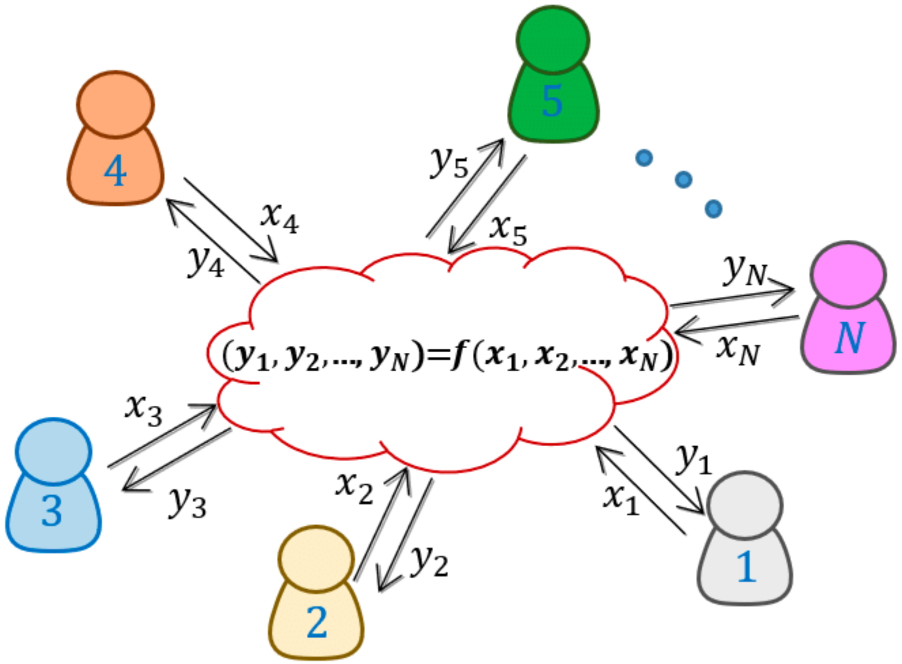
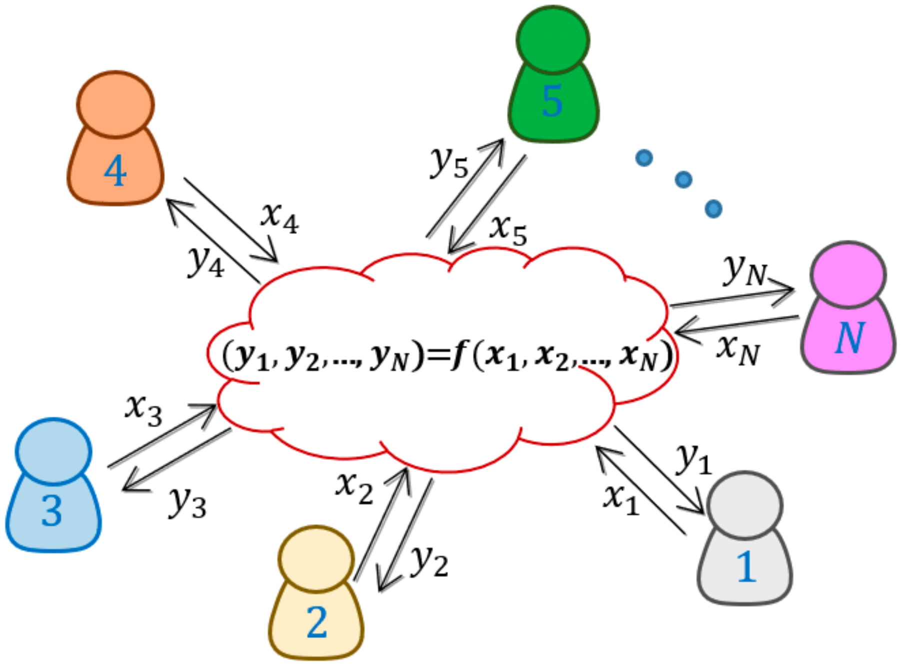
Private set intersection (PSI)
from pseudorandom correlation generator (PCG)

Geoffroy Couteau, **Dung Bui**

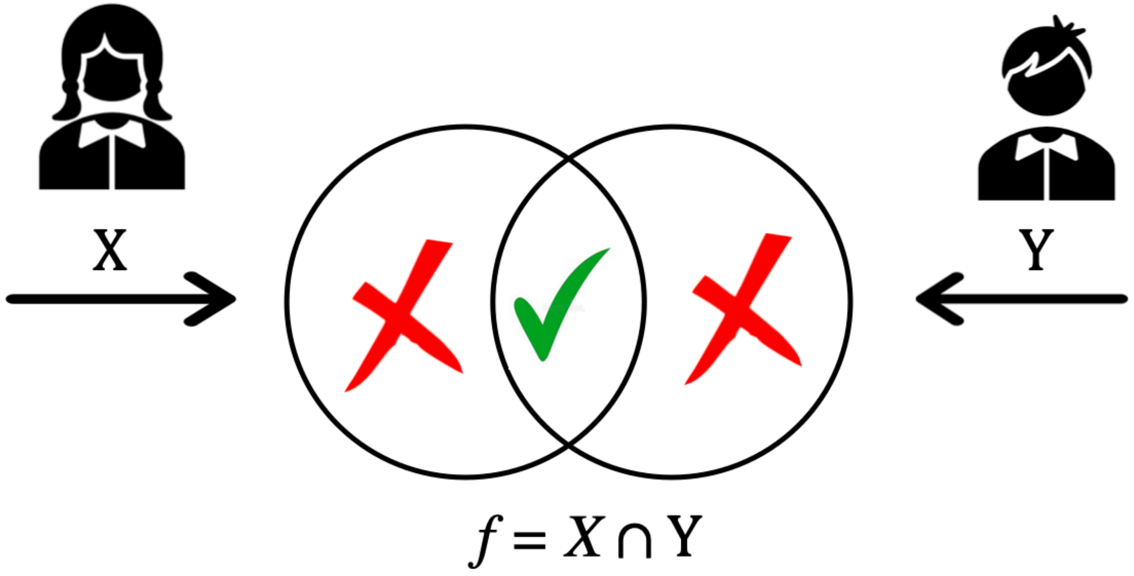
Secure computation (MPC)



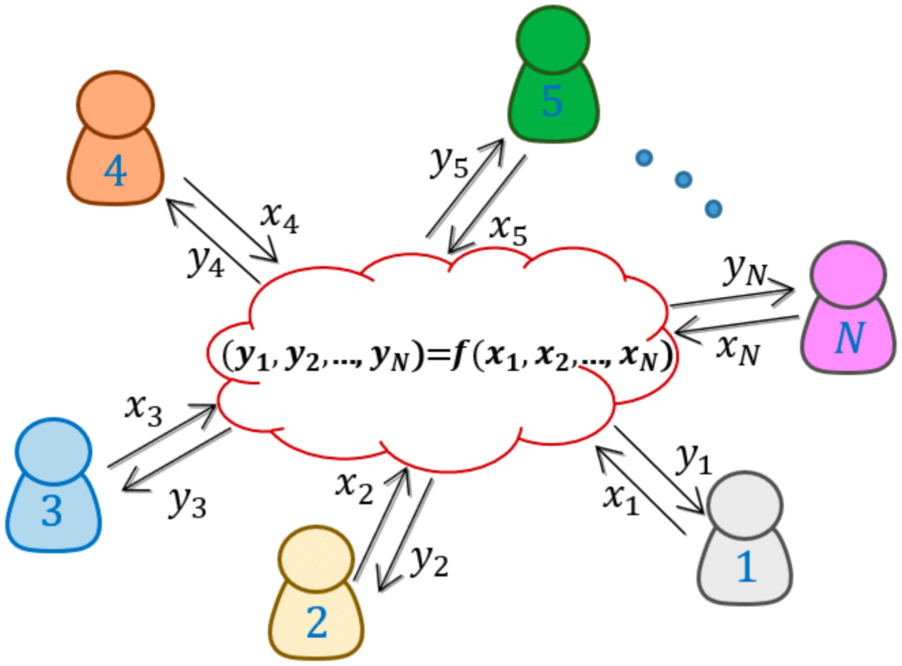
Secure computation (MPC)



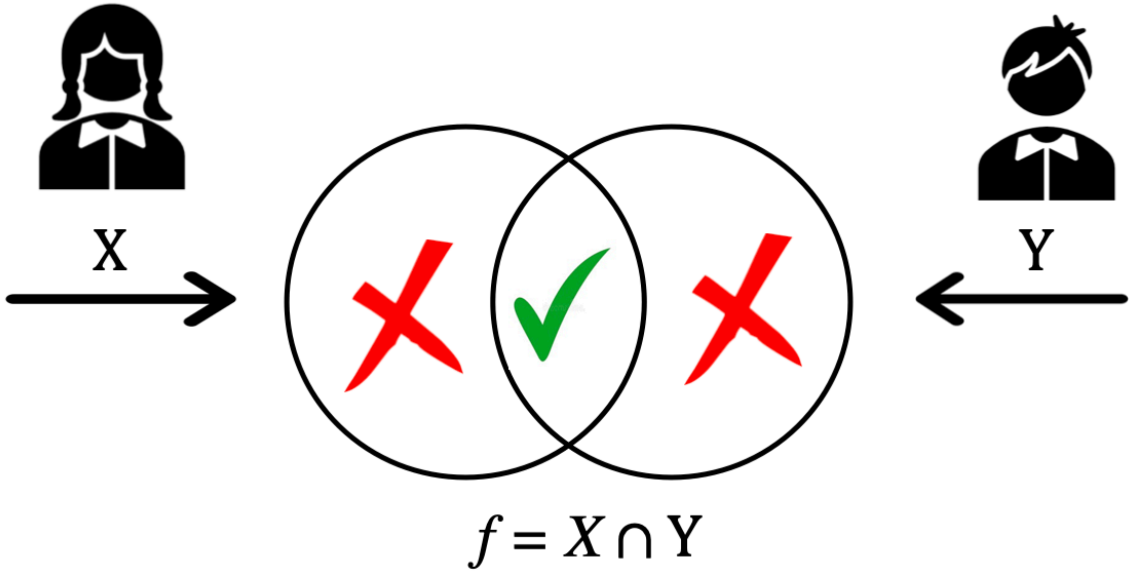
Private Set Intersection (PSI)



Secure computation (MPC)

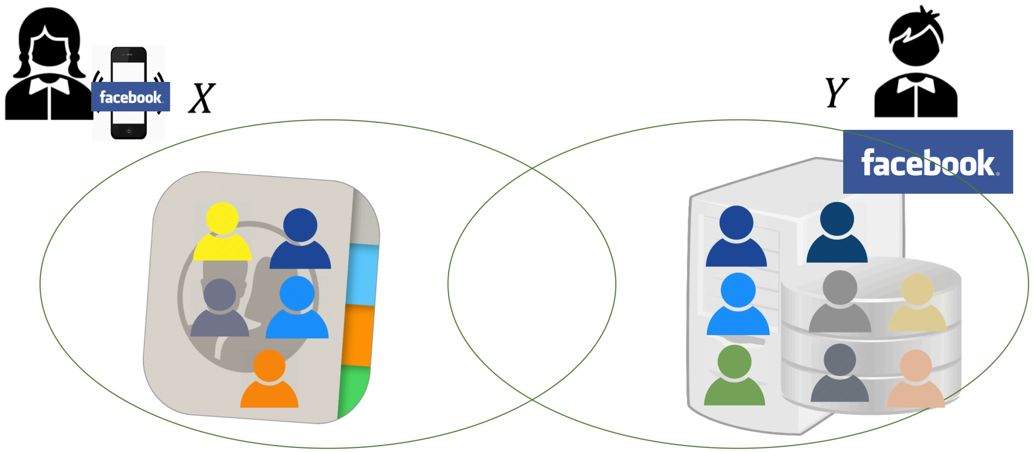


Private Set Intersection (PSI)

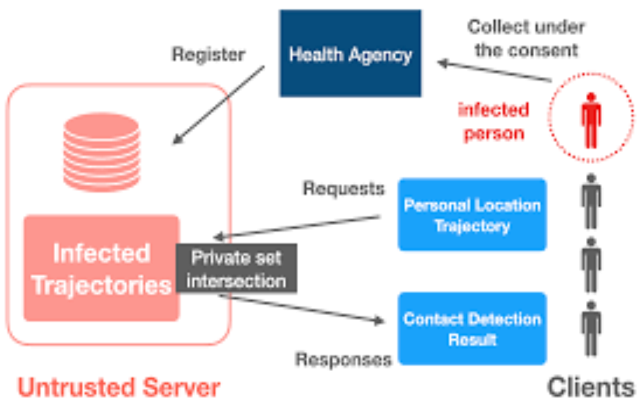


Adversary types:

- Semi-honest : following the protocol
- Malicious: may deviate from the protocol



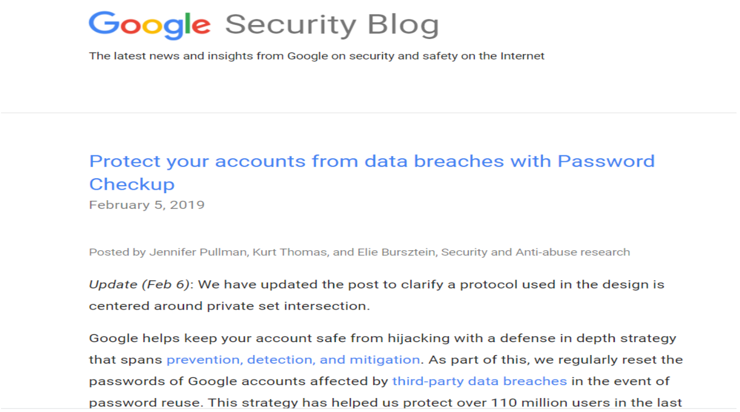
Contact Discovery



Contact Tracing

PSI

Ad Efficiency



Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Protect your accounts from data breaches with Password Checkup

February 5, 2019

Posted by Jennifer Pullman, Kurt Thomas, and Elie Bursztein, Security and Anti-abuse research

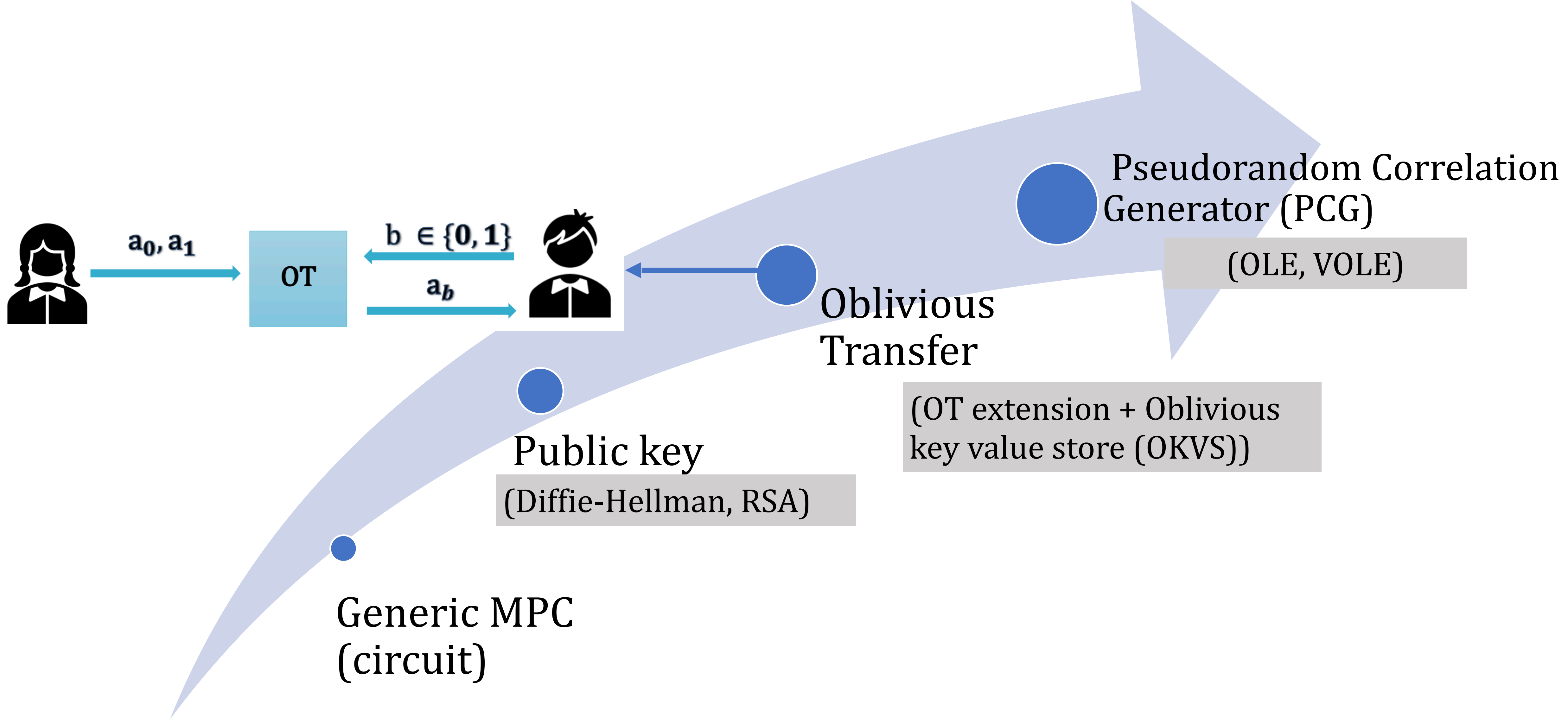
Update (Feb 6): We have updated the post to clarify a protocol used in the design is centered around private set intersection.

Google helps keep your account safe from hijacking with a defense in depth strategy that spans prevention, detection, and mitigation. As part of this, we regularly reset the passwords of Google accounts affected by third-party data breaches in the event of password reuse. This strategy has helped us protect over 110 million users in the last

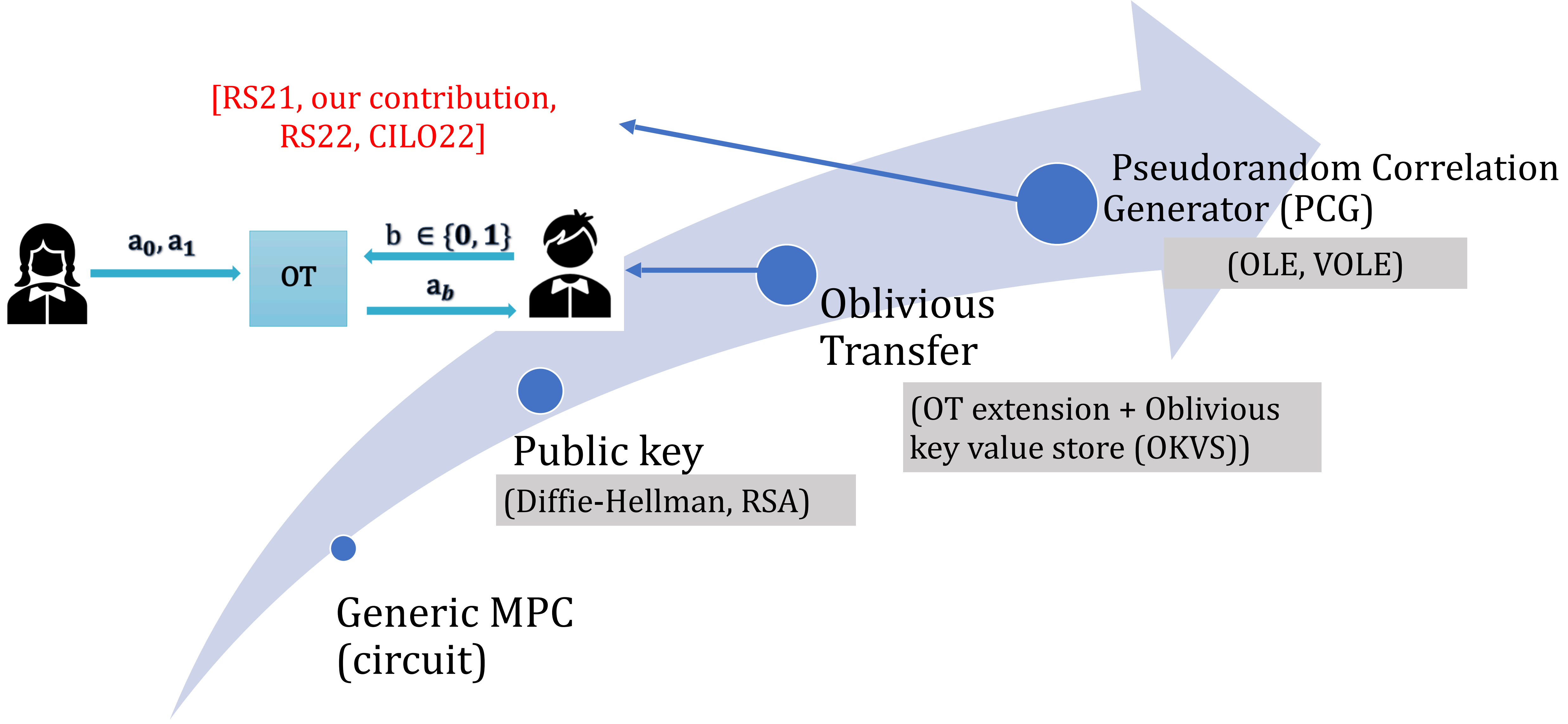
Google Password Checkup

Paradigms for PSI

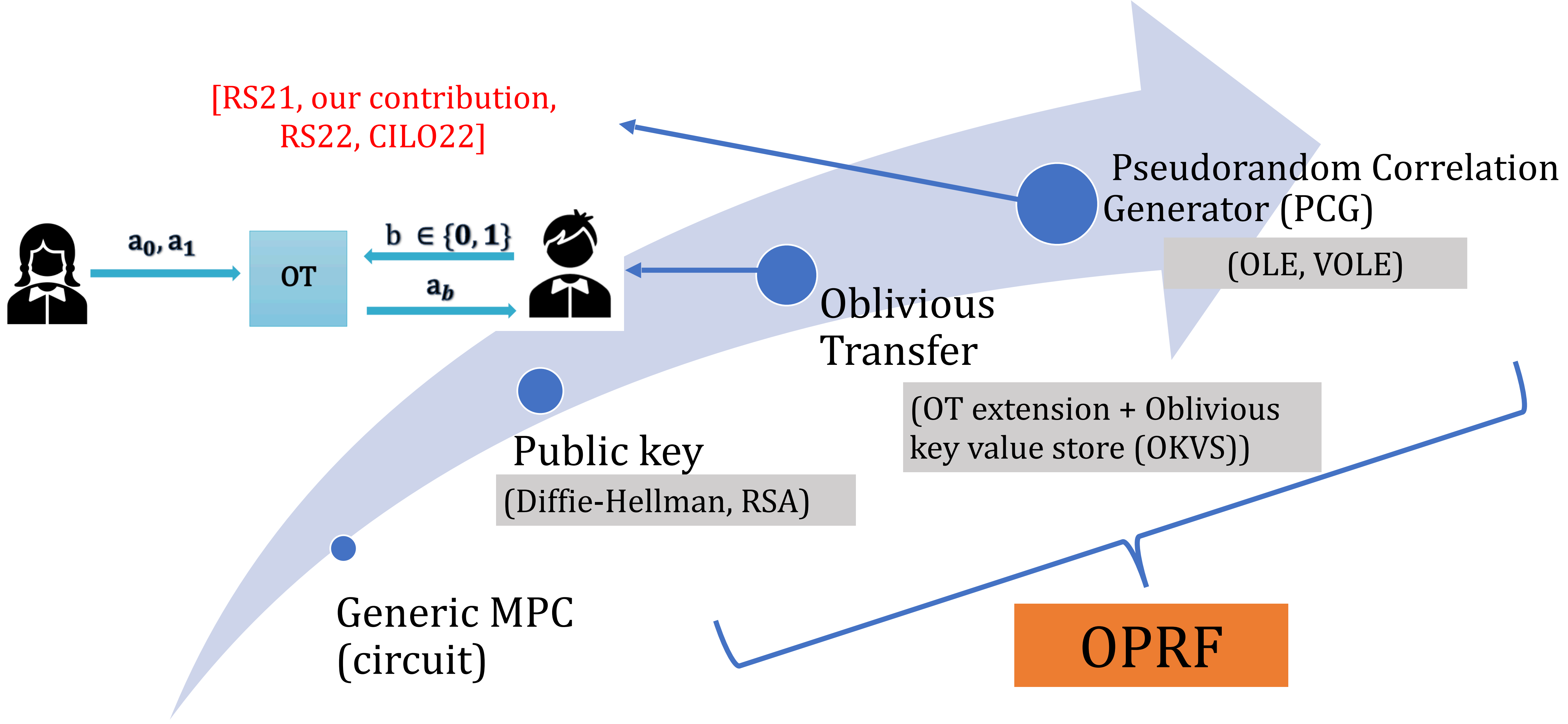
Paradigms for PSI



Paradigms for PSI



Paradigms for PSI



Oblivious PRF (OPRF)

Oblivious PRF (OPRF)

- A PRF is a keyed function $F_k()$

If k is random and kept secret then a polynomial-time adversary can not distinguish between the output of $F_k()$ and a truly random function

Oblivious PRF (OPRF)

- A PRF is a keyed function $F_k()$

If k is random and kept secret then a polynomial-time adversary can not distinguish between the output of $F_k()$ and a truly random function

PSI → Oblivious PRF (OPRF)

Oblivious PRF (OPRF)

- A PRF is a keyed function $F_k()$

If k is random and kept secret then a polynomial-time adversary can not distinguish between the output of $F_k()$ and a truly random function

PSI → Oblivious PRF (OPRF)

PSI

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

$Y = \{y_1, y_2, \ldots, y_n\}$

Oblivious PRF (OPRF)

- A PRF is a keyed function $F_k()$

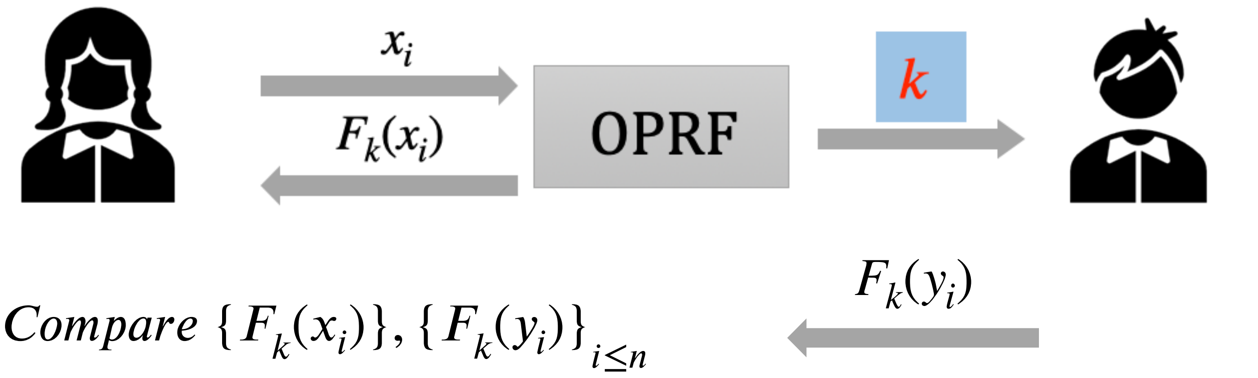
If k is random and kept secret then a polynomial-time adversary can not distinguish between the output of $F_k()$ and a truly random function

PSI → Oblivious PRF (OPRF)

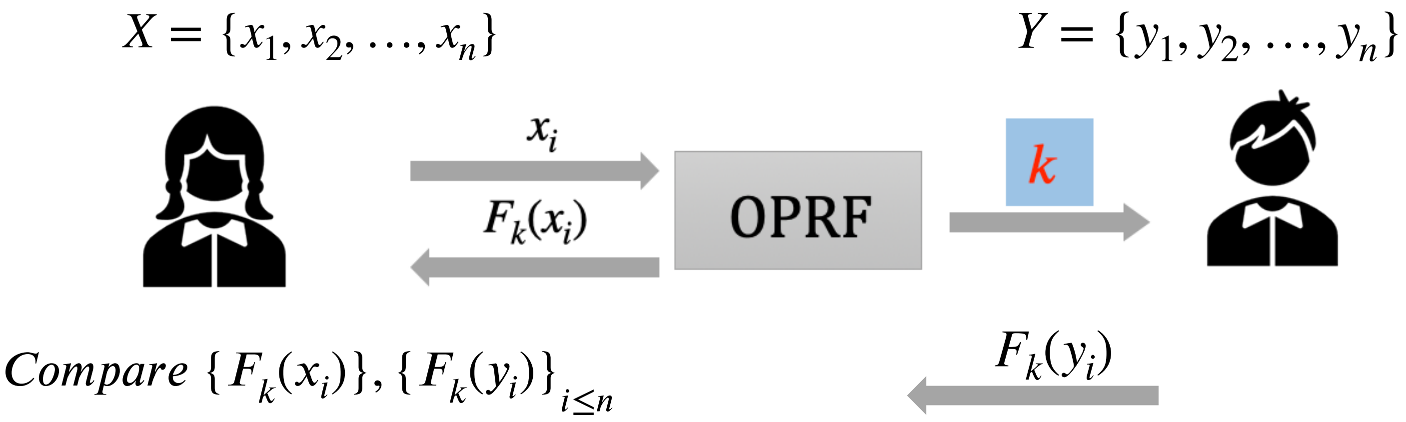
PSI

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

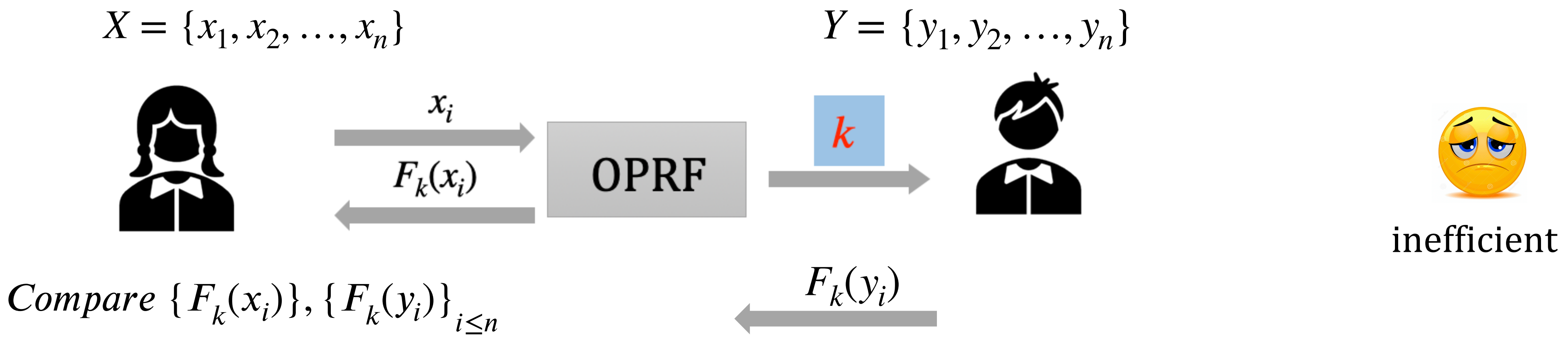
$Y = \{y_1, y_2, \ldots, y_n\}$



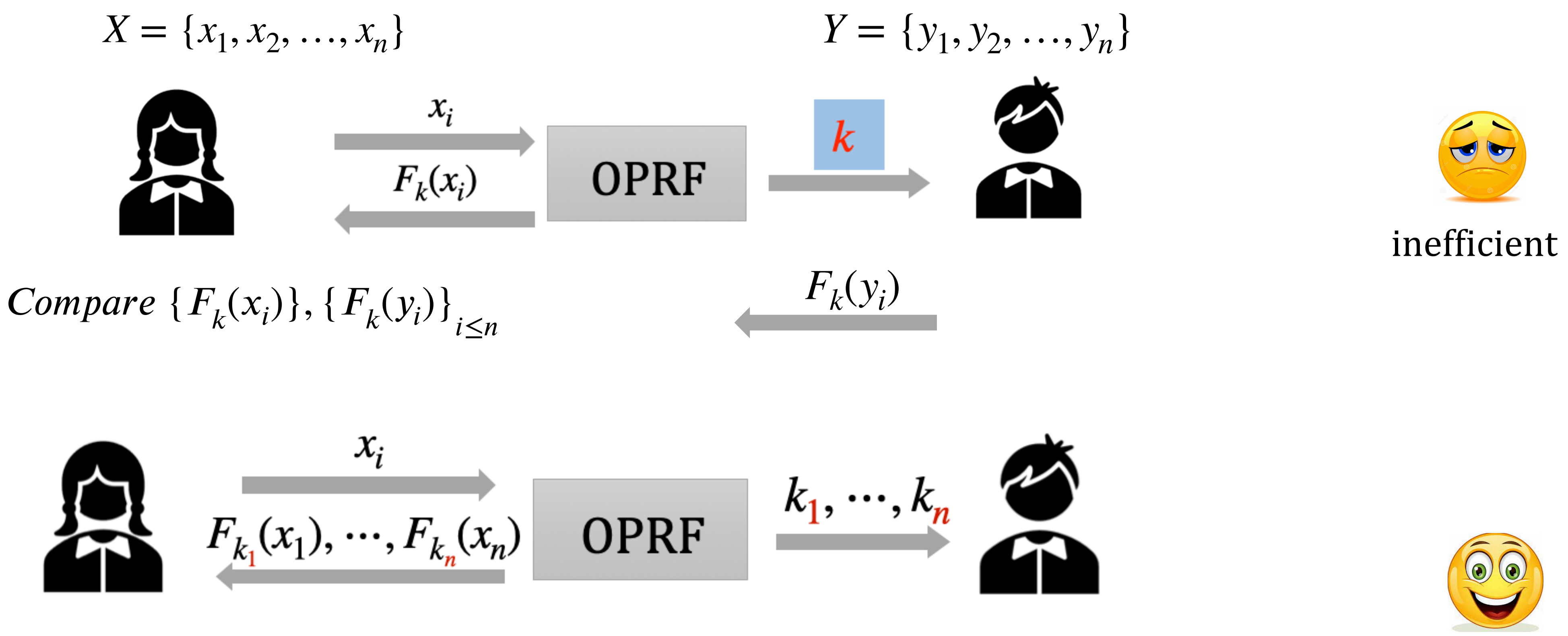
PSI from OPRF



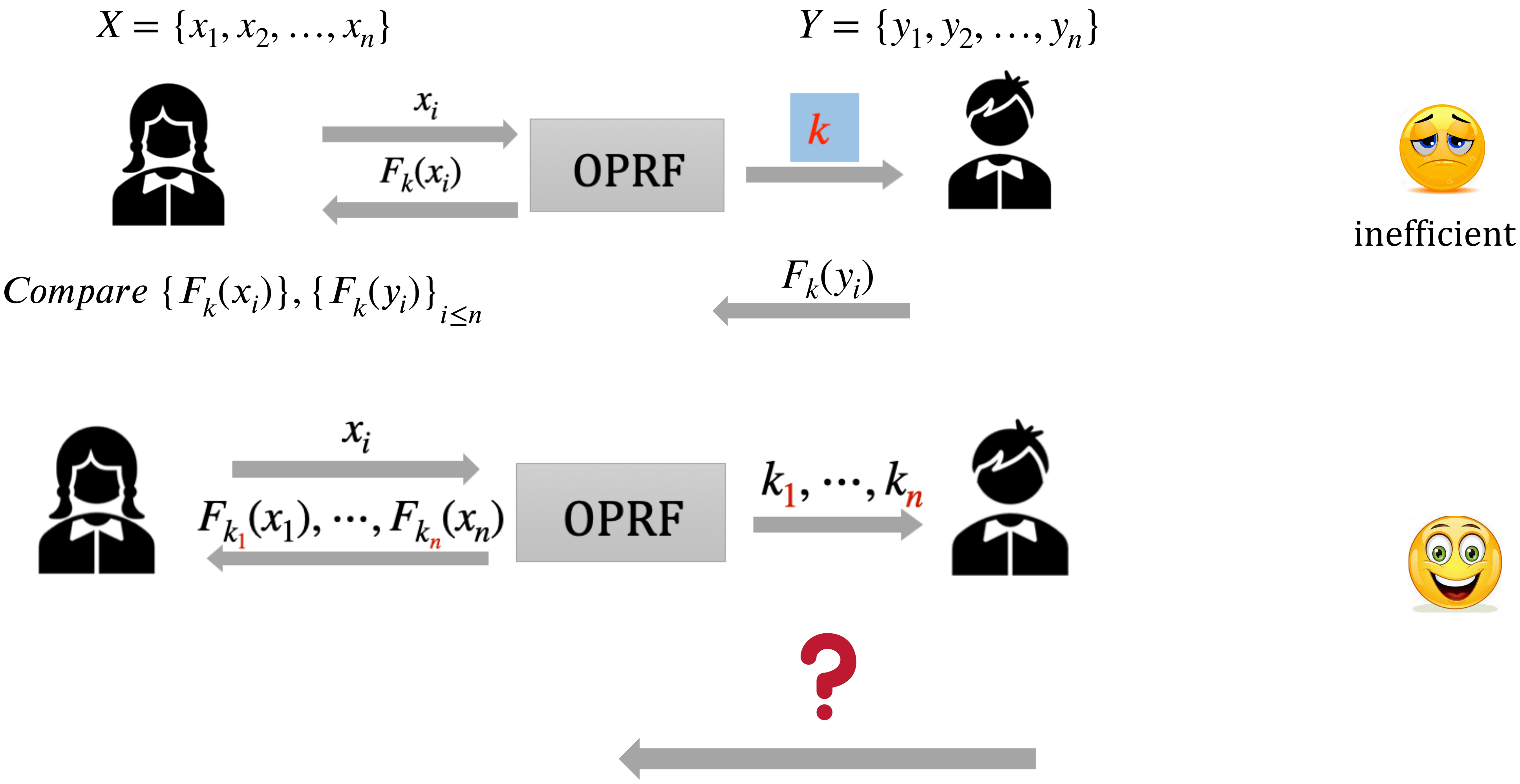
PSI from OPRF



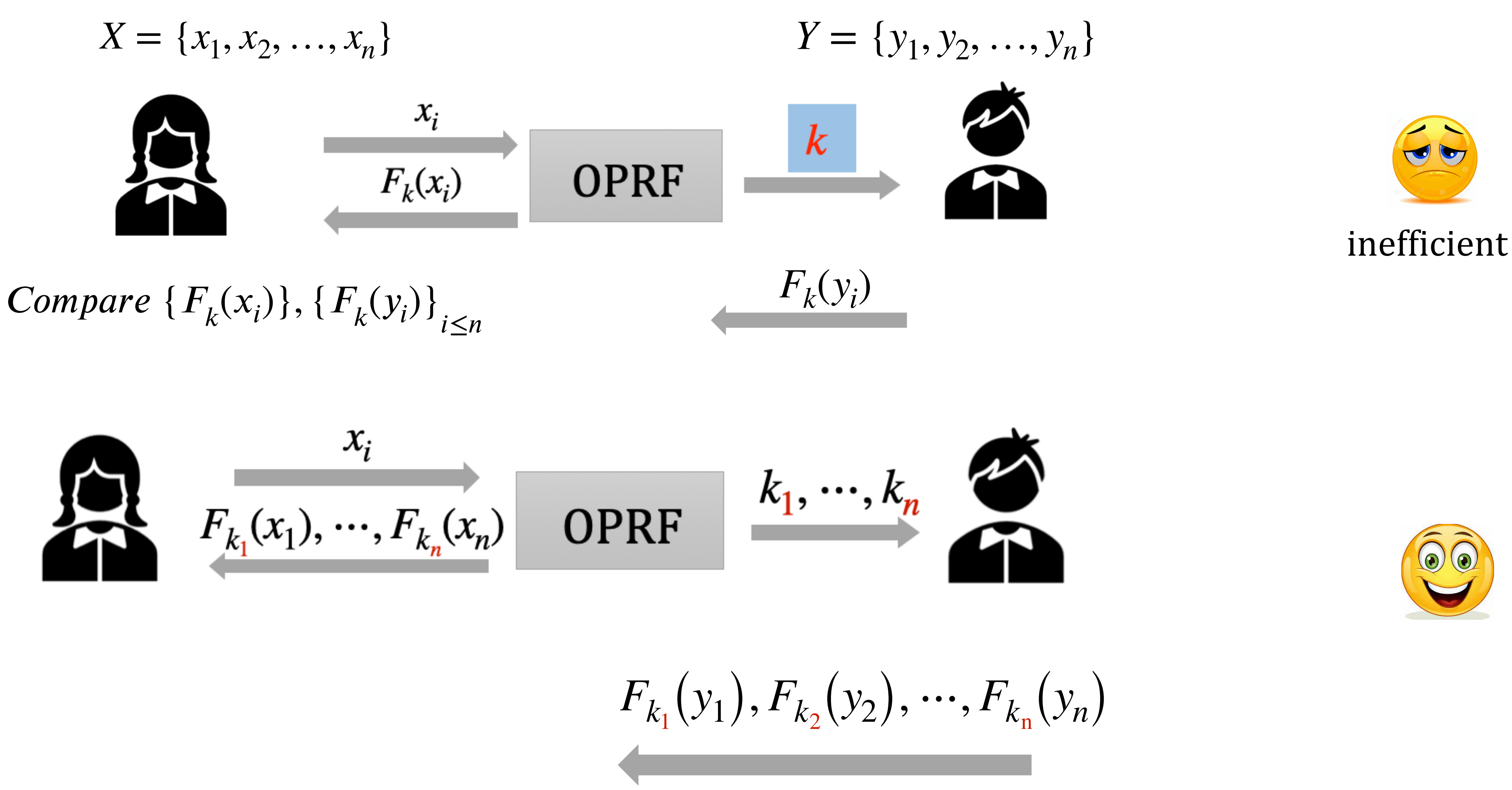
PSI from OPRF



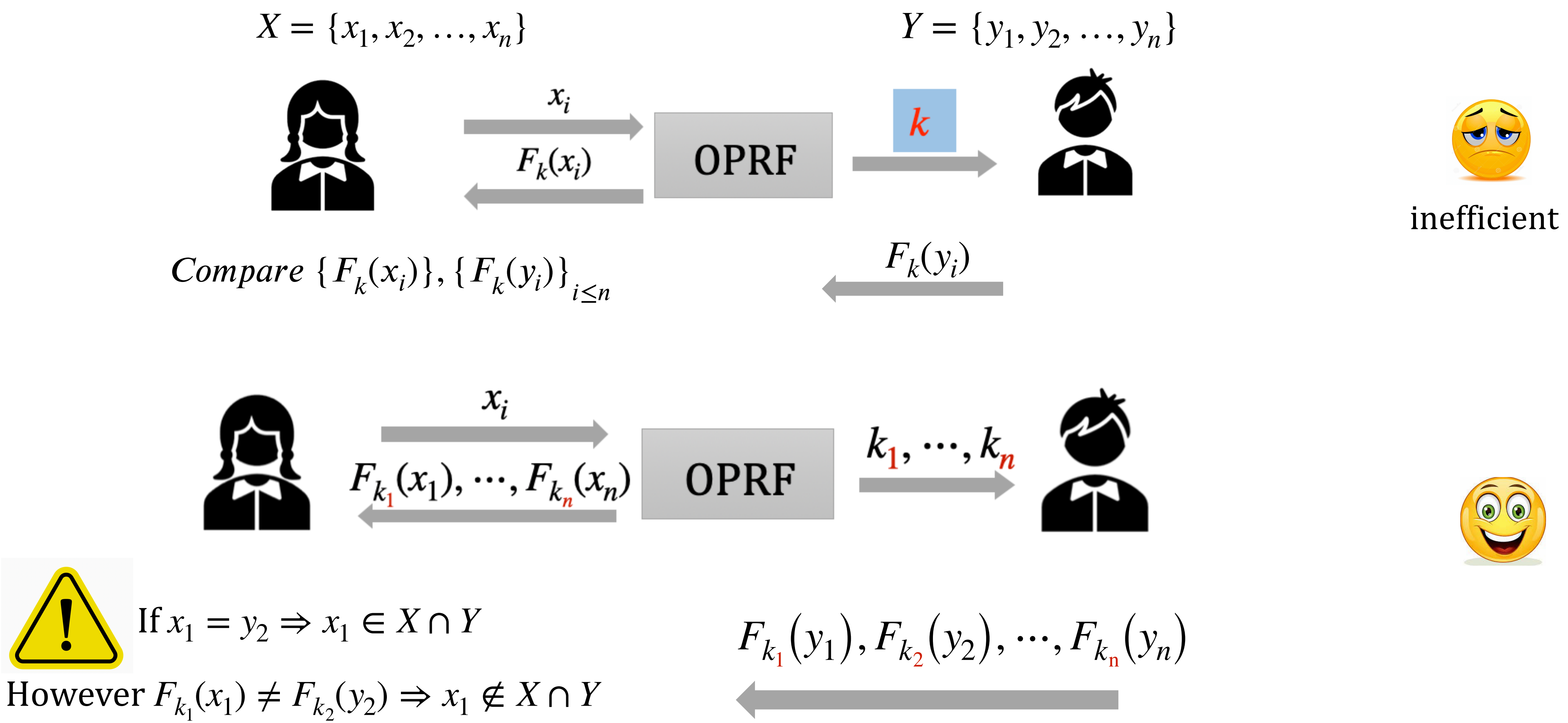
PSI from OPRF



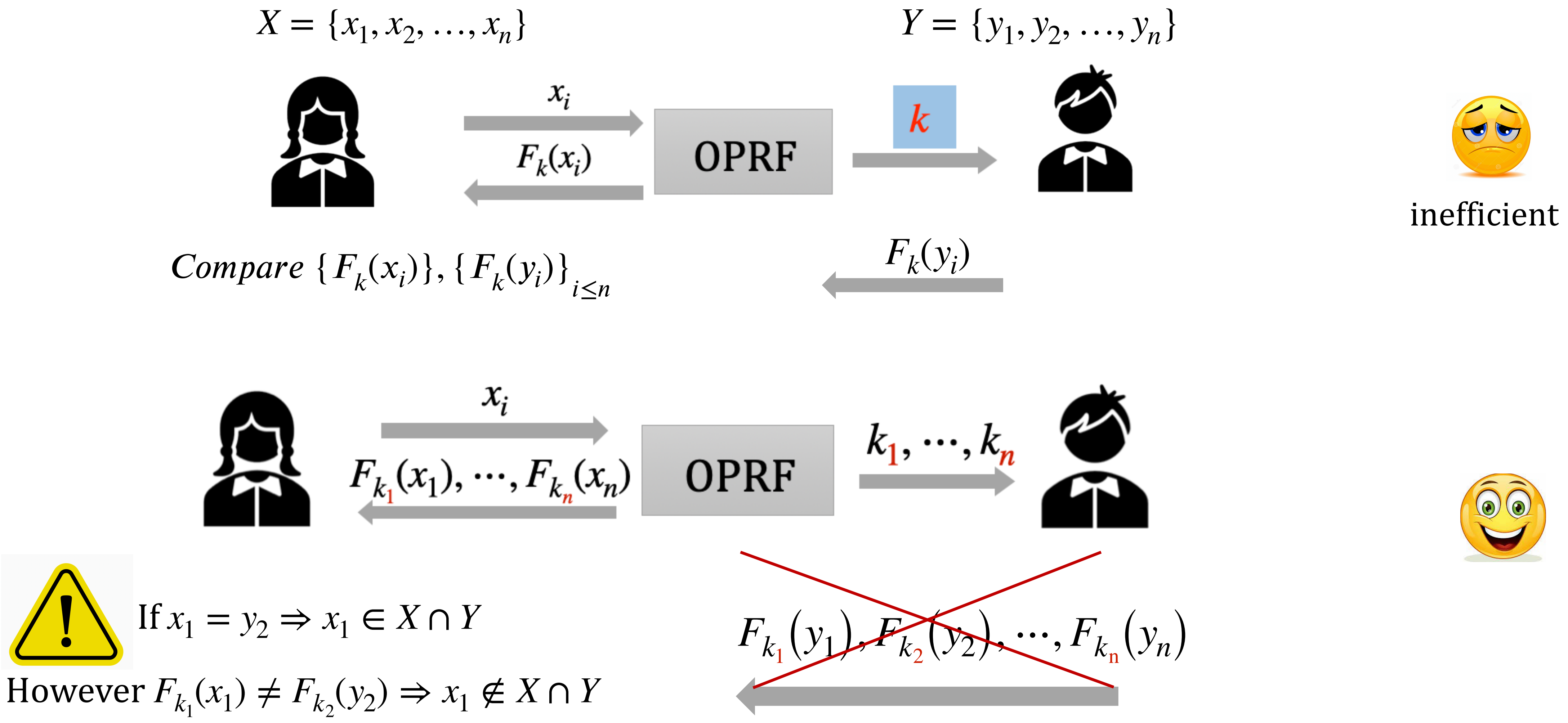
PSI from OPRF



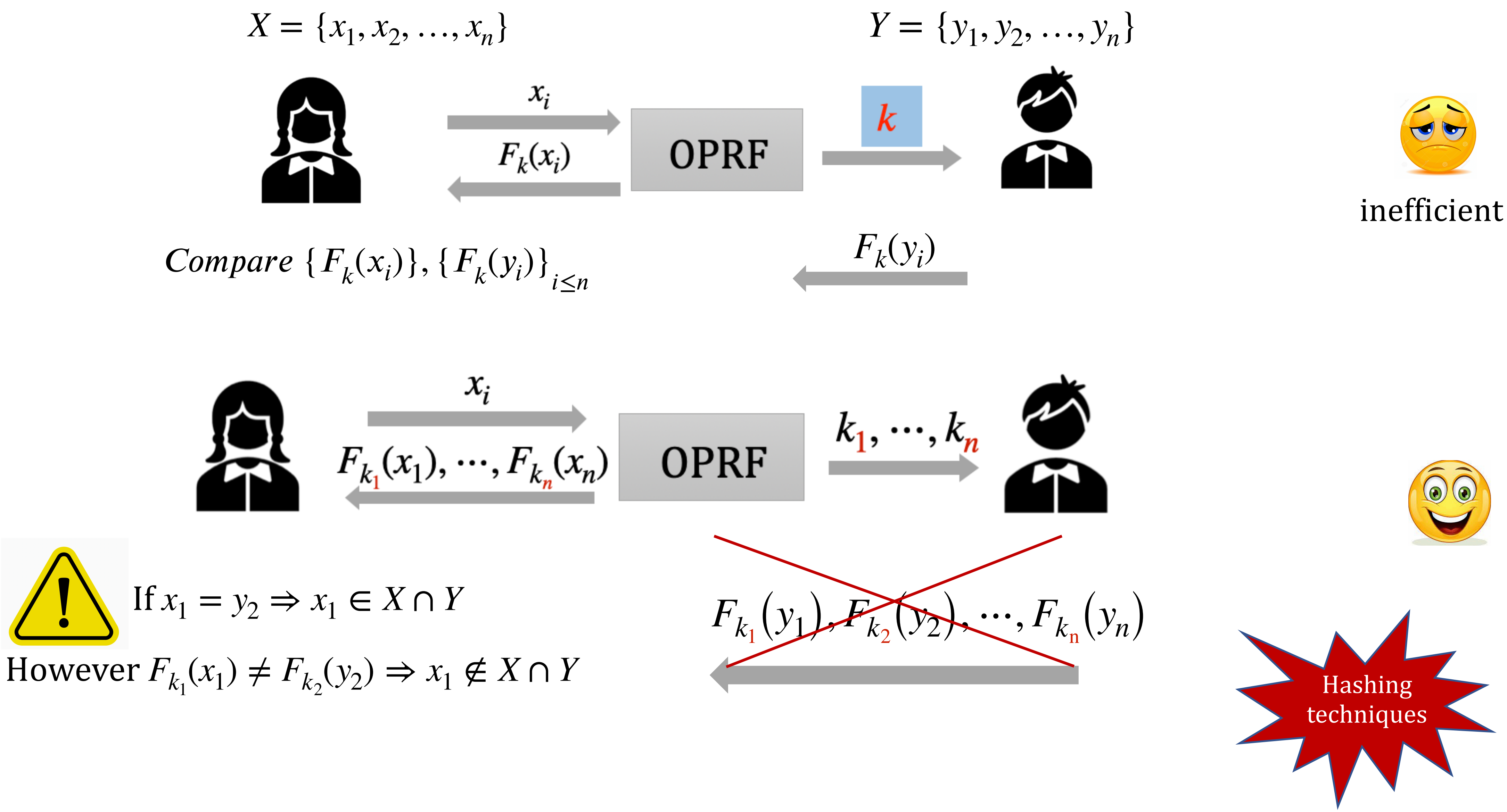
PSI from OPRF



PSI from OPRF



PSI from OPRF



Pseudorandom correlation generator (PCG)

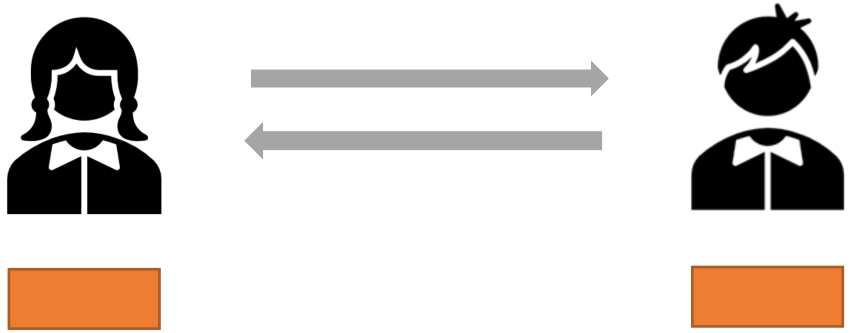
Pseudorandom correlation generator (PCG)

Secure computation ➡ Correlated random strings

Pseudorandom correlation generator (PCG)

Secure computation → Correlated random strings

Preprocessing phase:
Interactive protocol with short communication and computation. Alice and Bob store a small seed afterwards



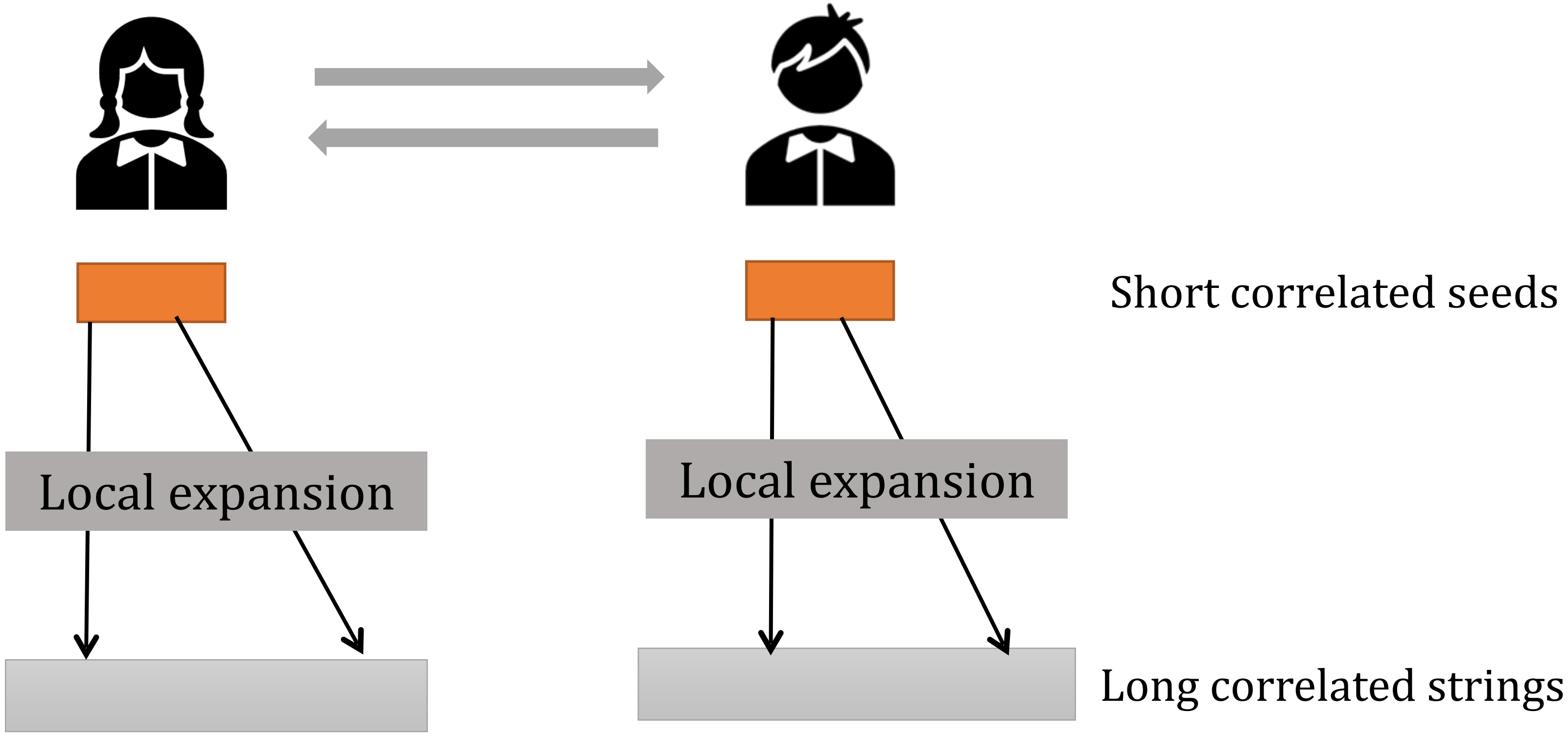
Short correlated seeds

Pseudorandom correlation generator (PCG)

Secure computation → Correlated random strings

Preprocessing phase:
Interactive protocol with short communication and computation. Alice and Bob store a small seed afterwards

'Silent' computation:
Alice and Bob
stretch their seeds into large pseudorandom correlated strings

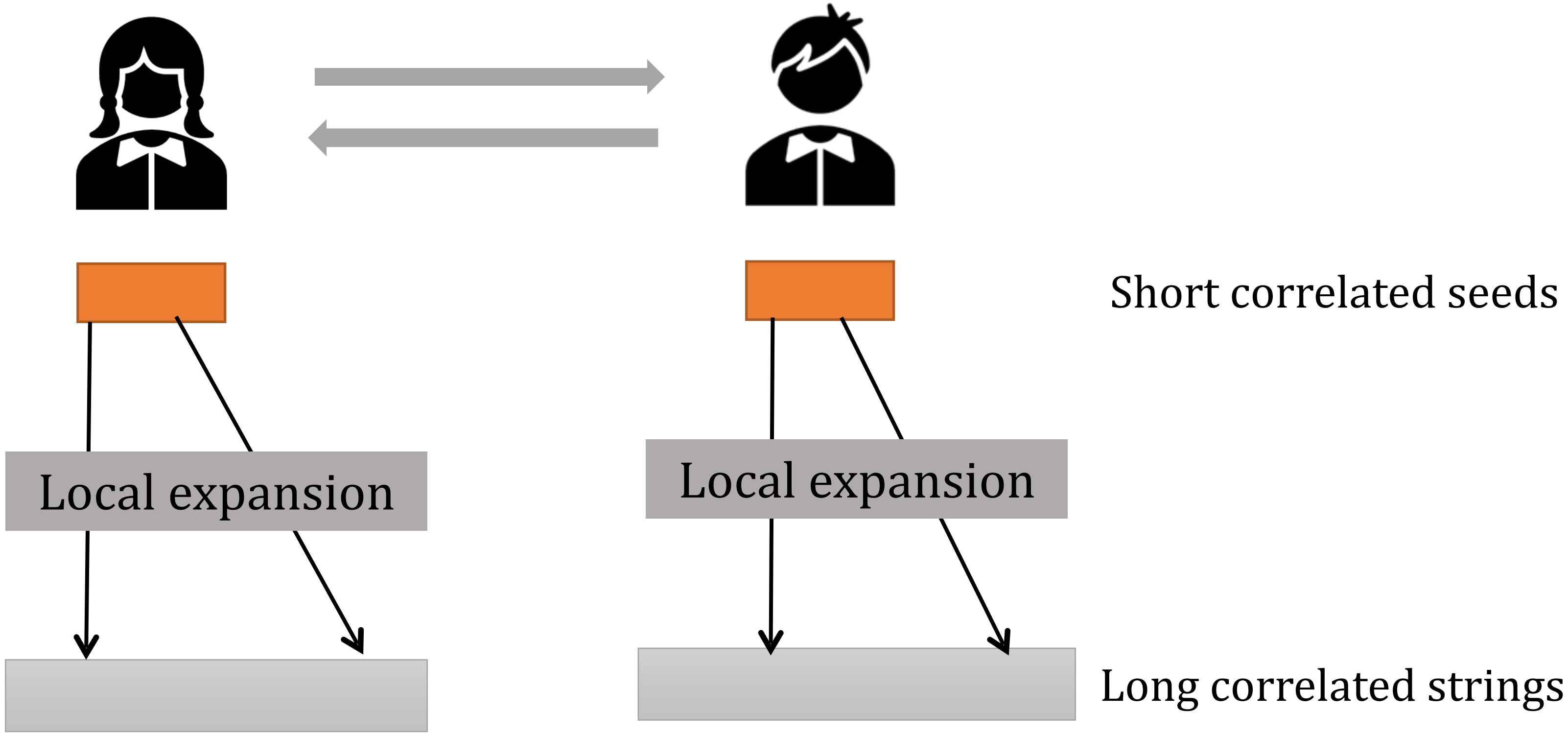


Pseudorandom correlation generator (PCG)

Secure computation → Correlated random strings

Preprocessing phase:
Interactive protocol with short communication and computation. Alice and Bob store a small seed afterwards

‘Silent’ computation:
Alice and Bob stretch their seeds into large pseudorandom correlated strings



Seeds can be silently expanded to long correlated strings

→ Parties just need to communicate and store a short correlated seeds

Pseudorandom correlation generator (PCG)

Pseudorandom correlation generator (PCG)

Batch-OLE

A finite field $E \subset F$

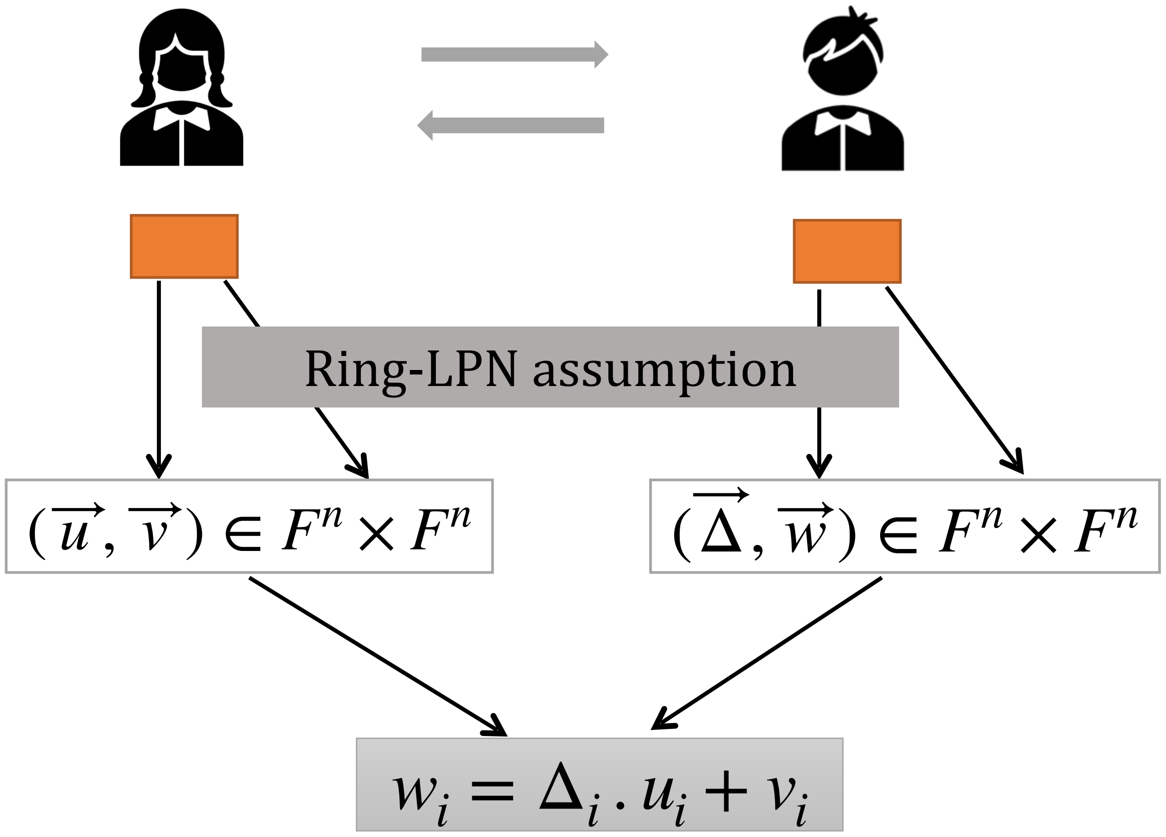
Vector OLE

Pseudorandom correlation generator (PCG)

Batch-OLE

A finite field $E \subset F$

Vector OLE

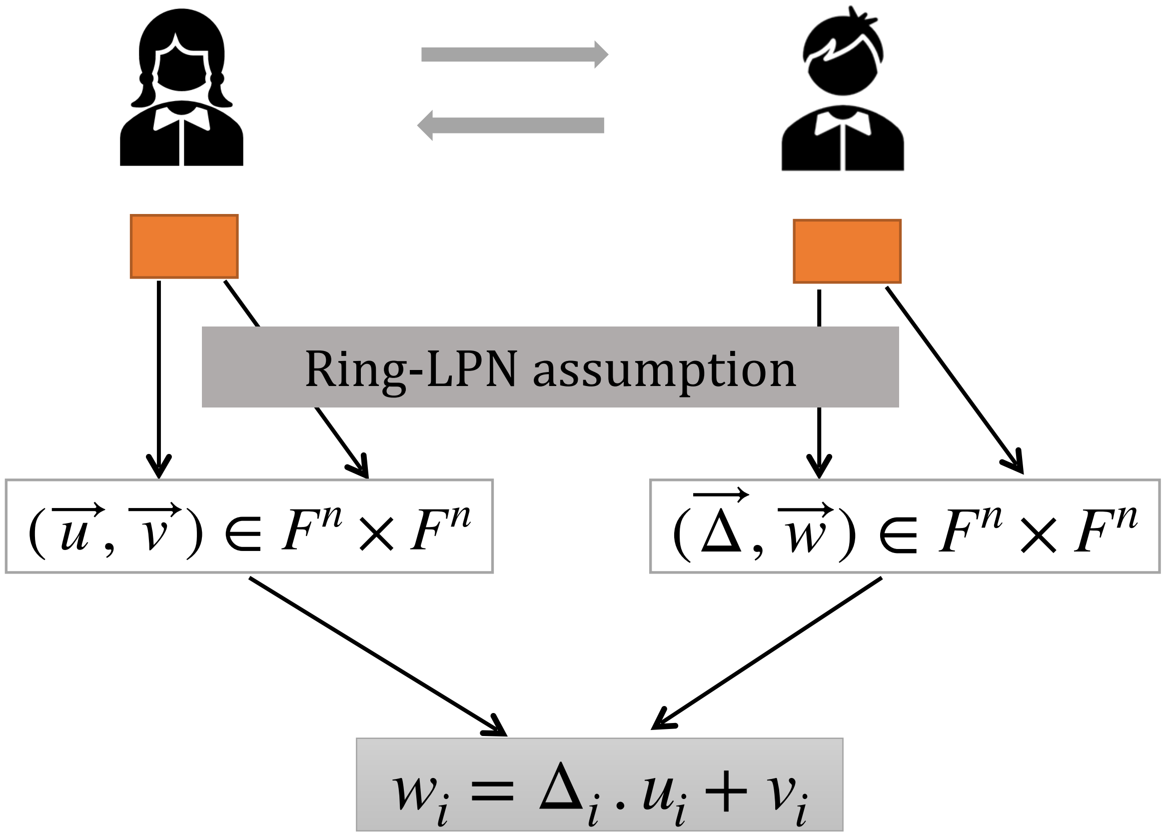


Pseudorandom correlation generator (PCG)

Batch-OLE

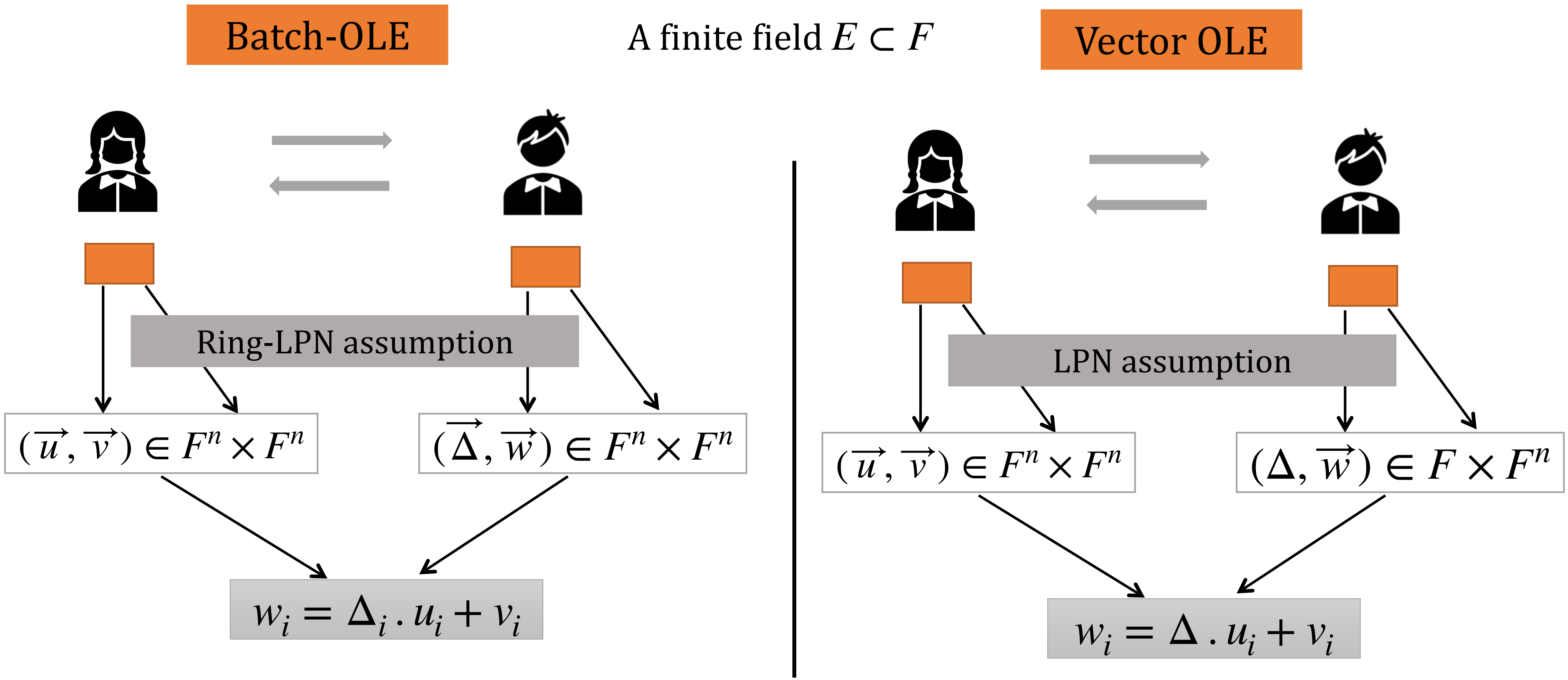
A finite field $E \subset F$

Vector OLE

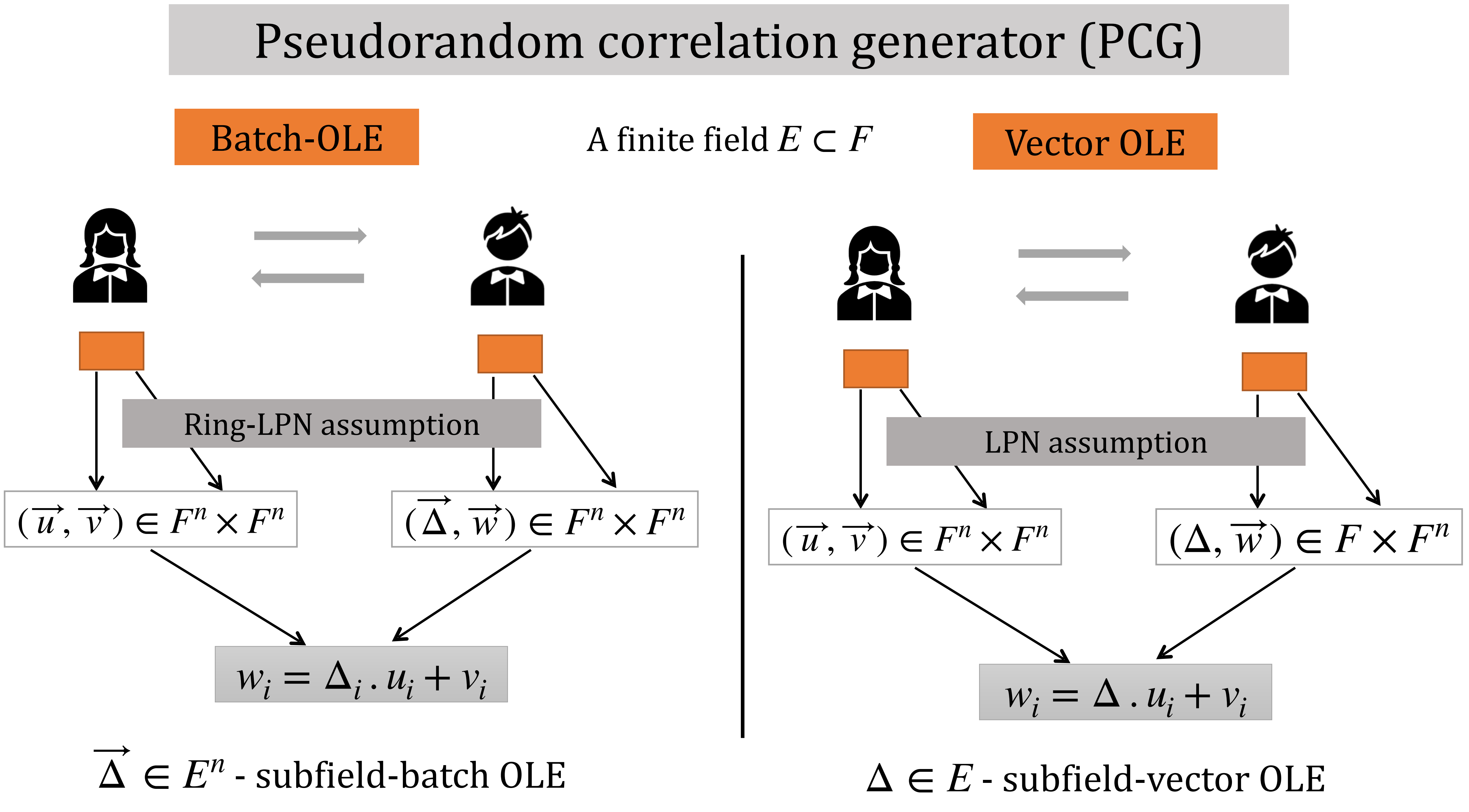


$\overrightarrow{\Delta} \in E^n$ - subfield-batch OLE

Pseudorandom correlation generator (PCG)



$\vec{\Delta} \in E^n$ - subfield-batch OLE


Pseudorandom correlation generator (PCG)
Batch-OLE
A finite field $E \subset F$
Vector OLE
Ring-LPN assumption
$(\overrightarrow{u}, \overrightarrow{v}) \in F^n \times F^n$
$(\overrightarrow{\Delta}, \overrightarrow{w}) \in F^n \times F^n$
$w_i = \Delta_i . u_i + v_i$
$\overrightarrow{\Delta} \in E^n$ - subfield-batch OLE
LPN assumption
$(\overrightarrow{u}, \overrightarrow{v}) \in F^n \times F^n$
$(\Delta, \overrightarrow{w}) \in F \times F^n$
$w_i = \Delta . u_i + v_i$
$\Delta \in E$ - subfield-vector OLE

Our contribution

Our contribution

- Construct two new PSIs from **Pseudorandom Correlation Generator (PCG)**

Our contribution

- Construct two new PSIs from **Pseudorandom Correlation Generator (PCG)**

- **A semi-honest PSI protocol** from subfield-vector OLE and generalized Cuckoo hashing:
  - The communication depends only on the input set and statistic security parameter
  - Handle Cuckoo hashing having multiple items per bin by a new OPRF
  - Efficient with competitive communication cost. For input size $\ell = 32,\ n = 2^{24},$ the communication is extremely small: only 147n bits

Our contribution

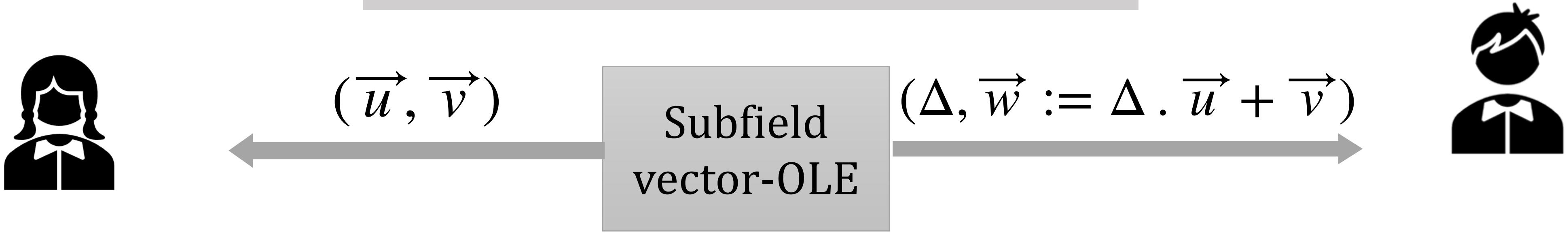
- Construct two new PSIs from **Pseudorandom Correlation Generator (PCG)**

- **A semi-honest PSI protocol** from subfield-vector OLE and generalized Cuckoo hashing:
  - The communication depends only on the input set and statistic security parameter
  - Handle Cuckoo hashing having multiple items per bin by a new OPRF
  - Efficient with competitive communication cost. For input size $\ell = 32,\ n = 2^{24}$, the communication is extremely small: only 147n bits

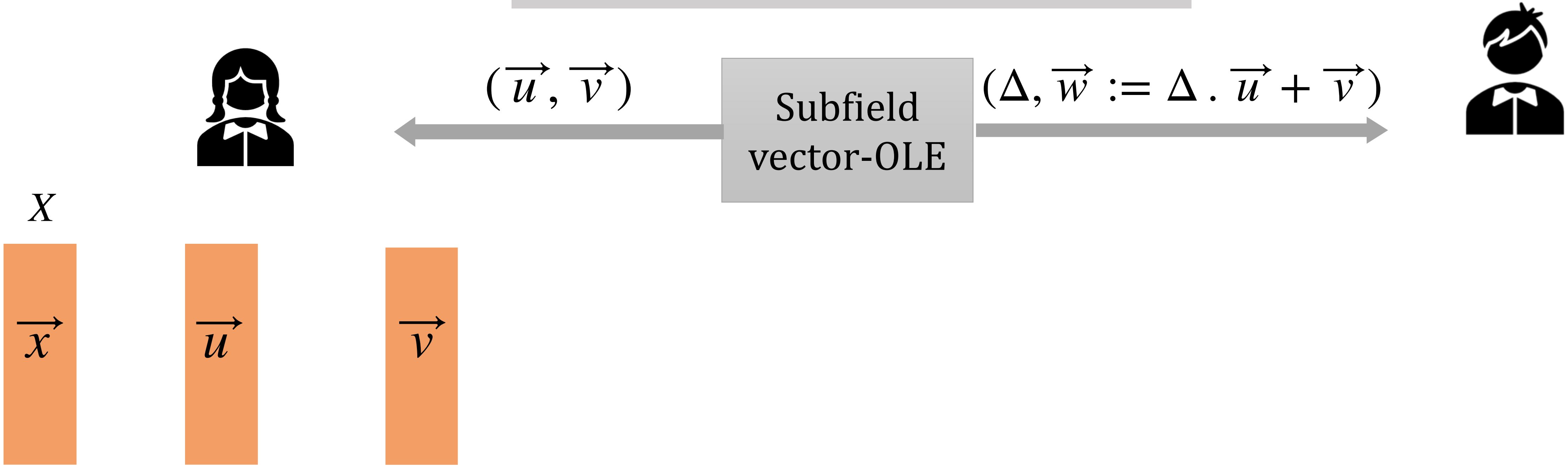
- **A malicious PSI protocol** from subfield-batch OLE and polynomial structure:
  - Secure without random oracle model or any tailor-made correlation robustness assumptions
  - Based on subfield-batch OLE on polynomial
  - Competitive communication even with the best ROM-based PSI

OPRF Construction

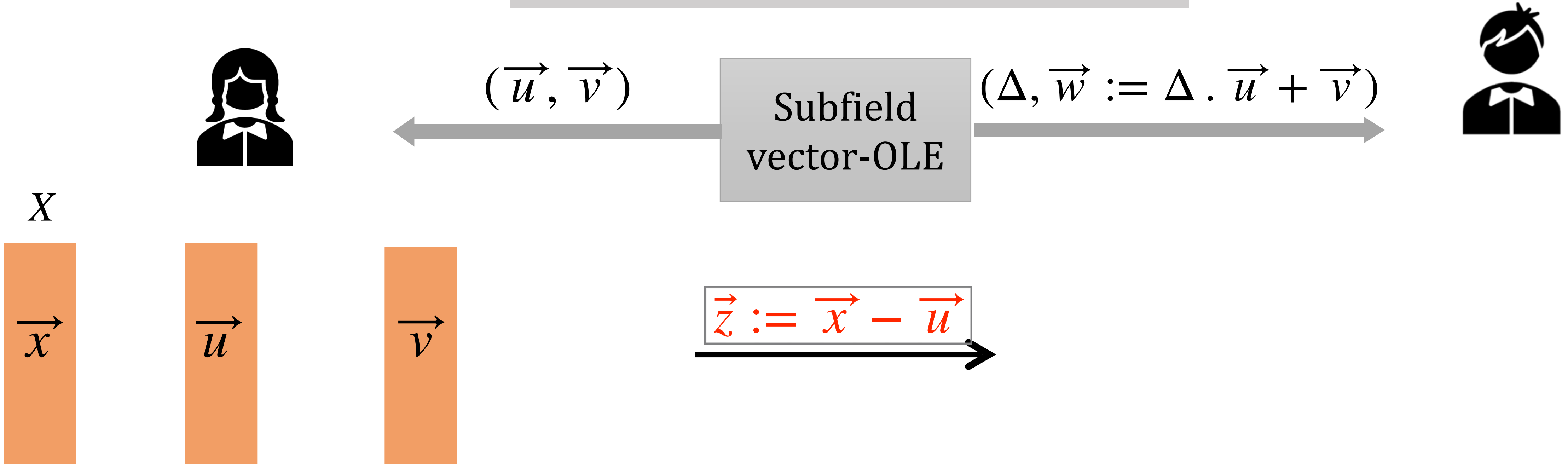
OPRF Construction



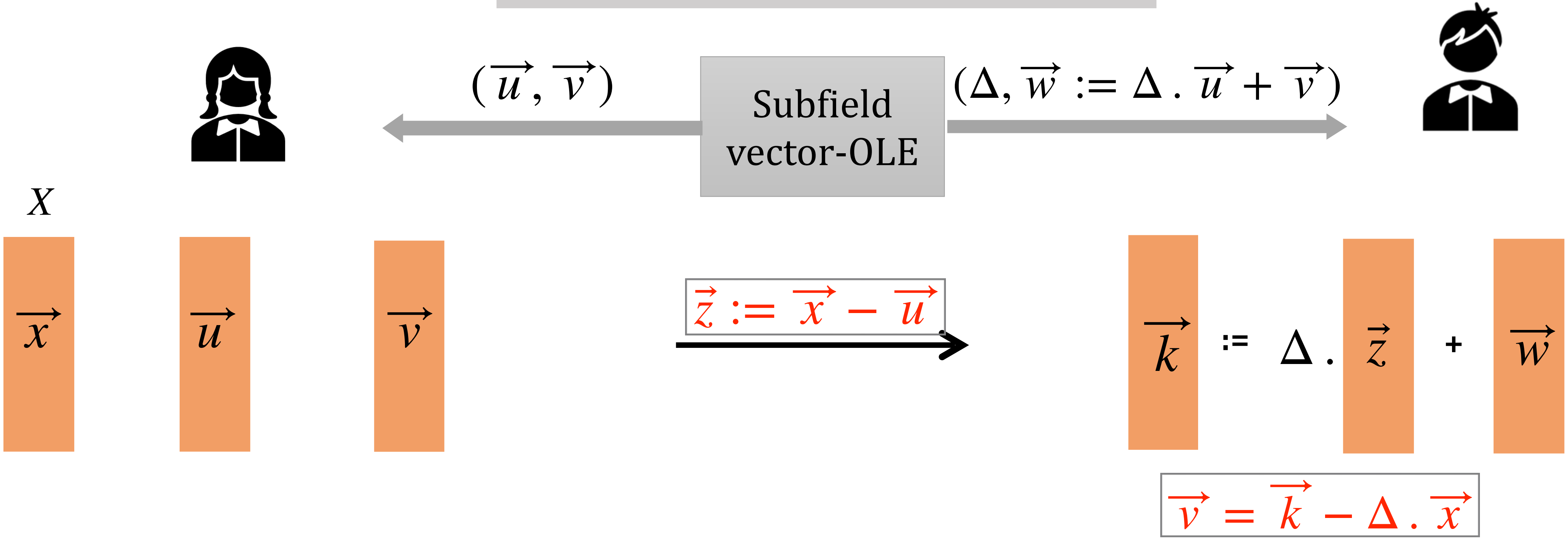
OPRF Construction



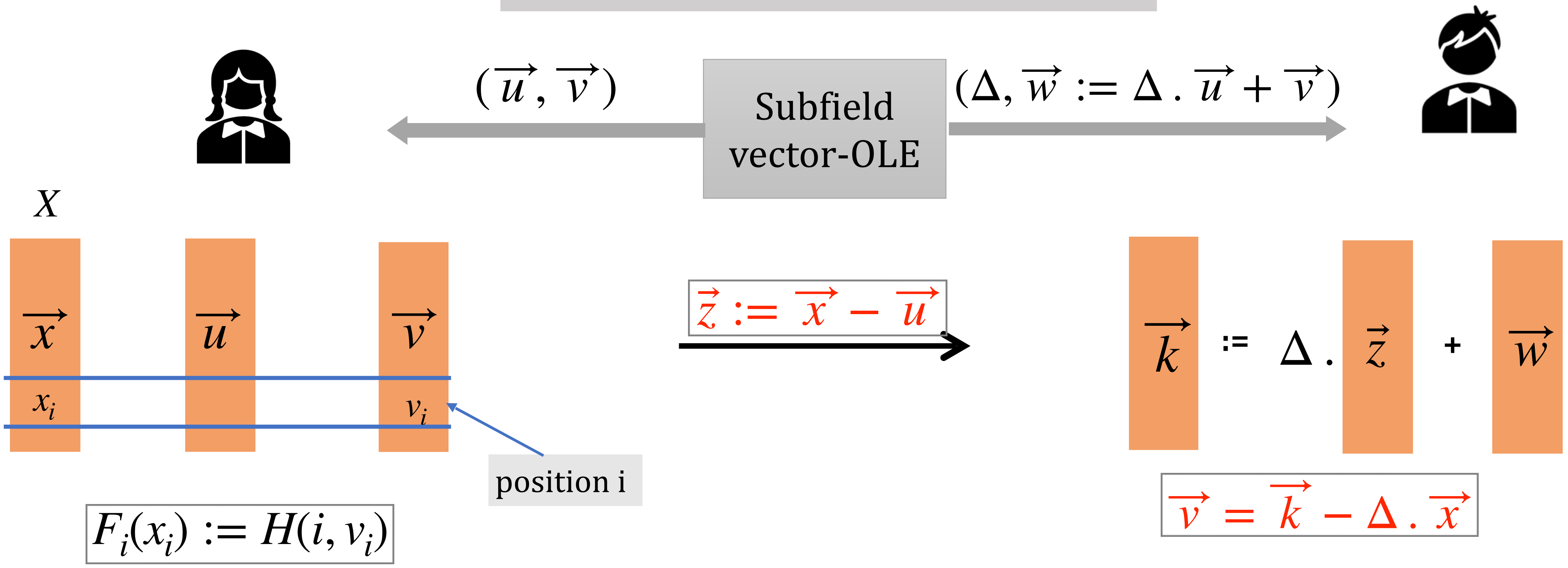
OPRF Construction



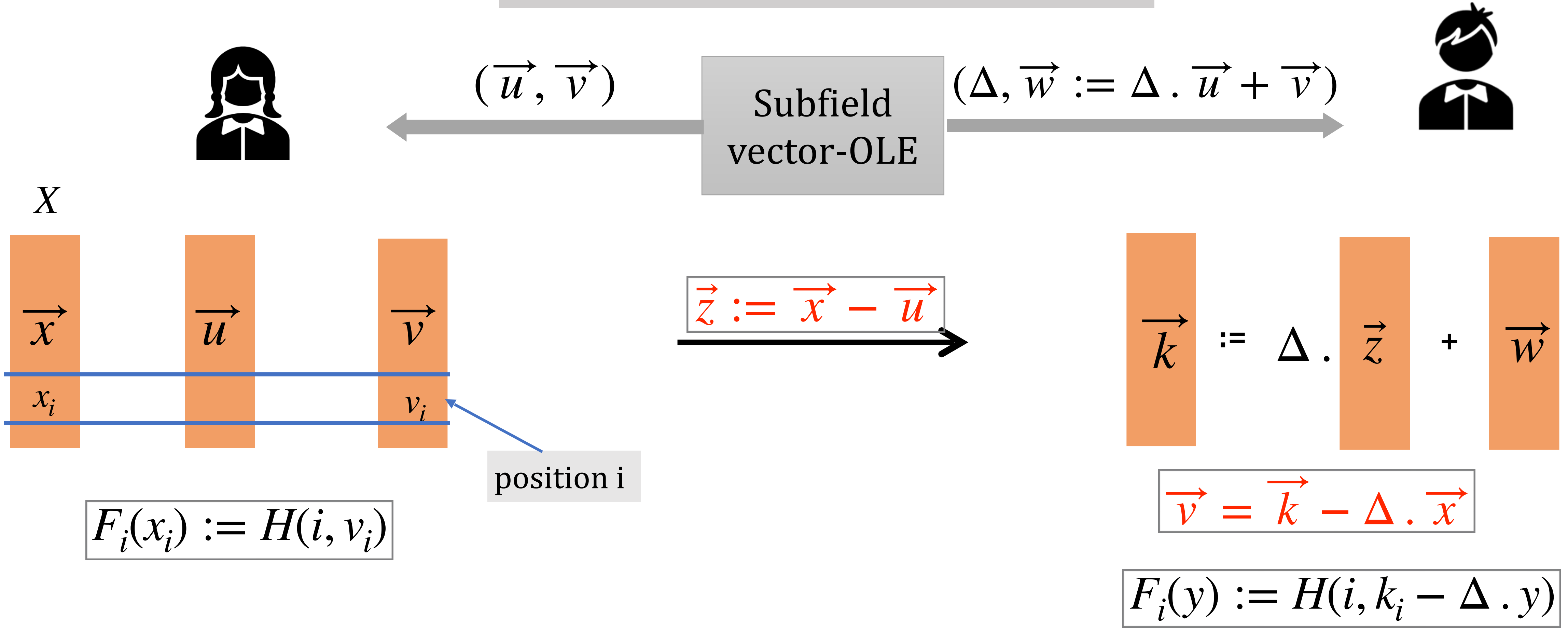
OPRF Construction



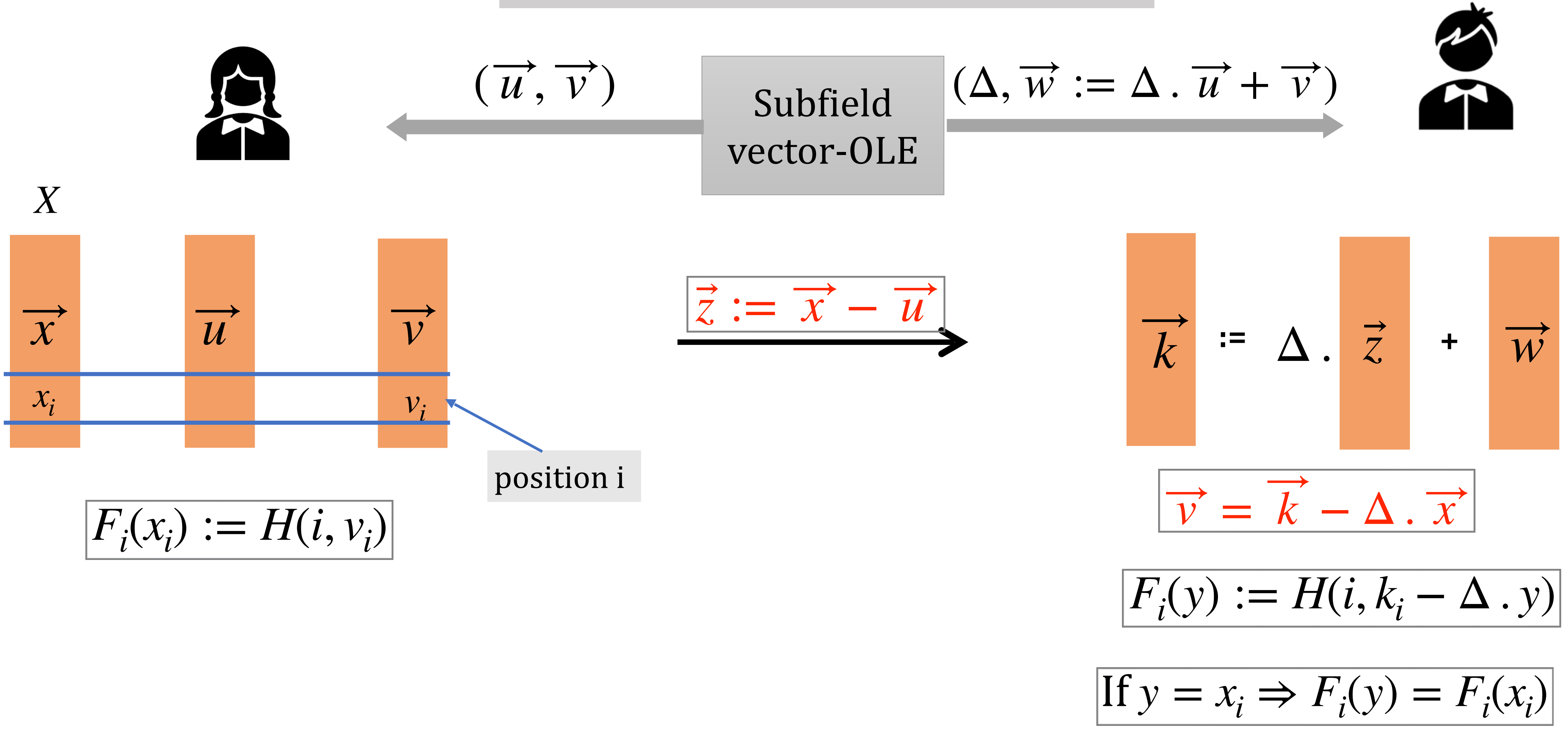
OPRF Construction



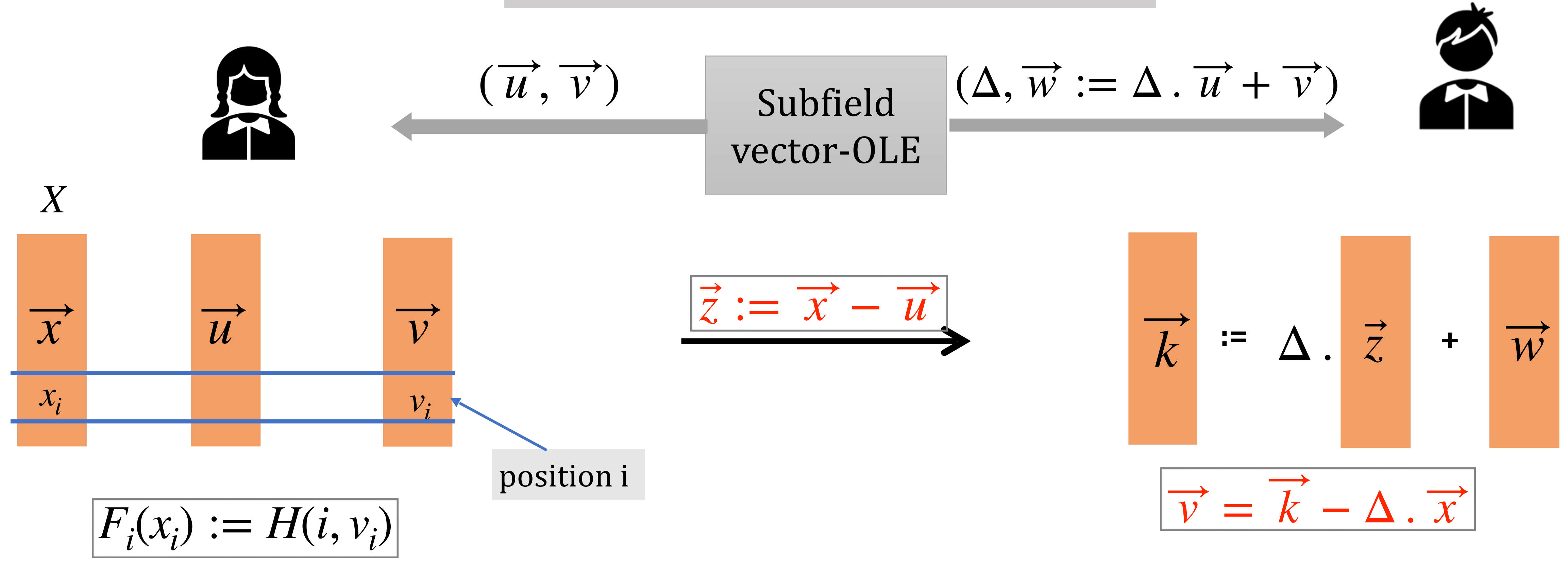
OPRF Construction



OPRF Construction



OPRF Construction



$$F_i(y) := H(i, k_i - \Delta . y)$$

If $y = x_i \Rightarrow F_i(y) = F_i(x_i)$

- Secure against semi-honest setting.
- Communication cost depends only on the bit-length of the input set.

Hashing scheme

Map n elements to m bins ($m > n$) by k hash functions $h_1, h_2, \ldots, h_k : \{0,1\}^* \rightarrow [m]$

Hashing scheme

Map n elements to m bins ($m > n$) by k hash functions $h_1, h_2, \ldots, h_k : \{0,1\}^* \rightarrow [m]$

Cuckoo hashing for Alice

Simple hashing for Bob

Hashing scheme

Map n elements to m bins ($m > n$) by k hash functions $h_1, h_2, \ldots, h_k : \{0,1\}^* \rightarrow [m]$

Cuckoo hashing for Alice

Simple hashing for Bob

Each bin contains at most one element

One element x is put into exactly one bin $h_i(x)$ for $i \in [k]$

Hashing scheme

Map n elements to m bins ($m > n$) by k hash functions $h_1, h_2, \ldots, h_k : \{0,1\}^* \to [m]$

Cuckoo hashing for Alice	Simple hashing for Bob

Each bin contains at most one element

One element x is put into exactly one bin $h_i(x)$ for $i \in [k]$

For empty bin, add a dummy item

Hashing scheme

Map n elements to m bins ($m > n$) by k hash functions $h_1, h_2, \ldots, h_k : \{0,1\}^* \rightarrow [m]$

Cuckoo hashing for Alice

Each bin contains at most one element

One element x is put into exactly one bin $h_i(x)$ for $i \in [k]$

For empty bin, add a dummy item

In our work, each bin contains at most d elements

➡ (d,k)- general Cuckoo hashing

Simple hashing for Bob

Hashing scheme

Map n elements to m bins ($m > n$) by k hash functions $h_1, h_2, \dots, h_k : \{0,1\}^* \to [m]$

Cuckoo hashing for Alice

Each bin contains at most one element

One element x is put into exactly one bin $h_i(x)$ for $i \in [k]$

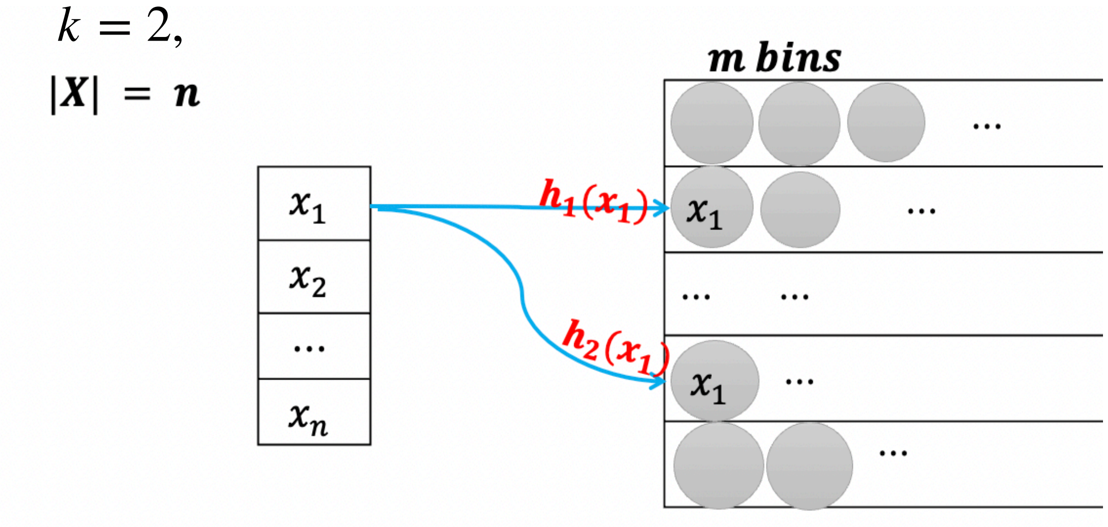
For empty bin, add a dummy item

In our work, each bin contains at most d elements

(d,k)- general Cuckoo hashing

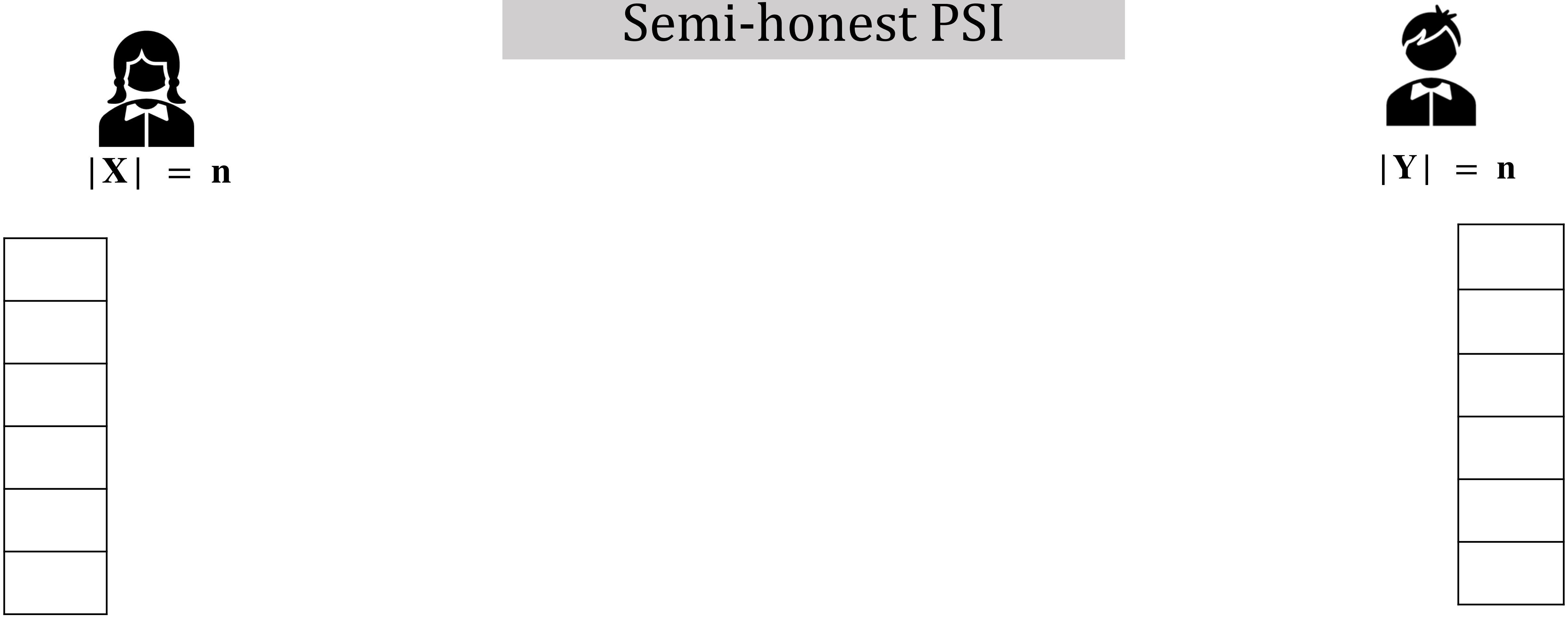
Simple hashing for Bob

Any element x is put into all k bins $h_i(x)$

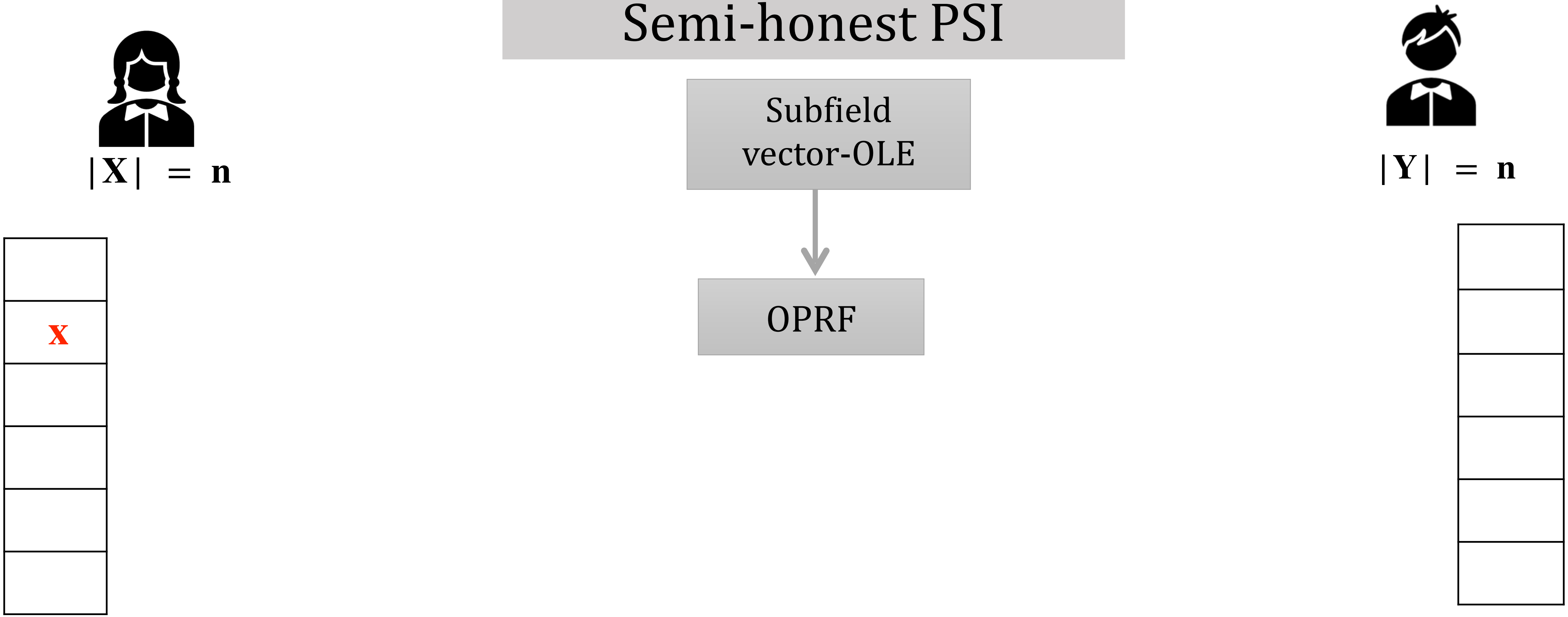


Semi-honest PSI

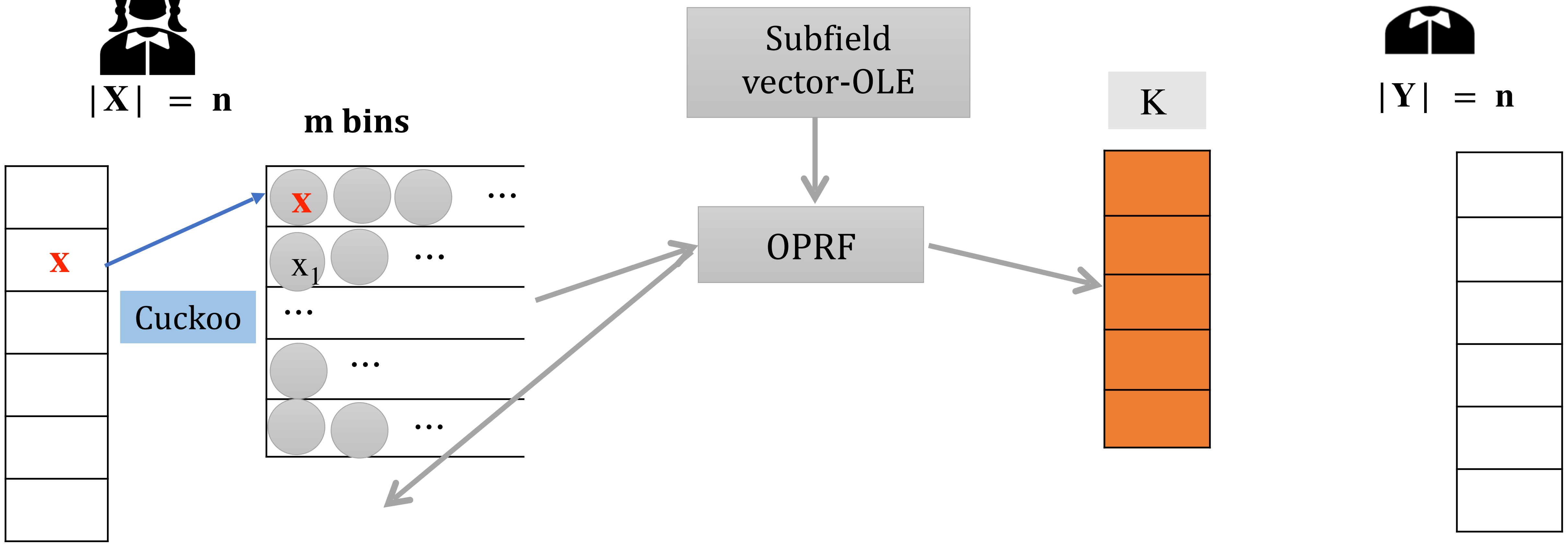
Semi-honest PSI

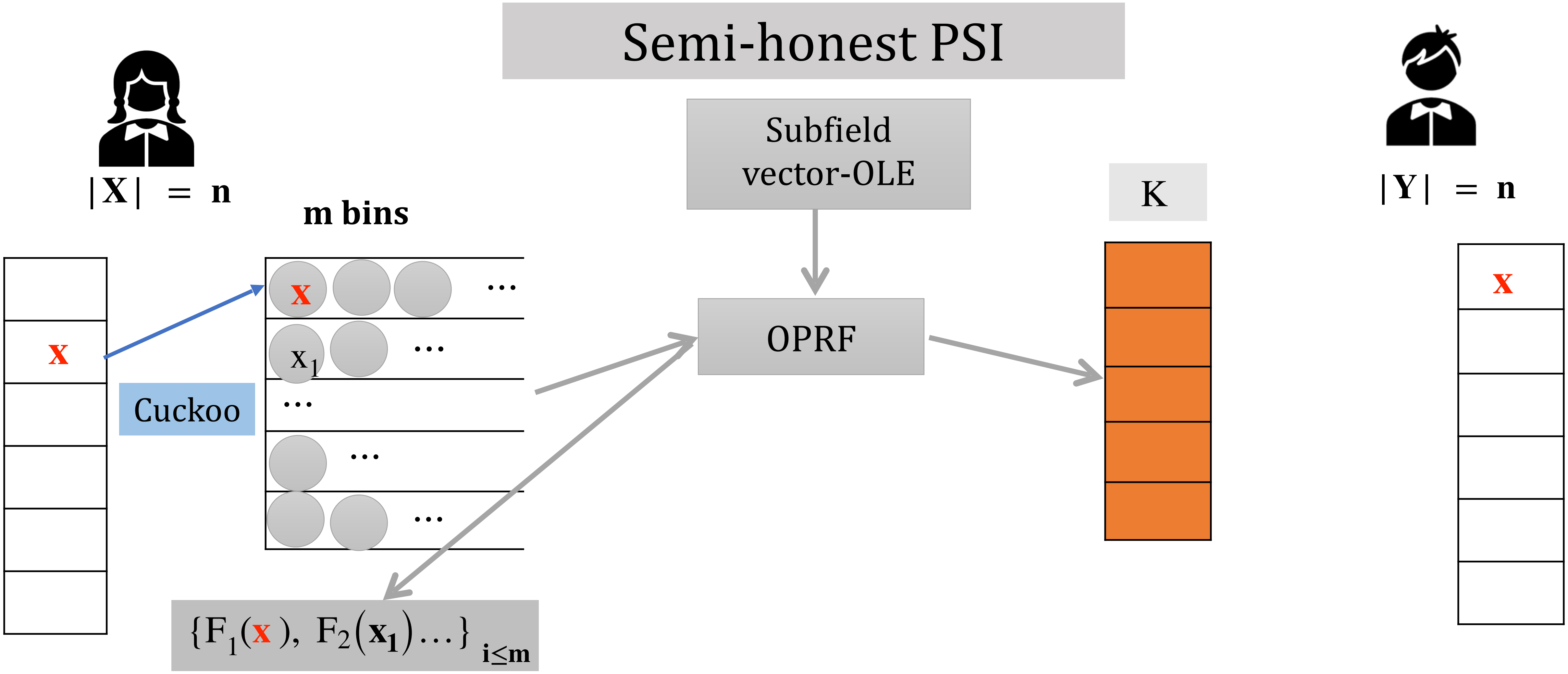


Semi-honest PSI

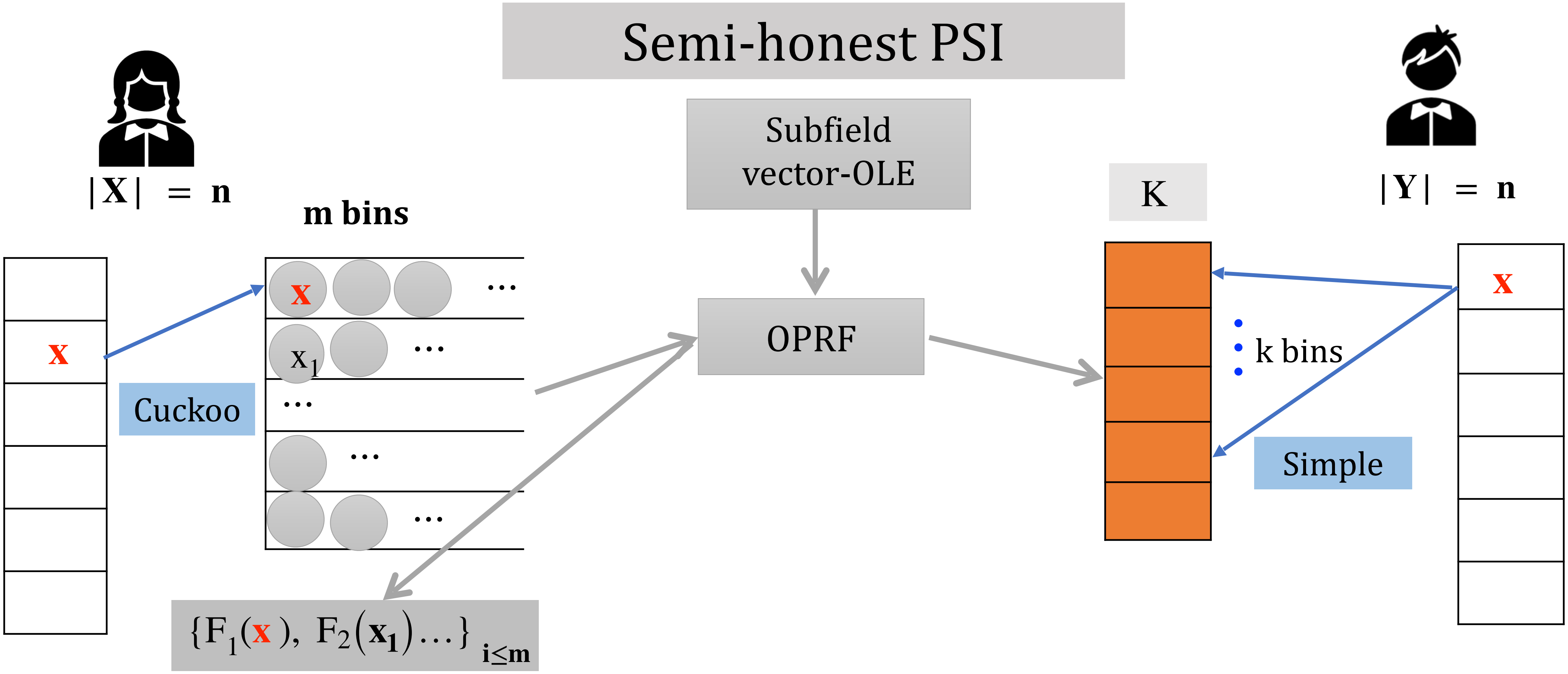


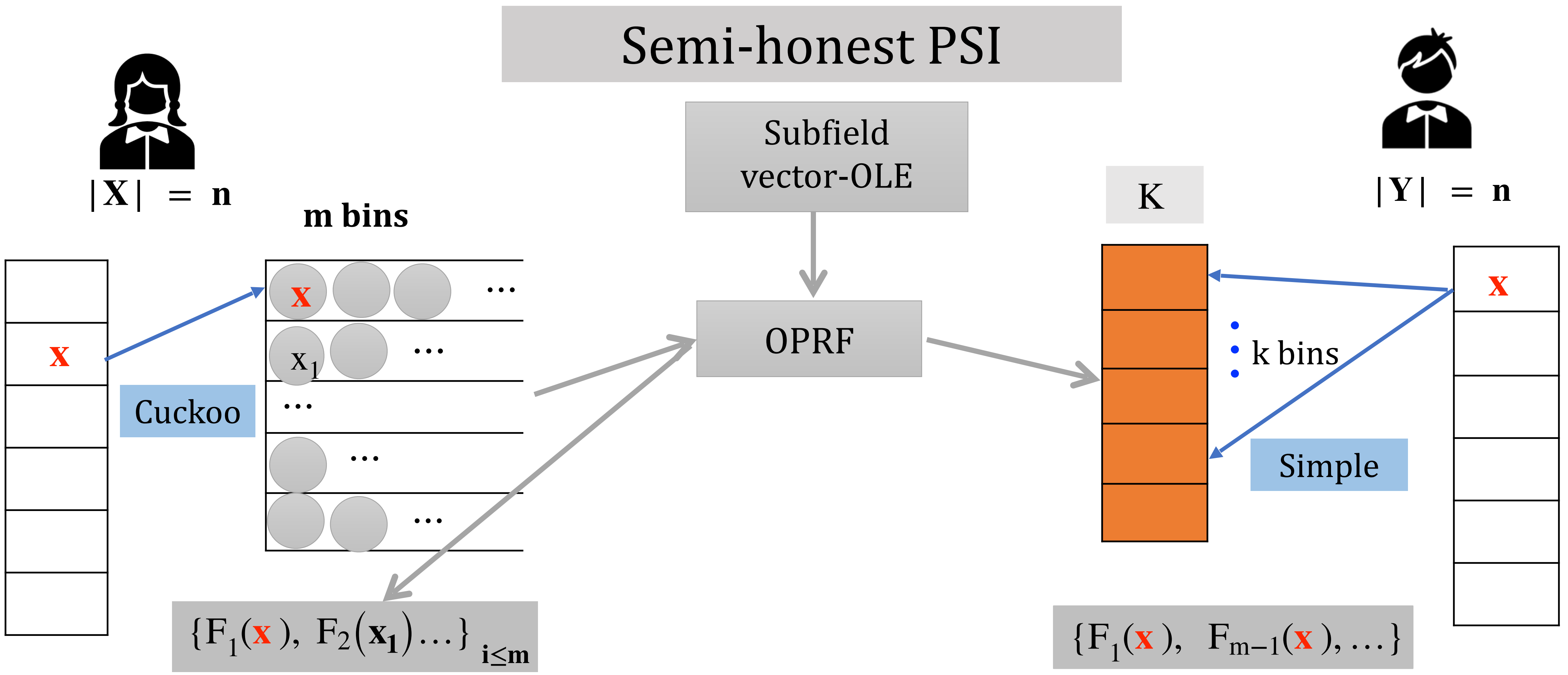
Semi-honest PSI

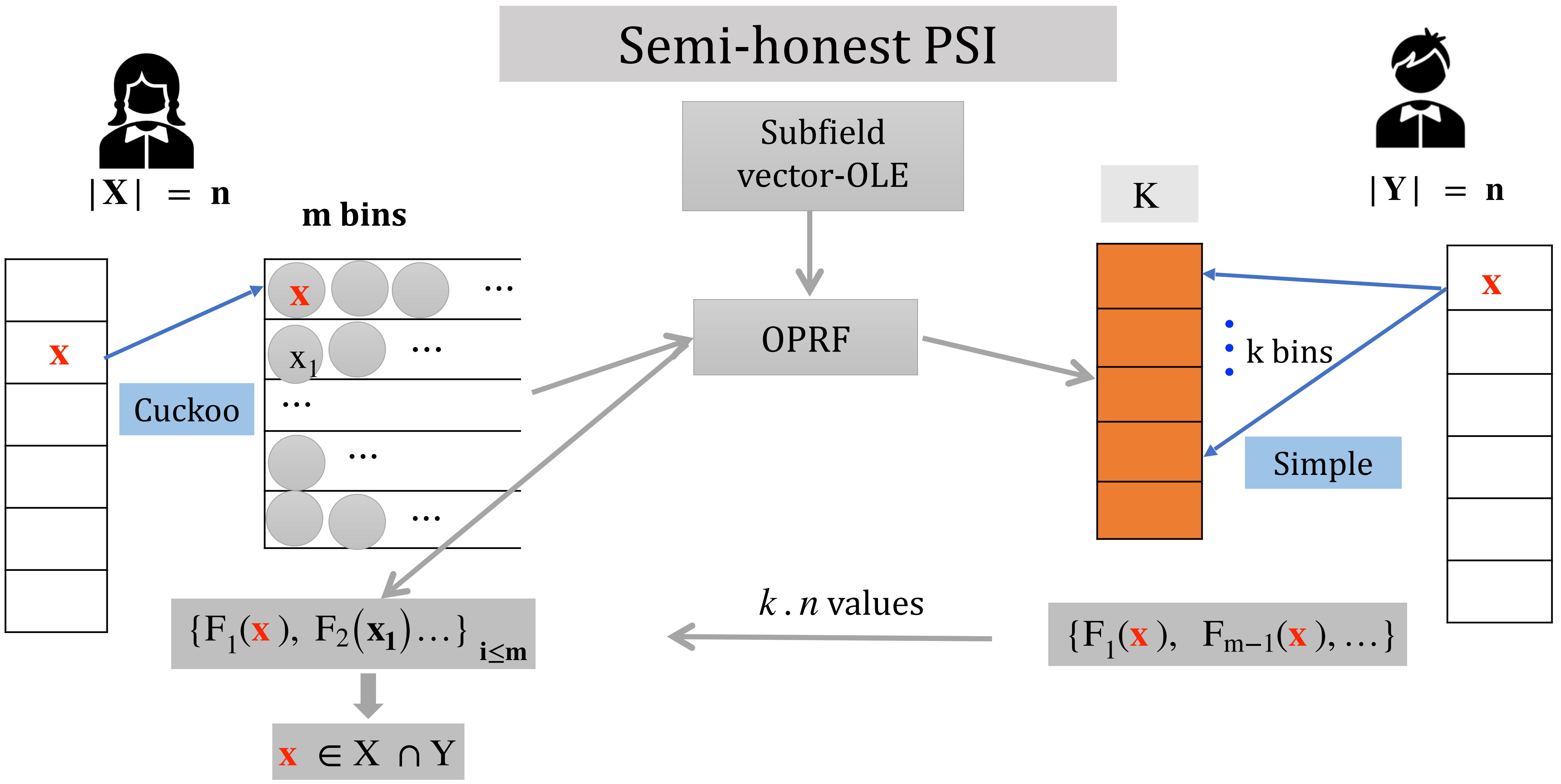


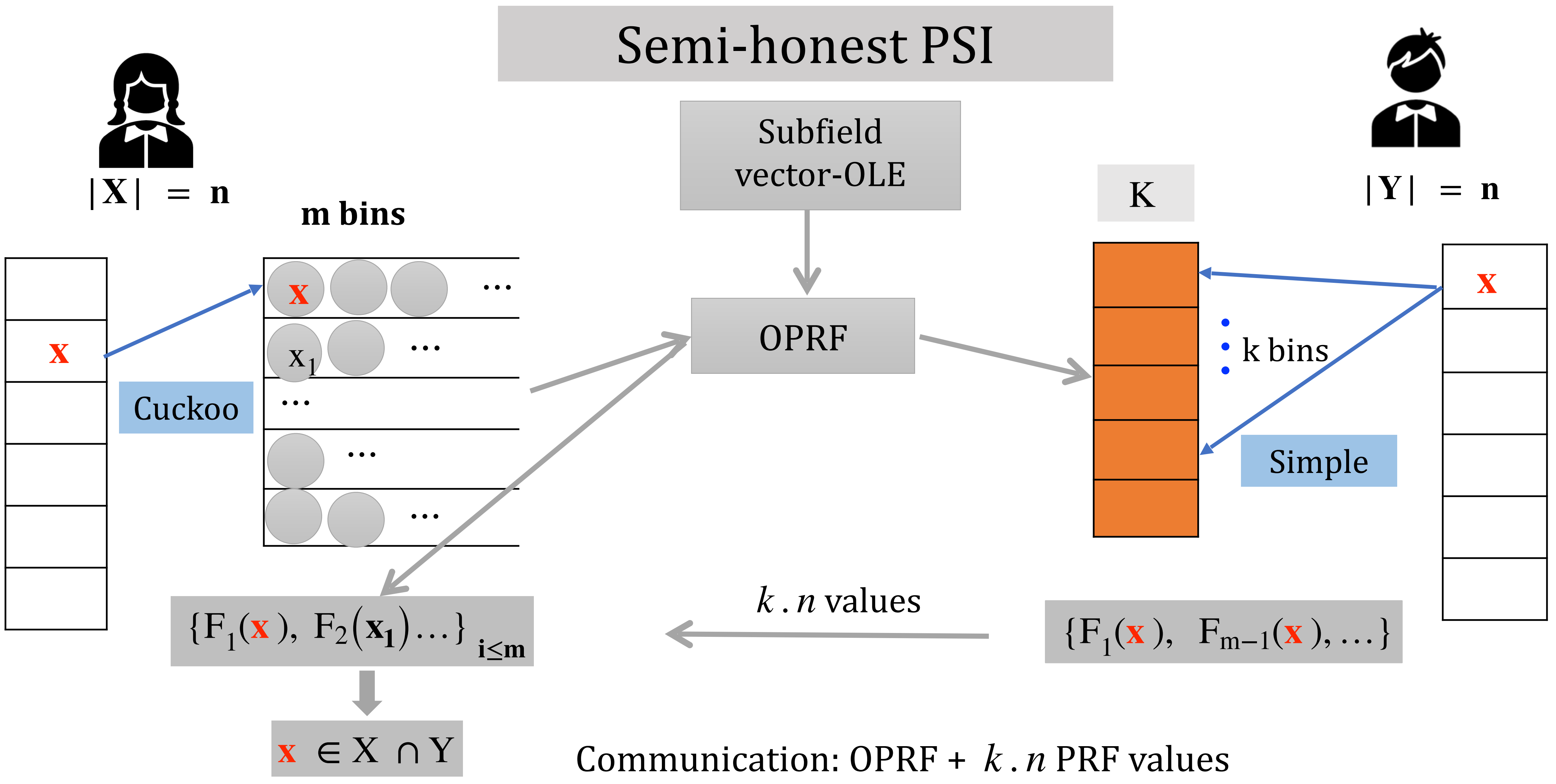


Semi-honest PSI
|X| = n
m bins
x
x
x1
…
Cuckoo
Subfield vector-OLE
OPRF
K
|Y| = n
x
{F1(x), F2(x1)…} i≤m


Semi-honest PSI
Subfield vector-OLE
OPRF
K
|X| = n
|Y| = n
m bins
x
x
x_1
x
Cuckoo
k bins
Simple
$\{F_1(x), F_2(x_1) \ldots\}_{i \le m}$


Semi-honest PSI
|X| = n
m bins
Cuckoo
Subfield vector-OLE
OPRF
K
k bins
Simple
|Y| = n
$\{F_1(x), F_2(x_1) \dots\}_{i \leq m}$
$\{F_1(x), F_{m-1}(x), \dots\}$


Semi-honest PSI
|X| = n
m bins
x
Cuckoo
x
x1
Subfield vector-OLE
OPRF
K
k bins
Simple
|Y| = n
x
{F1(x), F2(x1)...} i≤m
k . n values
{F1(x), Fm−1(x), ...}
x ∈ X ∩ Y


Semi-honest PSI
|X| = n
m bins
Subfield vector-OLE
OPRF
K
|Y| = n
x
x
x₁
Cuckoo
k bins
Simple
x
{F₁(x), F₂(x₁)…} i≤m
k . n values
{F₁(x), Fm−1(x), …}
x ∈ X ∩ Y
Communication: OPRF + k . n PRF values

Theoretical comparison

	$n = 2^{14}$	$n = 2^{16}$	$n = 2^{20}$	$n = 2^{24}$
Semi-honest setting				
KKRT16 [KKRT16]	$930n$	$936n$	$948n$	$960n$
PRTY19 [PRTY19] low*	$491n$	$493n$	$493n$	$494n$
PRTY19 [PRTY19] fast*	$560n$	$571n$	$579n$	$587n$
CM20 [CM20]	$668n$	$662n$	$674n$	$676n$
PRTY20 [PRTY20]	$1244n$	$1192n$	$1248n$	$1278n$
RS21 [RS21]	$2024n$	$898n$	$406n$	$374n$
RS21 [RS21] enhanced**	$280n$	$260n$	$263n$	$275n$
Ours ($\ell = 64$, GCH)	$246n$	$220n$	$210n$	$209n$
Ours ($\ell = 48$, GCH)	$215n$	$189n$	$179n$	$178n$
Ours ($\ell = 32$, GCH)	$184n$	$158n$	$148n$	$147n$
Ours ($\ell = 64$, 2CH)	$214n$	$190n$	$183n$	$185n$
Ours ($\ell = 48$, 2CH)	$193n$	$169n$	$162n$	$164n$
Ours ($\ell = 32$, 2CH)	$171n$	$148n$	$141n$	$142n$
Ours ($\ell = 64$, SH, $N = n/10$)	$332n$	$302n$	$284n$	$276n$
Ours ($\ell = 48$, SH, $N = n/10$)	$261n$	$230n$	$209n$	$198n$
Ours ($\ell = 32$, SH, $N = n/10$)	$191n$	$158n$	$133n$	$120n$
Ours ($\ell = 64$, SH, $N = 1$) ***	$154n$	$131n$	$125n$	$128n$
Ours ($\ell = 48$, SH, $N = 1$) ***	$138n$	$115n$	$109n$	$112n$
Ours ($\ell = 32$, SH, $N = 1$) ***	$122n$	$99n$	$93n$	$96n$

n: the size of the database.

l: bit-length of the inputs in the database.

[*] PRTY19 has two variants:
- SpOT-low (lowest communication, higher computation).
- SpOT-fast (higher communication, better computation).

[**] Using the 3H-GCT OKVS of GPRTY21 instead of PaXoS, and the VOLE of CRR21 instead of the one from WYKW21.

[***] Using $n = 1$ requires an expensive degree-n polynomial interpolation.

Thank you, Questions? ☺

For more details: ia.cr/2022/334