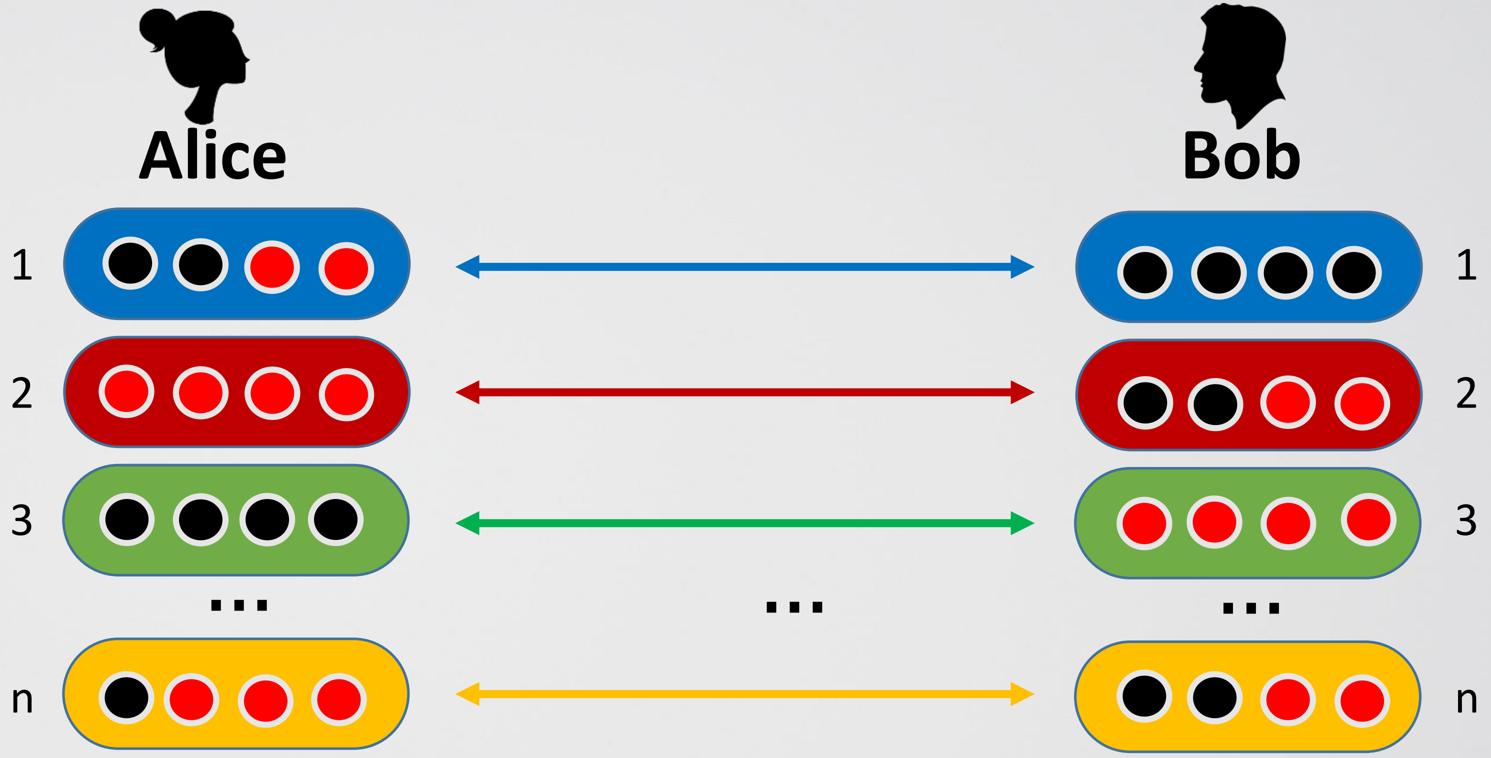
- h: item $\rightarrow$ bin
- Map n items to n bins
  - Some bins may have multiple items

PSI + Hashing

Map to bins

- Use a public h: item $\rightarrow$ bin
- Map n items to n bins
  - Some bins may have multiple items
- Perform bin-wise PSI
- Must hide # items per bin: $(< M = O(\log n)$ w.h.p)
  - Pad bins
- # in-circuit comparisons:

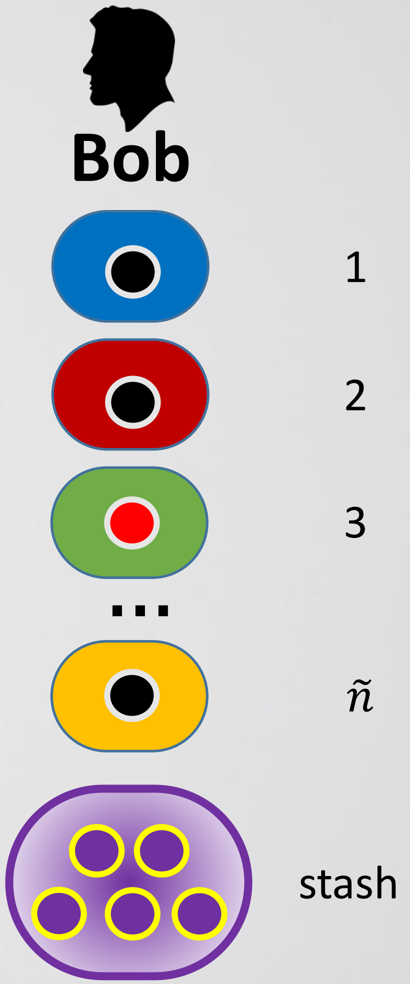
$$n \cdot M^2 = O(n \log^2 n)$$



Using 2 Hash Functions (Cuckoo hashing [PR,KMW])

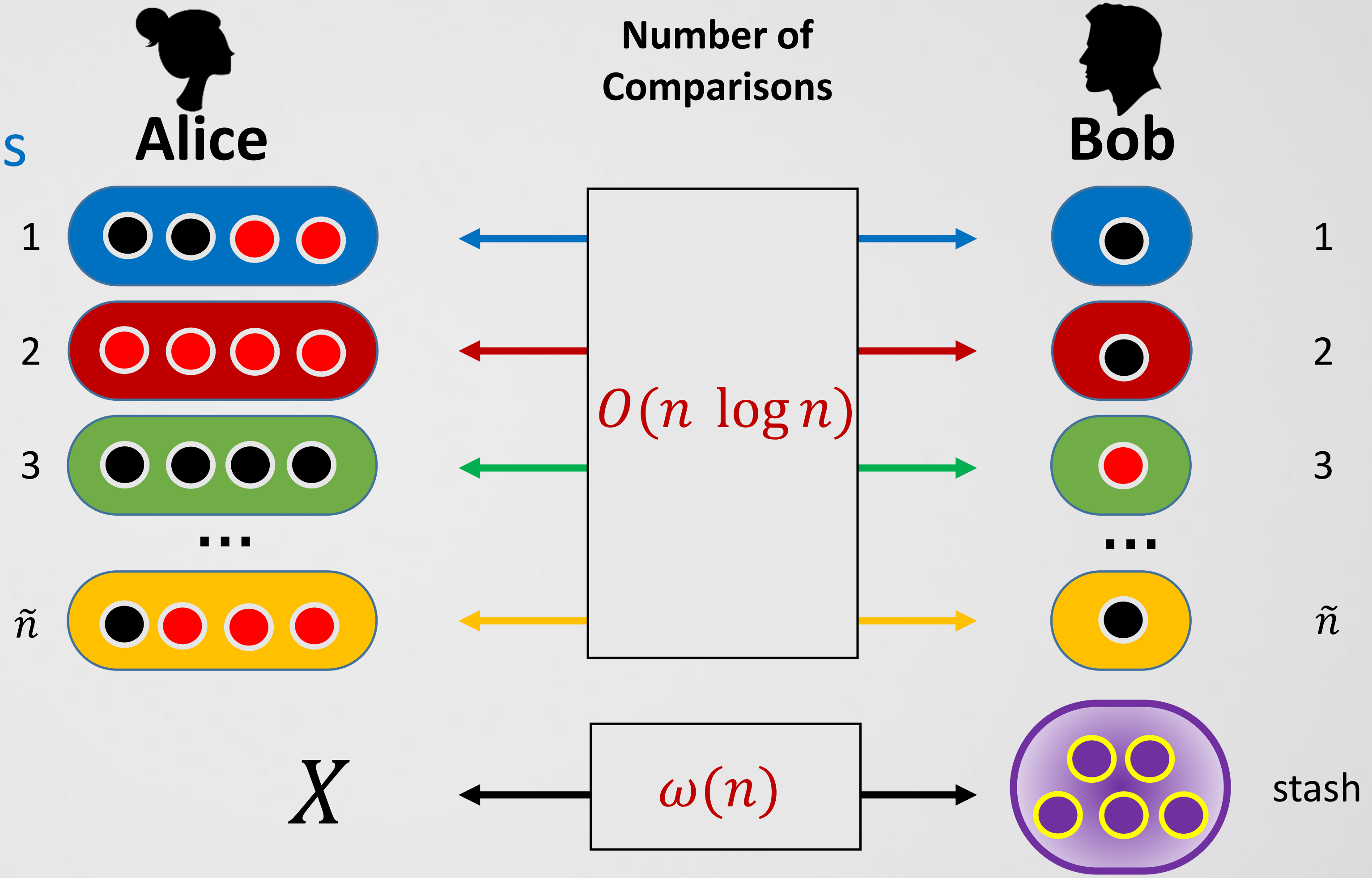
- h_1, h_2: item $\rightarrow$ bin
- Map n items to $(2+\epsilon)n$ bins
- Each bin can store at most one item!

- Succeeds with very high probability
- If we also have a stash of size s, all items *x* can be mapped to either *$h_1(x), h_2(x)$* or the stash, except with probability *$n^{-(s+1)}$*.



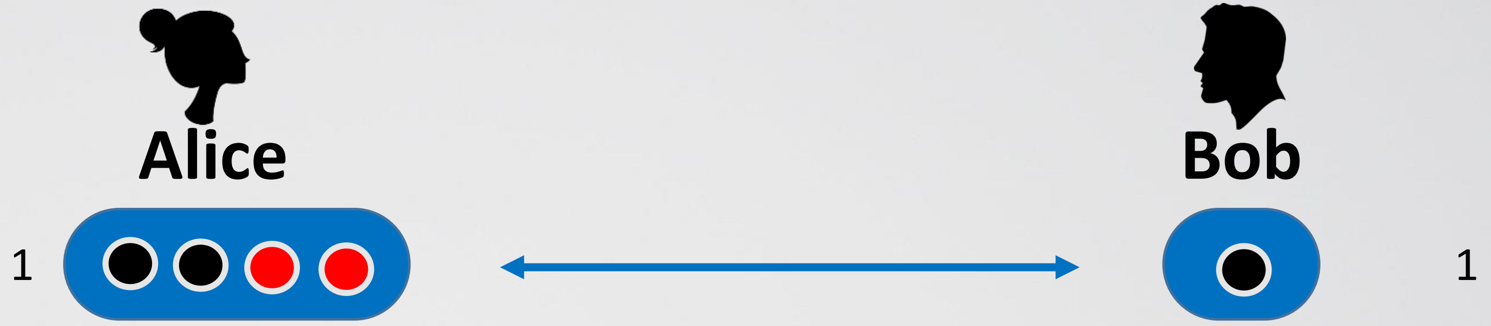
The Power of Using 2 Hash Functions (Cuckoo)

- h_1, h_2: item $\rightarrow$ bin
- Map n items to $(2+\epsilon)n$ bins
  - Alice – simple hashing
    - $x \rightarrow h_1(x)$ **and** $h_2(x)$
    - Max < $M = O(\log n)$
  - Bob – Cuckoo hashing
    - $y \rightarrow h_1(y)$, **or** $h_2(y)$
    - Max ≤ 1
- **Caveat:** stash size $\omega(1)$



Our Protocol

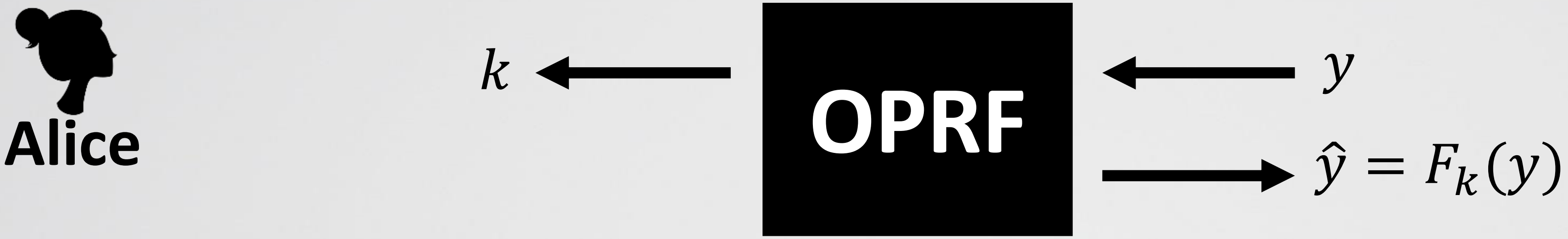
1st step



- **Recall:** each bin contributes $O(\log n)$ comparisons to the circuit
- We transform it to **only 1 comparison** using **Oblivious *Programmable* PRF**

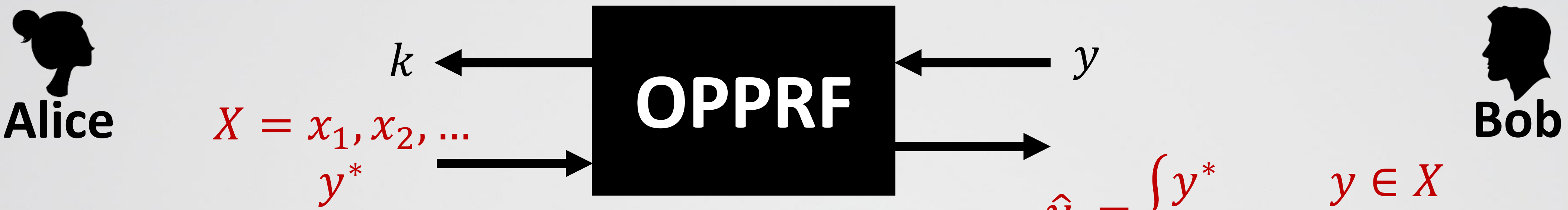
Our Protocol

Oblivious *Programmable* PRF (OPPRF) [KMPRT17]



Our Protocol

Oblivious *Programmable* PRF (OPPRF) [KMPRT17]



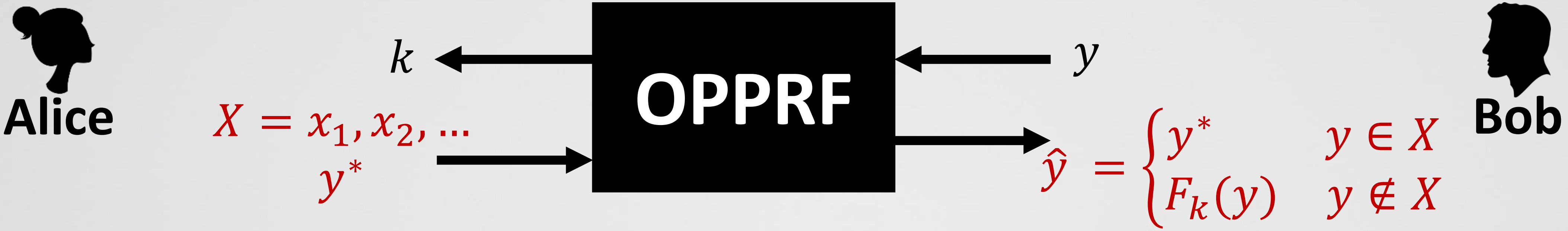
$$\hat{y} = \begin{cases} y^* & y \in X \\ F_k(y) & y \notin X \end{cases}$$

- To compare $x_1, x_2, \dots$ to y Alice sends:

$$\xrightarrow{\quad y^* \quad} \quad y^* =? \, \hat{y}$$

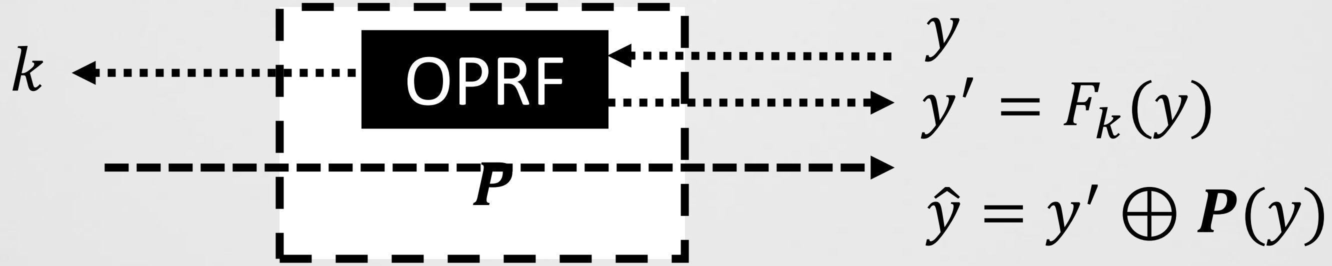
Our Protocol

Implementing Oblivious *Programmable* PRF (OPPRF) [KMPRT17]



$\hat{x}_i = F_k(x_i) \oplus y^*$

Interpolate $\boldsymbol{P}: x_i \to \hat{x}_i$



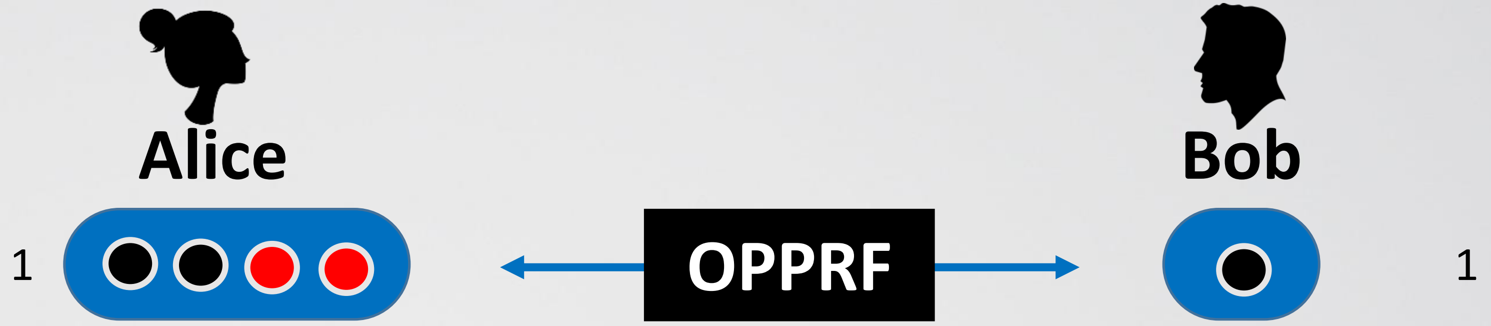
$$y = x_i \Rightarrow \hat{y} = F_k(x_i) \oplus \boldsymbol{P}(x_i) = F_k(x_i) \oplus F_k(x_i) \oplus y^* = y^*$$

$$y \neq x_i \Rightarrow \hat{y} = F_k(y) \oplus \boldsymbol{P}(y)$$

Our Protocol

1st step: "Programming" the PRF

- Alice "programs" $O(\log n)$ items
- ➔ Single comparison in the secure computation



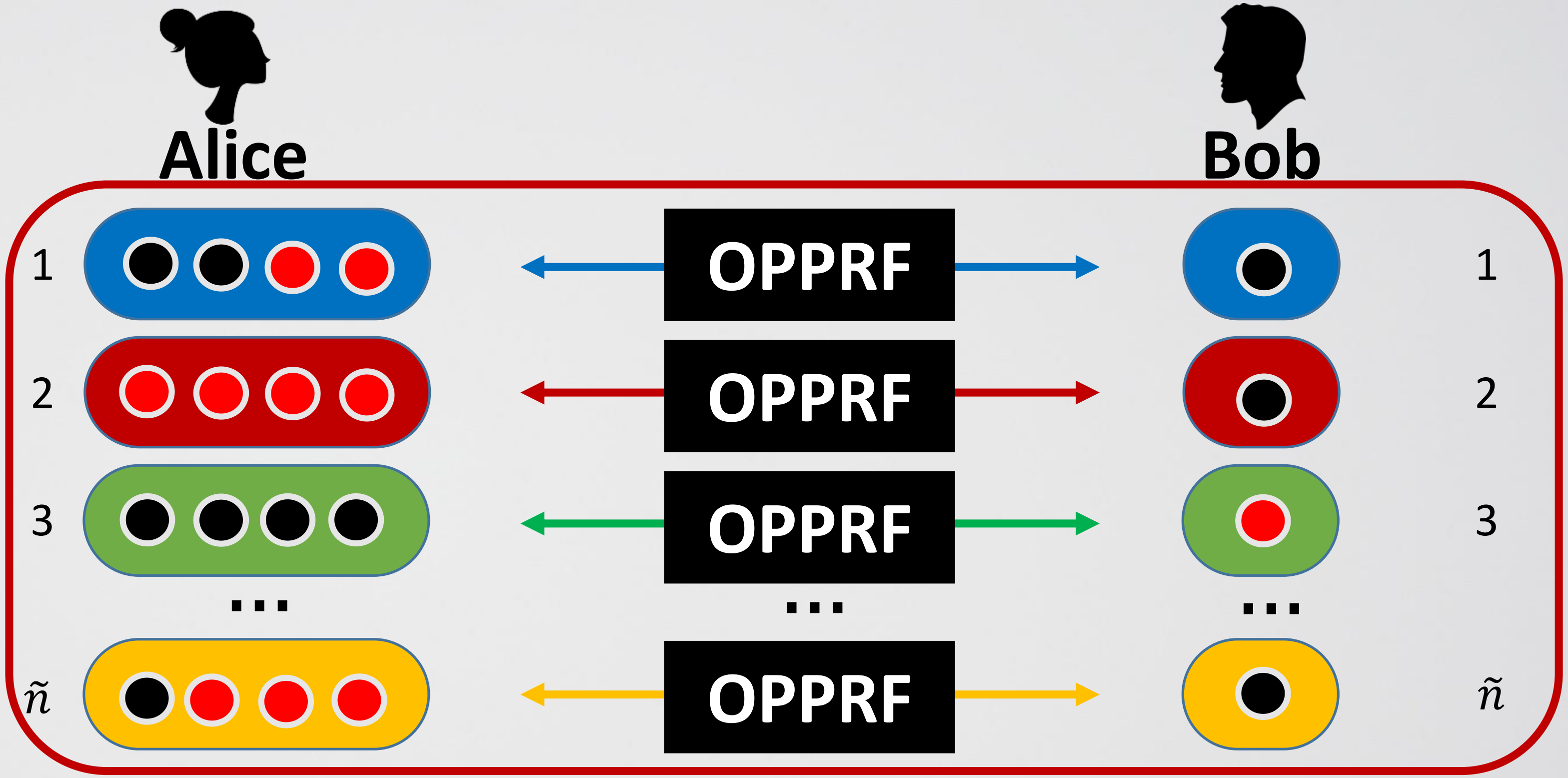
- **Communication of OPPRF:**
  - For each bin, linear in the number of programmed values
  - ➔ communication per bin remains $O(\log n)$

Our Protocol

2nd Step: Batching the OPPRF

- We can "batch" many OPPRFs with comm. $O(n)$
- **Preserving obliviousness without programming padded values!**

$O(n)$



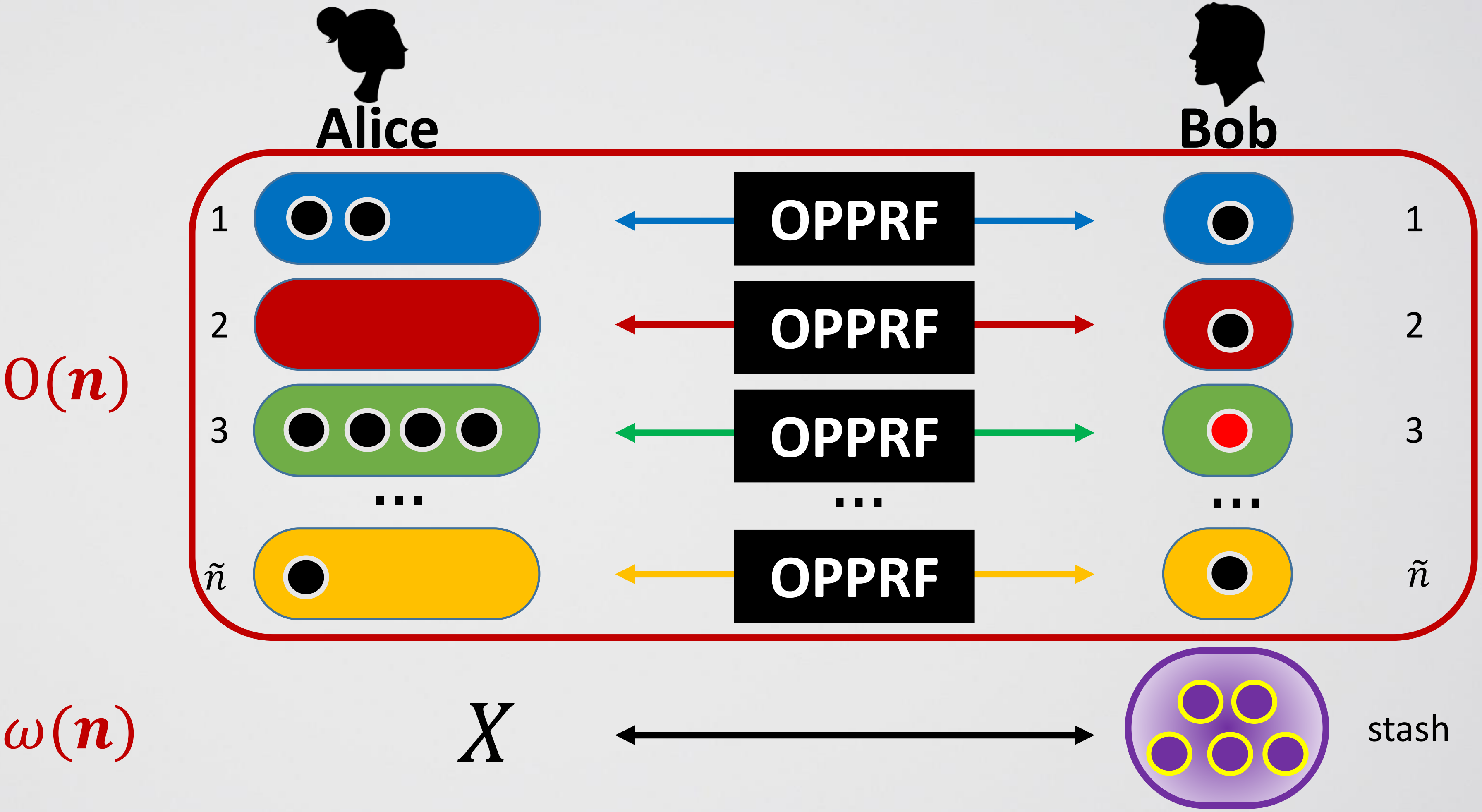
Using high degree polynomials

- Need to interpolate very high degree polynomials over arbitrary points
- Lagrange interpolation is too slow

- **Used FFT to do that with overhead $O(n\log^2 n)$**

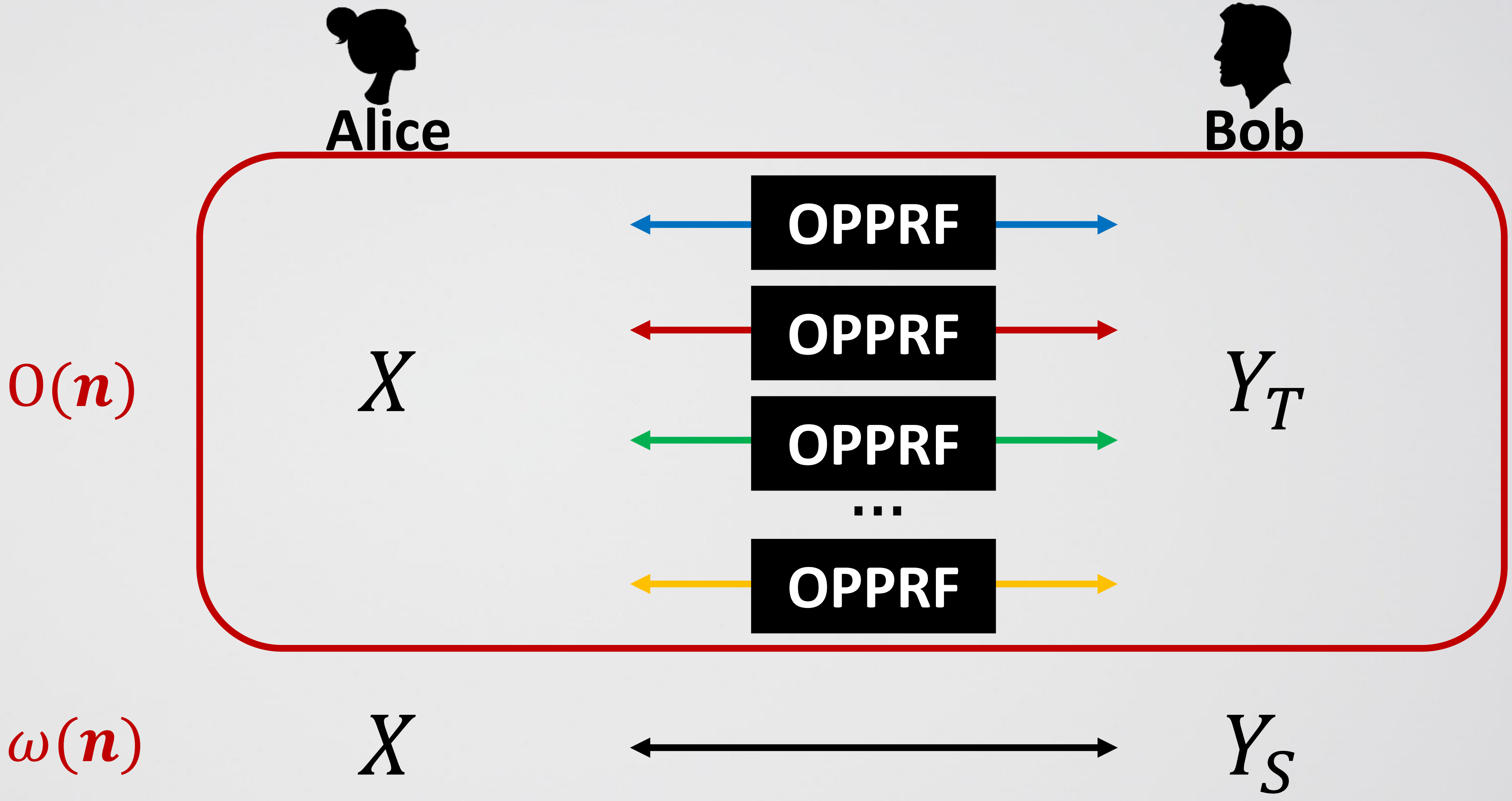
Our Protocol

3rd Step: Handling the Stash



Our Protocol

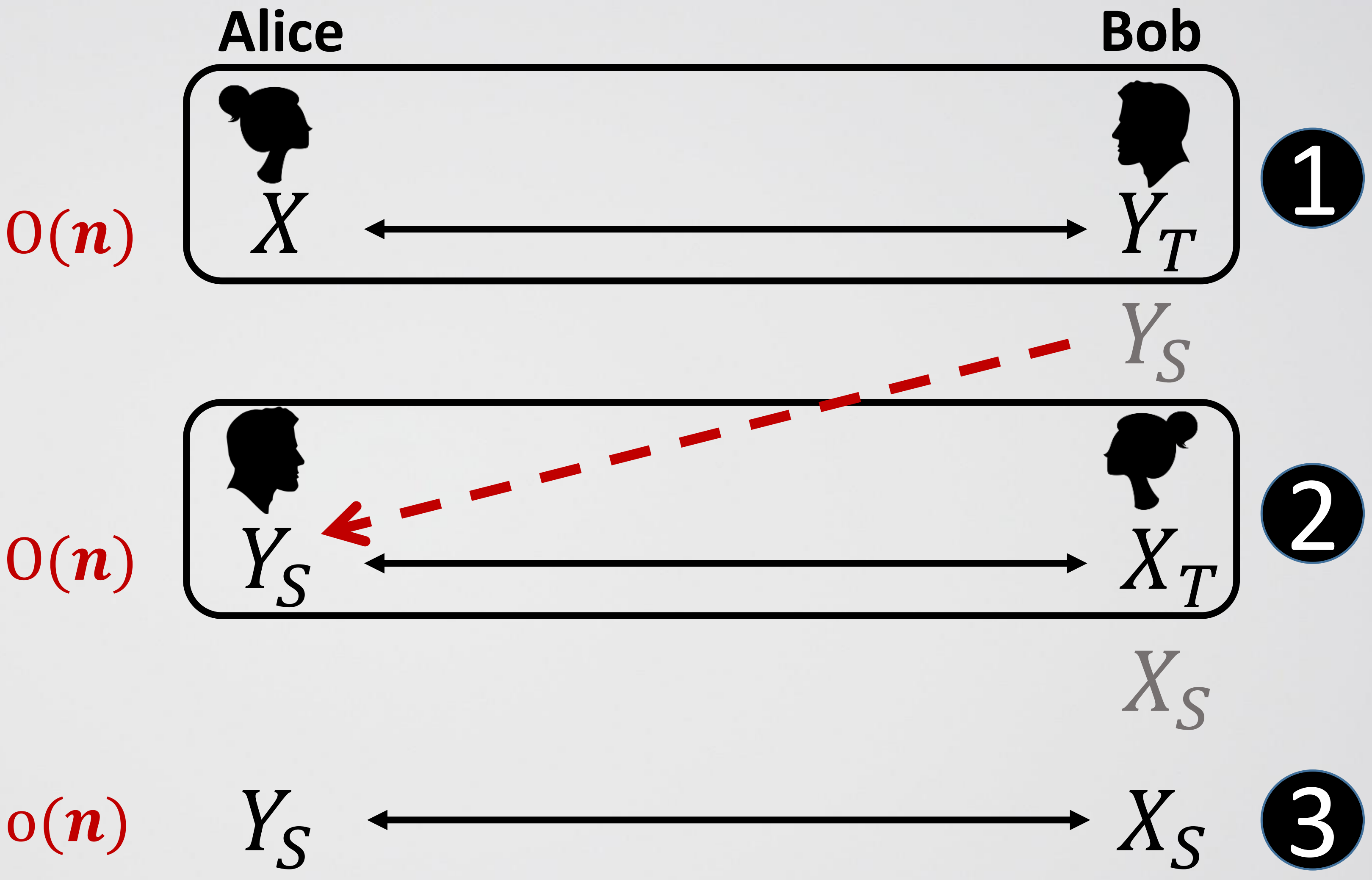
3rd Step: Handling the Stash



Dual Execution

3rd Step: Handling the Stash

- 3 phases protocol:



Experiments – PSI Analytics

vs. Previous Circuit-Based PSI [PSWW18]

$n = 2^{20}$ items of arbitrary bit-length

Fixing failure probability to 2^{-40}

	[HEK12]	**[PSWW18]**	**This work**
Communication	106 GB	25 GB	2.5 GB
Runtime (LAN)		5.5 min	2 min
Runtime (WAN)		25 min	4.5 min

vs. PSI-SUM [IKN+17]

65x faster (in LAN)

They leak intersection size

Conclusions

- MPC can help in getting rid of trusted parties
- Generic MPC is efficient if circuit size is small

- PSI in an important and interesting primitive, for which a naïve circuit is too large
- For such problems, need to design specific but adaptive MPC protocols