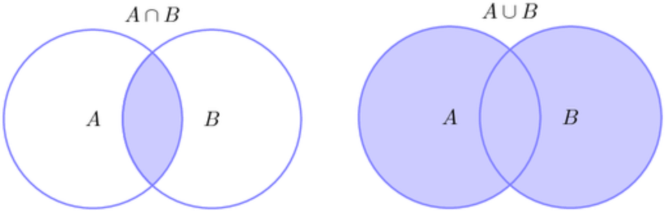
A Framework of Private Set Operations from Multi-query Reverse Private Membership Test



Yu Chen
Shandong University

Talk based on the following joint works

- **Yu Chen**, Min Zhang, Cong Zhang, Minglang Dong, Weiran Liu.
  Private Set Operations from Multi Query Reverse Private Membership Test.
  PKC 2024.

Outline

1. PSO Framework from mqRPMT
2. Construction of mqRPMT
    - 1st Construction from Commutative Weak PRF
    - 2nd Construction from Permuted Oblivious PRF
    - Connection Between mqPMT and mqRPMT
3. Comparison and Experimentation
4. Summary

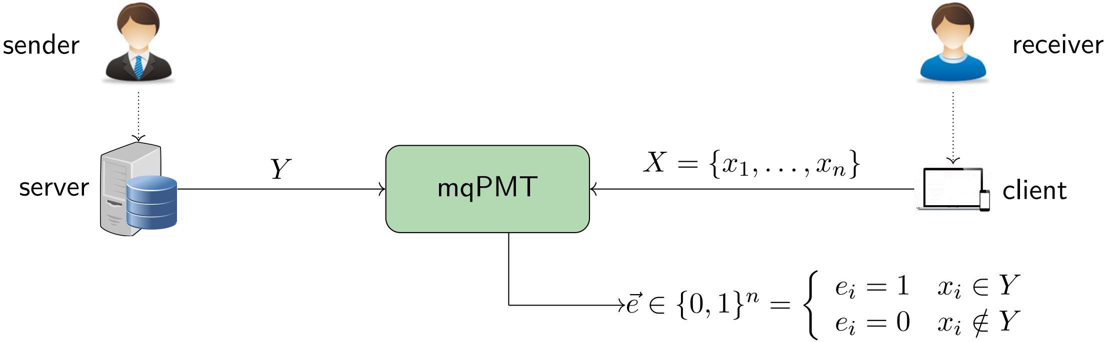
The landscape of PSO is isolated and complex.
Is there a unified yet simple framework?


问题的关键是找到关键问题

Outline

1. PSO Framework from mqRPMT
2. Construction of mqRPMT
    - 1st Construction from Commutative Weak PRF
    - 2nd Construction from Permuted Oblivious PRF
    - Connection Between mqPMT and mqRPMT
3. Comparison and Experimentation
4. Summary

Start Point: multi-query Private Membership Test (mqPMT) underlying PSI



sender

receiver

server

Y

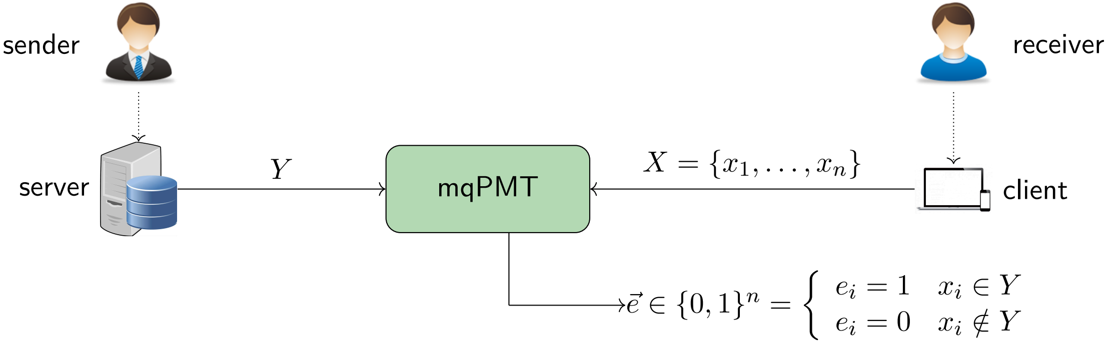
mqPMT

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

client

$$\vec{e} \in \{0,1\}^n = \begin{cases} e_i = 1 & x_i \in Y \\ e_i = 0 & x_i \notin Y \end{cases}$$

Start Point: multi-query Private Membership Test (mqPMT) underlying PSI



sender

receiver

server

Y

mqPMT

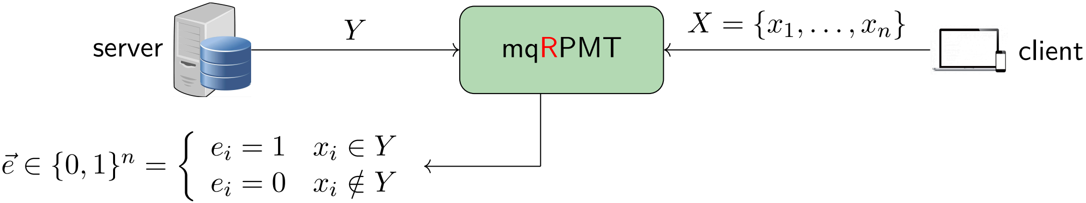
$X = \{x_1, \dots, x_n\}$

client

$$\vec{e} \in \{0,1\}^n = \begin{cases} e_i = 1 & x_i \in Y \\ e_i = 0 & x_i \notin Y \end{cases}$$

- Problem: the client learns both x_i and e_i, a.k.a. the intersection $\rightsquigarrow$ not suitable for protocols that should hide intersection, such as PCSI and PSU.

The core protocol: multi-query Reverse Private Membership Test (mqRPMT)



server

Y

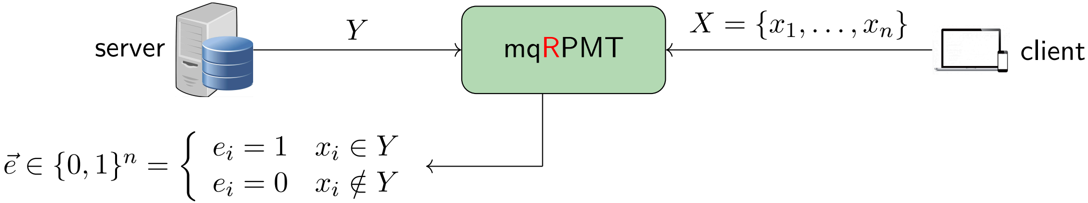
mqRPMT

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

client

$$\vec{e} \in \{0,1\}^n = \begin{cases} e_i = 1 & x_i \in Y \\ e_i = 0 & x_i \notin Y \end{cases}$$

The core protocol: multi-query Reverse Private Membership Test (mqRPMT)


server

Y

mqRPMT

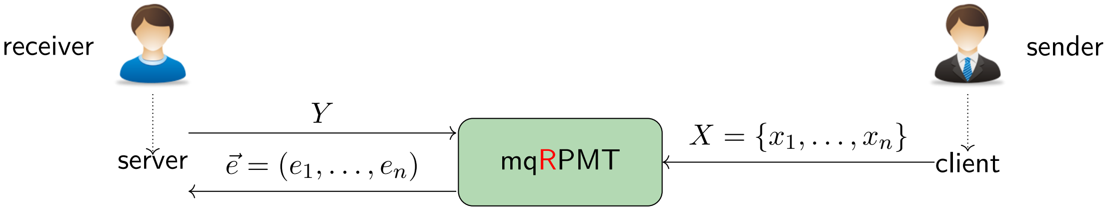
$X = \{x_1, \ldots, x_n\}$

client

$$\vec{e} \in \{0,1\}^n = \begin{cases} e_i = 1 & x_i \in Y \\ e_i = 0 & x_i \notin Y \end{cases}$$

- The server learns e_i, while the client learns x_i, a.k.a. the information of intersection is shared between the two parties $\rightsquigarrow$ suitable for all PSO protocols

PSO from mqRPMT



receiver

sender

server

client

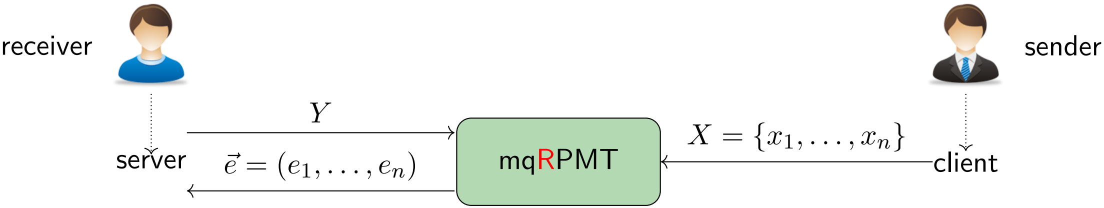
Y

$\vec{e} = (e_1, \ldots, e_n)$

mqRPMT

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

PSO from mqRPMT



receiver

sender

server

Y

$\vec{e} = (e_1, \ldots, e_n)$

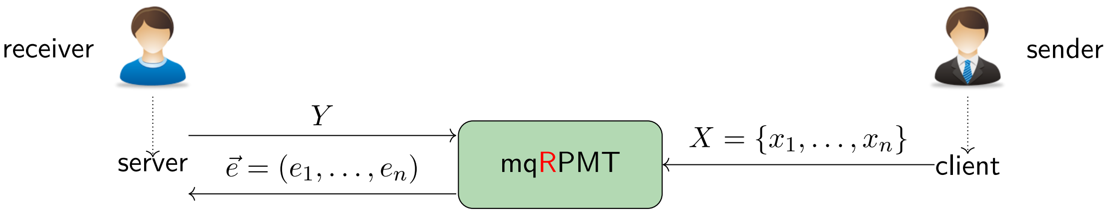
mqRPMT

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

client

directly yields PSI-card: $|X \cap Y|$ is the Hamming weight of $\vec{e}$

PSO from mqRPMT



receiver

sender

Y

server

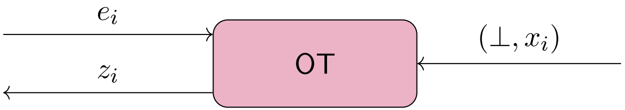
$\vec{e} = (e_1, \ldots, e_n)$

mqRPMT

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

client

yields PSI coupled with OT: receiver obtains $X \cap Y$



e_i

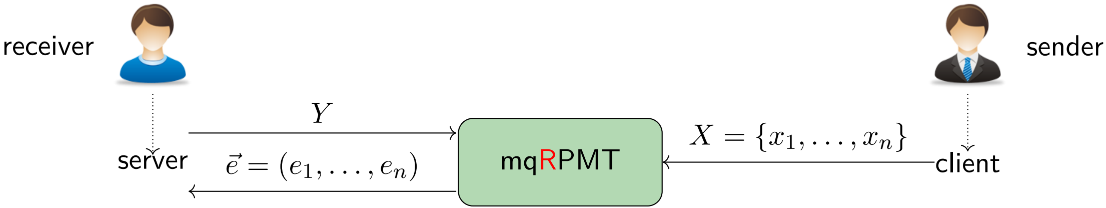
z_i

OT

$(\perp, x_i)$

$$z_i = \begin{cases} \perp & e_i = 0 \\ x_i & e_i = 1 \end{cases}$$

PSO from mqRPMT



receiver

sender

Y

server

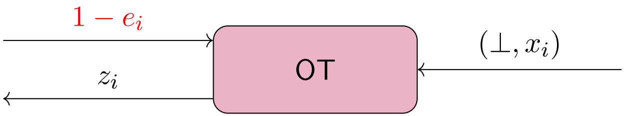
$\vec{e} = (e_1, \ldots, e_n)$

mqRPMT

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

client

yields PSU coupled with OT (flipping $\vec{e}$): receiver obtains $X - Y$



$1 - e_i$

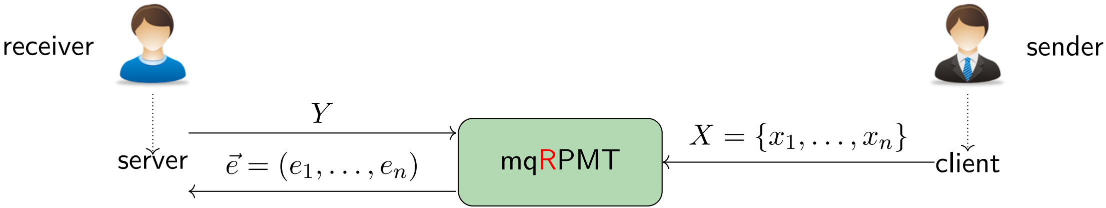
OT

$(\perp, x_i)$

z_i

$$z_i = \begin{cases} x_i & e_i = 0 \\ \perp & e_i = 1 \end{cases}$$

PSO from mqRPMT



receiver

sender

server

client

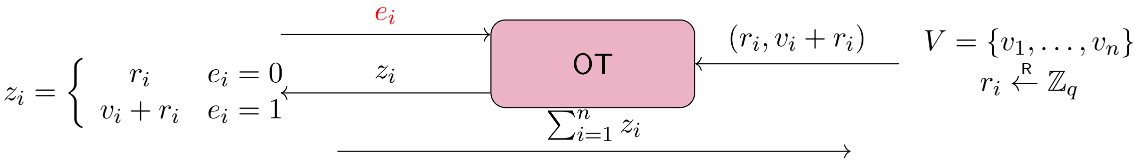
Y

$\vec{e} = (e_1, \ldots, e_n)$

mqRPMT

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

yields PSI-card-sum coupled with OT and masking trick



e_i

z_i

OT

$(r_i, v_i + r_i)$

$V = \{v_1, \ldots, v_n\}$

$r_i \xleftarrow{\mathsf{R}} \mathbb{Z}_q$

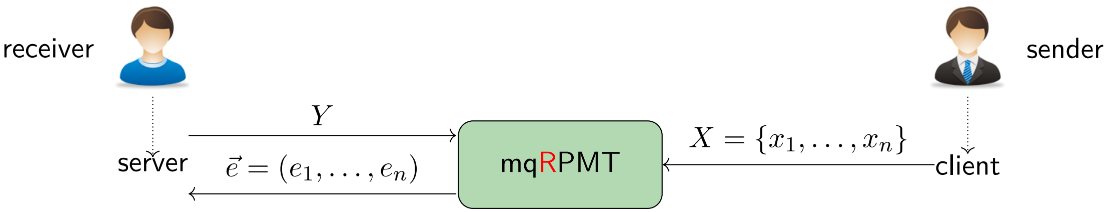
$$z_i = \begin{cases} r_i & e_i = 0 \\ v_i + r_i & e_i = 1 \end{cases}$$

$\sum_{i=1}^{n} z_i$

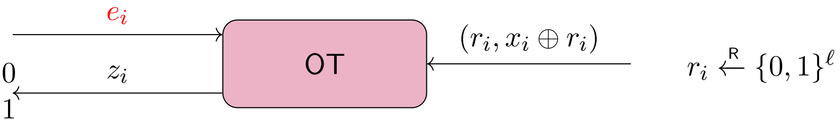
receiver obtains $|X \cap Y|$

sender obtains $\sum_{x_i \in Y} v_i = \sum_{i=1}^{n} z_i - \sum_{i=1}^{n} r_i$

PSO from mqRPMT



yields PSI-card-secret-share coupled with OT and masking trick

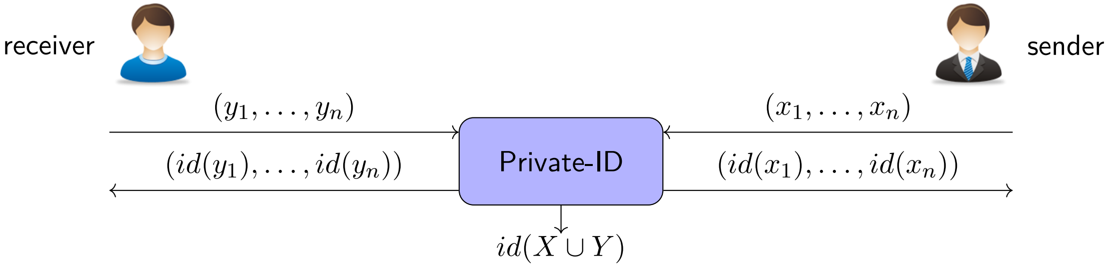


$$z_i = \begin{cases} r_i & e_i = 0 \\ x_i \oplus r_i & e_i = 1 \end{cases}$$

receiver obtains $|X \cap Y|$ and z_i

sender has $x_i \oplus r_i$

Private-ID



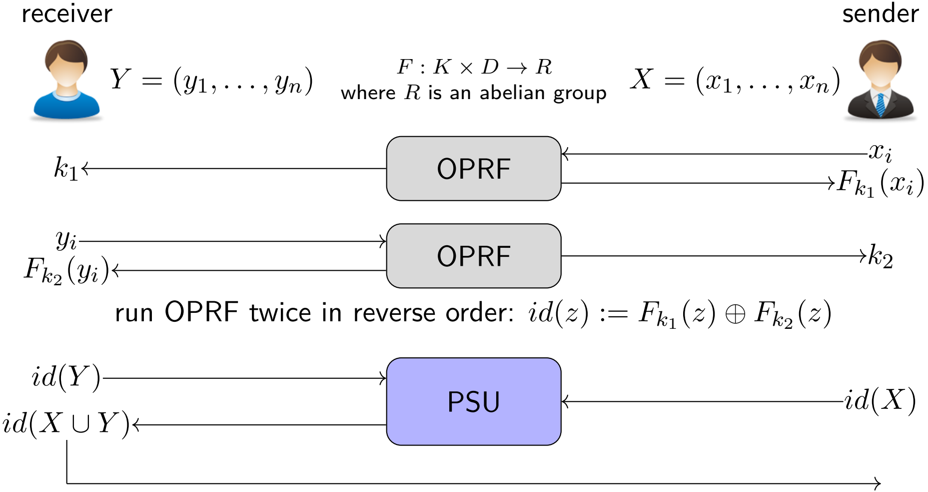
Buddhavarapu et al. [BKM+20] proposed private-ID:

- assigns two parties a random identifier per item
- each party obtains identifiers to his own set, as well as identifiers of the union

With private-ID, two parties can sort their private set w.r.t. a global set of identifiers, and then can proceed any desired private computation item by item, being assured that identical items are aligned.

Prior Construction of Private-ID

[BKM$^+$20] gave a concrete DDH-based protocol. [GMR$^+$21] showed how to build private-ID from OPRF and PSU.



receiver

sender

$Y = (y_1, \ldots, y_n)$

$F : K \times D \to R$
where R is an abelian group

$X = (x_1, \ldots, x_n)$

k_1

OPRF

x_i

$F_{k_1}(x_i)$

y_i

$F_{k_2}(y_i)$

OPRF

k_2

run OPRF twice in reverse order: $id(z) := F_{k_1}(z) \oplus F_{k_2}(z)$

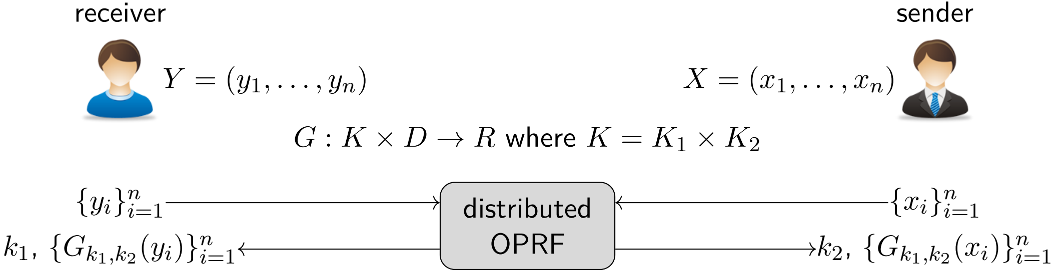
$id(Y)$

$id(X \cup Y)$

PSU

$id(X)$

Our Construction of Private-ID



receiver

$Y = (y_1, \dots, y_n)$

sender

$X = (x_1, \dots, x_n)$

$G : K \times D \to R$ where $K = K_1 \times K_2$

$\{y_i\}_{i=1}^n$

distributed OPRF

$\{x_i\}_{i=1}^n$

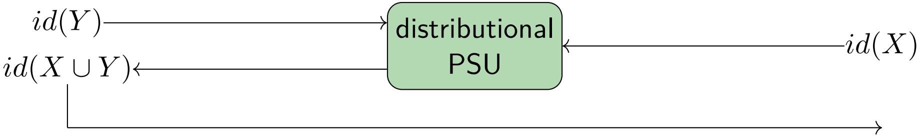
$k_1, \{G_{k_1,k_2}(y_i)\}_{i=1}^n$

$k_2, \{G_{k_1,k_2}(x_i)\}_{i=1}^n$

set $id(z) = G_{k_1,k_2}(z)$

standard notion are defined w.r.t. any private inputs $\rightsquigarrow$ arbitrary protocol composition

relaxed notion w.r.t. distribution of private inputs $\rightsquigarrow$ efficiency improvement



$id(Y)$

distributional PSU

$id(X)$

$id(X \cup Y)$

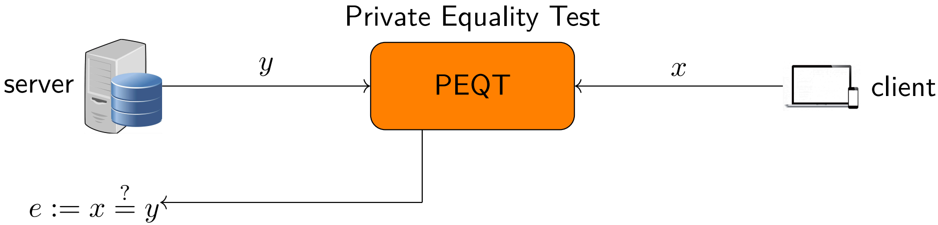
Outline

1. PSO Framework from mqRPMT
2. Construction of mqRPMT
    - 1st Construction from Commutative Weak PRF
    - 2nd Construction from Permuted Oblivious PRF
    - Connection Between mqPMT and mqRPMT
3. Comparison and Experimentation
4. Summary

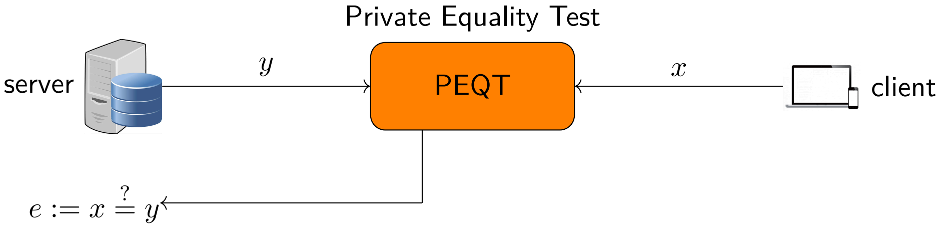
Outline

1. PSO Framework from mqRPMT
2. Construction of mqRPMT
   - 1st Construction from Commutative Weak PRF
   - 2nd Construction from Permuted Oblivious PRF
   - Connection Between mqPMT and mqRPMT
3. Comparison and Experimentation
4. Summary

Starting Point: PEQT



Starting Point: PEQT

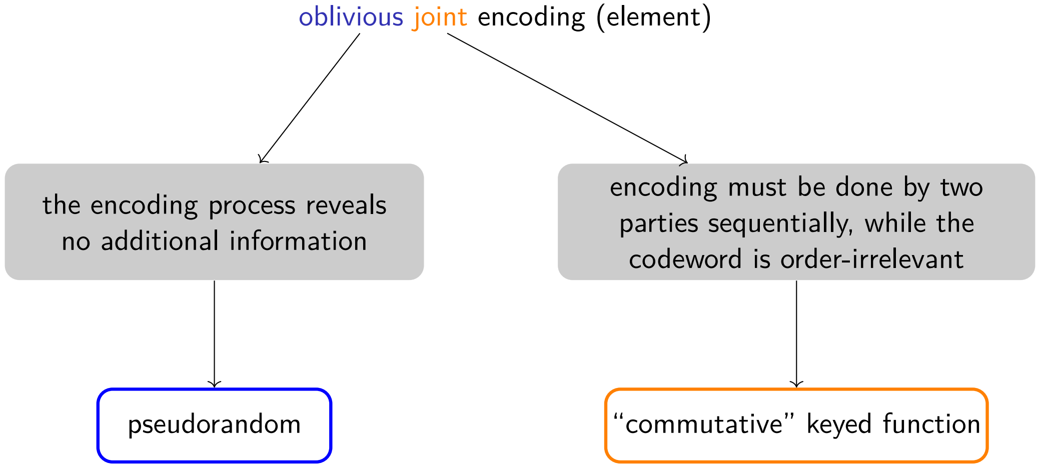


Observation: PEQT is not only an extreme case of mqPMT, but also an extreme case of mqRPMT

Goal: build PEQT amenable to extension:

$$y \rightsquigarrow Y = \{y_1, \ldots, y_m\},\ x \rightsquigarrow X = \{x_1, \ldots, x_n\},\ e \rightsquigarrow \vec{e} = (e_1, \ldots, e_n)$$

High-level Idea



oblivious joint encoding (element)

the encoding process reveals no additional information

encoding must be done by two parties sequentially, while the codeword is order-irrelevant

pseudorandom

“commutative” keyed function

Commutative Weak PRF

We first formally define two standard properties for keyed functions.

Composable. For a family of keyed functions $F : K \times D \to R$, F is 2-composable if $R \subseteq D$ (special case $R = D$) $\rightsquigarrow F_{k_1}(F_{k_2}(\cdot))$ is well-defined.

Commutative. A family of composable keyed functions is commutative if:

$$\forall k_1, k_2 \in K, \forall x \in D : F_{k_1}(F_{k_2}(x)) = F_{k_2}(F_{k_1}(x))$$

Commutative Weak PRF

We first formally define two standard properties for keyed functions.

Composable. For a family of keyed functions $F : K \times D \to R$, F is 2-composable if $R \subseteq D$ (special case $R = D$) $\leadsto F_{k_1}(F_{k_2}(\cdot))$ is well-defined.

Commutative. A family of composable keyed functions is commutative if:

$$\forall k_1, k_2 \in K, \forall x \in D : F_{k_1}(F_{k_2}(x)) = F_{k_2}(F_{k_1}(x))$$

Definition 1 (Commutative Weak PRF)

$F : K \times D \to D$ is cwPRF if it satisfies weak pseudorandomness ($k \xleftarrow{\mathsf{R}} K$, $x \xleftarrow{\mathsf{R}} X$) and commutative property simultaneously. When F is a permutation, we say F is cwPRP.

Commutative Weak PRF

We first formally define two standard properties for keyed functions.

Composable. For a family of keyed functions $F : K \times D \to R$, F is 2-composable if $R \subseteq D$ (special case $R = D$) $\rightsquigarrow F_{k_1}(F_{k_2}(\cdot))$ is well-defined.

Commutative. A family of composable keyed functions is commutative if:

$$\forall k_1, k_2 \in K, \forall x \in D : F_{k_1}(F_{k_2}(x)) = F_{k_2}(F_{k_1}(x))$$

Definition 1 (Commutative Weak PRF)

$F : K \times D \to D$ is cwPRF if it satisfies weak pseudorandomness ($k \xleftarrow{\mathsf{R}} K$, $x \xleftarrow{\mathsf{R}} X$) and commutative property simultaneously. When F is a permutation, we say F is cwPRP.

Why merely weak pseudorandomness?

Commutativity denies standard pseudorandomness. Consider the following attack:

- $\mathcal{A}$ picks $k' \xleftarrow{\mathsf{R}} K$, $x \xleftarrow{\mathsf{R}} D$, queries the real-or-random oracle at point $F_{k'}(x)$ and x, receiving y' and y. $\mathcal{A}$ then outputs '1' iff $F_{k'}(y) = y'$

$$F_{k'}(y = F_k(x)) = F_k(F_{k'}(x)) = y'$$

Construction of cwPRF

Construction (DDH-based cwPRF)

- Setup(1^κ): runs GroupGen$(1^\kappa) \to (\mathbb{G}, g, p)$, output $pp = (\mathbb{G}, g, p)$ which defines
$$F : \mathbb{Z}_p \times \mathbb{G} \to \mathbb{G} \text{ as } F_k(x) := x^k$$
- KeyGen(pp): outputs $k \xleftarrow{\mathsf{R}} \mathbb{Z}_p$.
- Eval(k, x): on input $k \in \mathbb{Z}_p$ and $x \in \mathbb{G}$, outputs x^k.

DDH assumption $\Rightarrow$ weak pseudorandomness

Commutativity: $\forall k_1, k_2 \in K$ and $\forall x \in D$: $F_{k_1}(F_{k_2}(x)) = x^{k_1 k_2} = F_{k_2}(F_{k_1}(x))$

cwPRF is the “right” cryptographic abstraction of the classic DH function

Post-quantum Secure cwPRF

cwPRF can be analogously built from weak pseudorandom efficient group action, which is in turn based on supersingular isogeny assumption.

- Supersingular isogeny is still believed to be post-quantum secure so far, but its presumed post-quantum security is shaky.

Post-quantum Secure cwPRF

cwPRF can be analogously built from weak pseudorandom efficient group action, which is in turn based on supersingular isogeny assumption.

- Supersingular isogeny is still believed to be post-quantum secure so far, but its presumed post-quantum security is shaky.

Can we build cwPRF from lattice-based assumption?

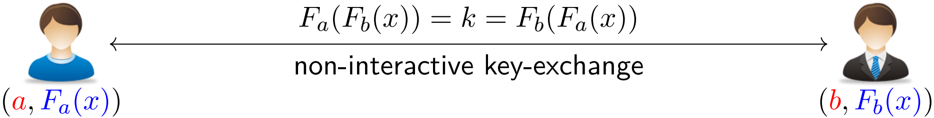
Post-quantum Secure cwPRF

cwPRF can be analogously built from weak pseudorandom efficient group action, which is in turn based on supersingular isogeny assumption.

- Supersingular isogeny is still believed to be post-quantum secure so far, but its presumed post-quantum security is shaky.

Can we build cwPRF from lattice-based assumption?

Note that cwPRF ⇒ NIKE.



$$F_a(F_b(x)) = k = F_b(F_a(x))$$

non-interactive key-exchange

$(a, F_a(x))$

$(b, F_b(x))$

A recent result of Guo et al. [GKRS22] indicated that it would be difficult to construct NIKE from lattice-based assumptions.

giving lattice-based cwPRF or proving impossibility will lead to progress on some other well-studied questions in cryptography

Randomness Enhancement

But what we need for mqRPMT is standard pseudorandomness.

Solution: hash-then-evaluate

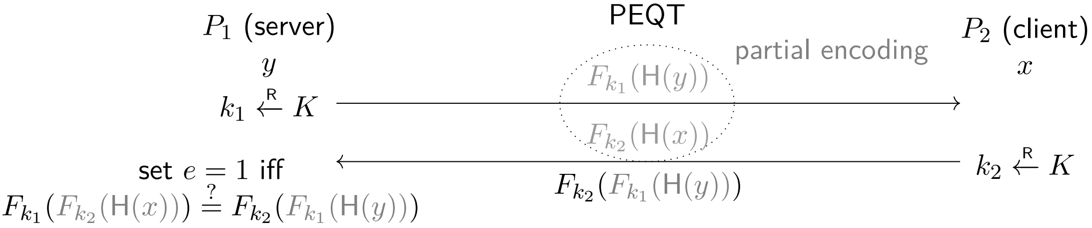
- Domain extension: handle arbitrary domain $X = \{0,1\}^*$
- Randomness amplification: weak $\rightsquigarrow$ standard

$$\begin{tikzcd}
X \arrow[r, "\text{random oracle } \mathsf{H}", "\text{randomness amplification}"'] \arrow[rr, bend left, "\text{standard PRF } F_k(\mathsf{H}(\cdot)) : K \times X \to R"] & D \arrow[r, "\text{weak PRF } F_k(\cdot)"] & R
\end{tikzcd}$$

Commutativity still holds w.r.t. H (suffice for mqRPMT)

$$F_{k_1}(F_{k_2}(\mathsf{H}(x))) = F_{k_2}(F_{k_1}(\mathsf{H}(x)))$$

mqRPMT from cwPRF



PEQT

P_1 (server)

y

$k_1 \xleftarrow{\mathsf{R}} K$

partial encoding

P_2 (client)

x

$F_{k_1}(\mathsf{H}(y))$

$F_{k_2}(\mathsf{H}(x))$

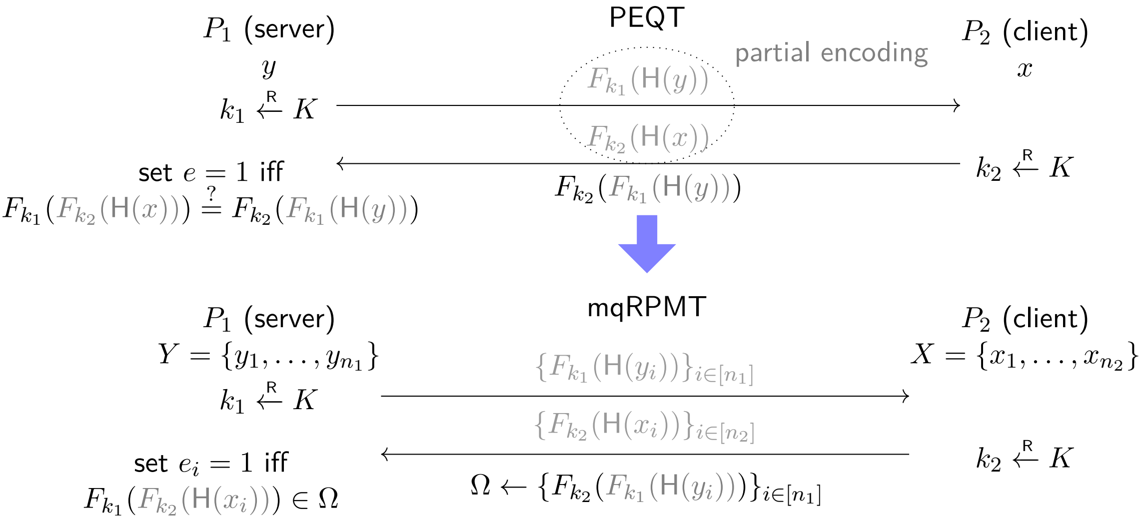
$k_2 \xleftarrow{\mathsf{R}} K$

$F_{k_2}(F_{k_1}(\mathsf{H}(y)))$

set $e = 1$ iff

$F_{k_1}(F_{k_2}(\mathsf{H}(x))) \stackrel{?}{=} F_{k_2}(F_{k_1}(\mathsf{H}(y)))$

mqRPMT from cwPRF



PEQT

P_1 (server)

y

$k_1 \xleftarrow{R} K$

partial encoding

P_2 (client)

x

$F_{k_1}(\mathsf{H}(y))$

$F_{k_2}(\mathsf{H}(x))$

$k_2 \xleftarrow{R} K$

$F_{k_2}(F_{k_1}(\mathsf{H}(y)))$

set $e = 1$ iff

$F_{k_1}(F_{k_2}(\mathsf{H}(x))) \overset{?}{=} F_{k_2}(F_{k_1}(\mathsf{H}(y)))$

mqRPMT

P_1 (server)

$Y = \{y_1, \dots, y_{n_1}\}$

$k_1 \xleftarrow{R} K$

P_2 (client)

$X = \{x_1, \dots, x_{n_2}\}$

$\{F_{k_1}(\mathsf{H}(y_i))\}_{i \in [n_1]}$

$\{F_{k_2}(\mathsf{H}(x_i))\}_{i \in [n_2]}$

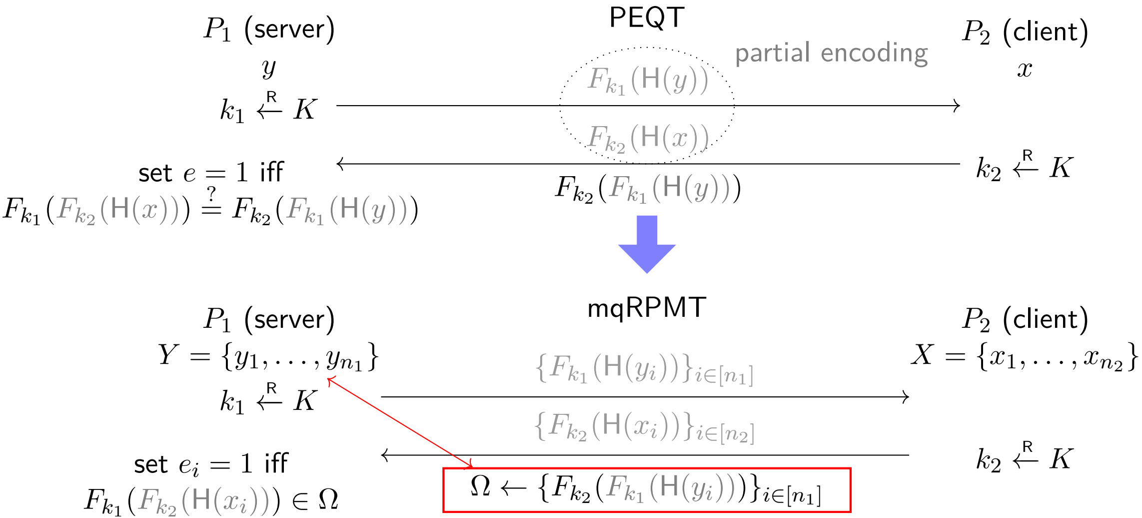
$k_2 \xleftarrow{R} K$

$\Omega \leftarrow \{F_{k_2}(F_{k_1}(\mathsf{H}(y_i)))\}_{i \in [n_1]}$

set $e_i = 1$ iff

$F_{k_1}(F_{k_2}(\mathsf{H}(x_i))) \in \Omega$

mqRPMT from cwPRF



PEQT

P_1 (server)

y

$k_1 \xleftarrow{R} K$

set $e = 1$ iff

$F_{k_1}(F_{k_2}(\mathsf{H}(x))) \overset{?}{=} F_{k_2}(F_{k_1}(\mathsf{H}(y)))$

$\xrightarrow{F_{k_1}(\mathsf{H}(y))}$

$\xleftarrow{F_{k_2}(\mathsf{H}(x))}$

$F_{k_2}(F_{k_1}(\mathsf{H}(y)))$

partial encoding

P_2 (client)

x

$k_2 \xleftarrow{R} K$

mqRPMT

P_1 (server)

$Y = \{y_1, \ldots, y_{n_1}\}$

$k_1 \xleftarrow{R} K$

set $e_i = 1$ iff

$F_{k_1}(F_{k_2}(\mathsf{H}(x_i))) \in \Omega$

$\xrightarrow{\{F_{k_1}(\mathsf{H}(y_i))\}_{i \in [n_1]}}$

$\xleftarrow{\{F_{k_2}(\mathsf{H}(x_i))\}_{i \in [n_2]}}$

$\boxed{\Omega \leftarrow \{F_{k_2}(F_{k_1}(\mathsf{H}(y_i)))\}_{i \in [n_1]}}$

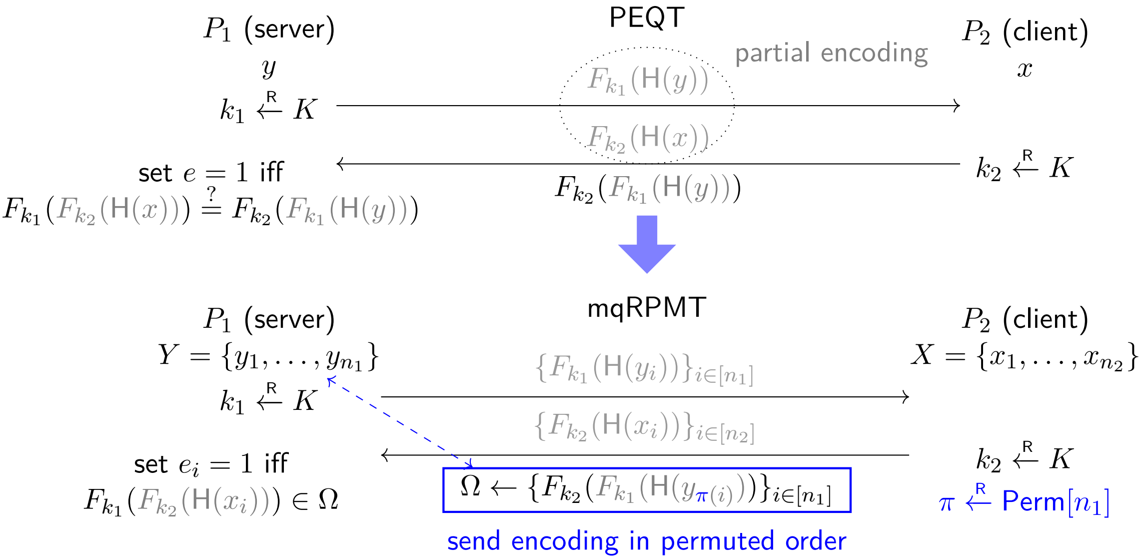
P_2 (client)

$X = \{x_1, \ldots, x_{n_2}\}$

$k_2 \xleftarrow{R} K$

send encoding in order reveals intersection!

mqRPMT from cwPRF



PEQT

P_1 (server)

y

$k_1 \xleftarrow{R} K$

$F_{k_1}(\mathsf{H}(y))$

partial encoding

$F_{k_2}(\mathsf{H}(x))$

P_2 (client)

x

$k_2 \xleftarrow{R} K$

$F_{k_2}(F_{k_1}(\mathsf{H}(y)))$

set $e = 1$ iff

$F_{k_1}(F_{k_2}(\mathsf{H}(x))) \overset{?}{=} F_{k_2}(F_{k_1}(\mathsf{H}(y)))$

mqRPMT

P_1 (server)

$Y = \{y_1, \ldots, y_{n_1}\}$

$k_1 \xleftarrow{R} K$

$\{F_{k_1}(\mathsf{H}(y_i))\}_{i \in [n_1]}$

$\{F_{k_2}(\mathsf{H}(x_i))\}_{i \in [n_2]}$

P_2 (client)

$X = \{x_1, \ldots, x_{n_2}\}$

$k_2 \xleftarrow{R} K$

$\pi \xleftarrow{R} \mathsf{Perm}[n_1]$

$\boxed{\Omega \leftarrow \{F_{k_2}(F_{k_1}(\mathsf{H}(y_{\pi(i)})))\}_{i \in [n_1]}}$

send encoding in permuted order

set $e_i = 1$ iff

$F_{k_1}(F_{k_2}(\mathsf{H}(x_i))) \in \Omega$

mqRPMT from cwPRF

PEQT

P_1 (server) | P_2 (client)

y | x

$k_1 \xleftarrow{R} K$ $\xrightarrow{F_{k_1}(\mathsf{H}(y))}$

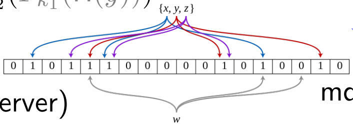
partial encoding

$\xleftarrow{F_{k_2}(\mathsf{H}(x))}$ $k_2 \xleftarrow{R} K$

$F_{k_2}(F_{k_1}(\mathsf{H}(y)))$

set $e = 1$ iff

$$F_{k_1}(F_{k_2}(\mathsf{H}(x))) \overset{?}{=} F_{k_2}(F_{k_1}(\mathsf{H}(y)))$$



mqRPMT

P_1 (server) | P_2 (client)

$Y = \{y_1, \ldots, y_{n_1}\}$ | $X = \{x_1, \ldots, x_{n_2}\}$

$k_1 \xleftarrow{R} K$ $\xrightarrow{\{F_{k_1}(\mathsf{H}(y_i))\}_{i \in [n_1]}}$

$\xleftarrow{\{F_{k_2}(\mathsf{H}(x_i))\}_{i \in [n_2]}}$ $k_2 \xleftarrow{R} K$

$$\boxed{\Omega \leftarrow \mathsf{BloomFilter}(\{F_{k_2}(F_{k_1}(\mathsf{H}(y_i)))\}_{i \in [n_1]})}$$

set $e_i = 1$ iff

$$F_{k_1}(F_{k_2}(\mathsf{H}(x_i))) \in \Omega$$

more space efficient: admit small false positive probability

Complexity Analysis

Consider the balanced setting: $n_1 = n_2 = n$

Table: Complexity of cwPRF-based mqRPMT.

Computation	$4n\times\ F_k(\cdot) + 2n \times \mathsf{H}(\cdot)$ hash-to-domain
Communication	$3n \times \lvert D\rvert$ or $2n \times \lvert D\rvert + n \cdot 1.44\lambda$ $(\ll \lvert D\rvert)$

cwPRF-based mqRPMT is optimal in the sense that both computation and communication complexities are strictly linear in n

Instantiating the PSO framework with cwPRF-based mqRPMT, DDH assumption strikes back with the first strictly linear PSU protocol

incredibly simple and efficient

Outline

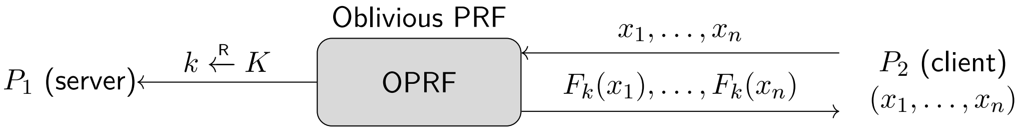
1. PSO Framework from mqRPMT

2. Construction of mqRPMT
    - 1st Construction from Commutative Weak PRF
    - 2nd Construction from Permuted Oblivious PRF
    - Connection Between mqPMT and mqRPMT

3. Comparison and Experimentation

4. Summary

Starting Point: mqPMT/PSI from OPRF



Oblivious PRF

P_1 (server)

$k \xleftarrow{R} K$

OPRF

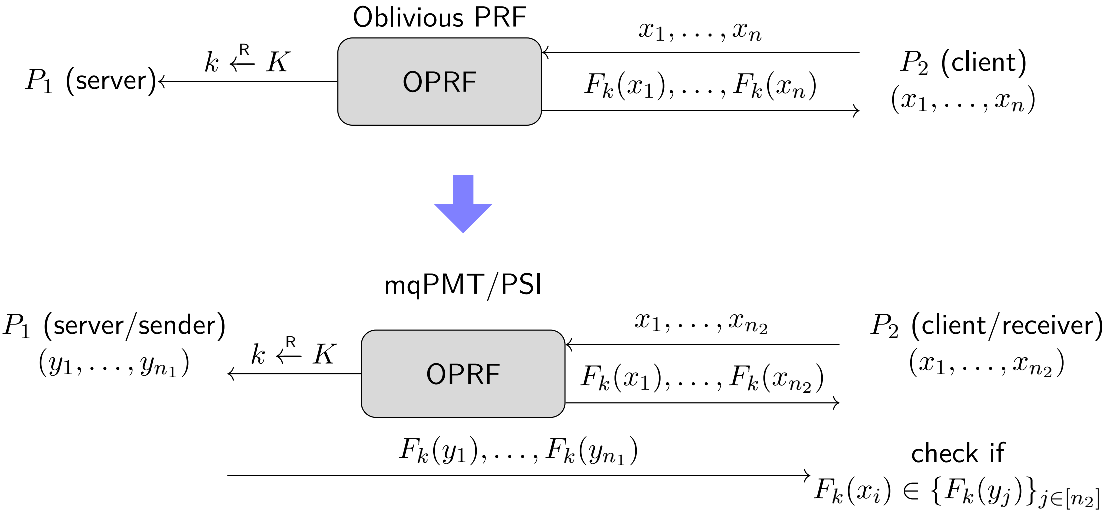
$x_1, \ldots, x_n$

$F_k(x_1), \ldots, F_k(x_n)$

P_2 (client)

$(x_1, \ldots, x_n)$

Starting Point: mqPMT/PSI from OPRF



Oblivious PRF

P_1 (server)

$k \xleftarrow{\mathsf{R}} K$

OPRF

$x_1, \ldots, x_n$

$F_k(x_1), \ldots, F_k(x_n)$

P_2 (client)

$(x_1, \ldots, x_n)$

mqPMT/PSI

P_1 (server/sender)

$(y_1, \ldots, y_{n_1})$

$k \xleftarrow{\mathsf{R}} K$

OPRF

$x_1, \ldots, x_{n_2}$

$F_k(x_1), \ldots, F_k(x_{n_2})$

P_2 (client/receiver)

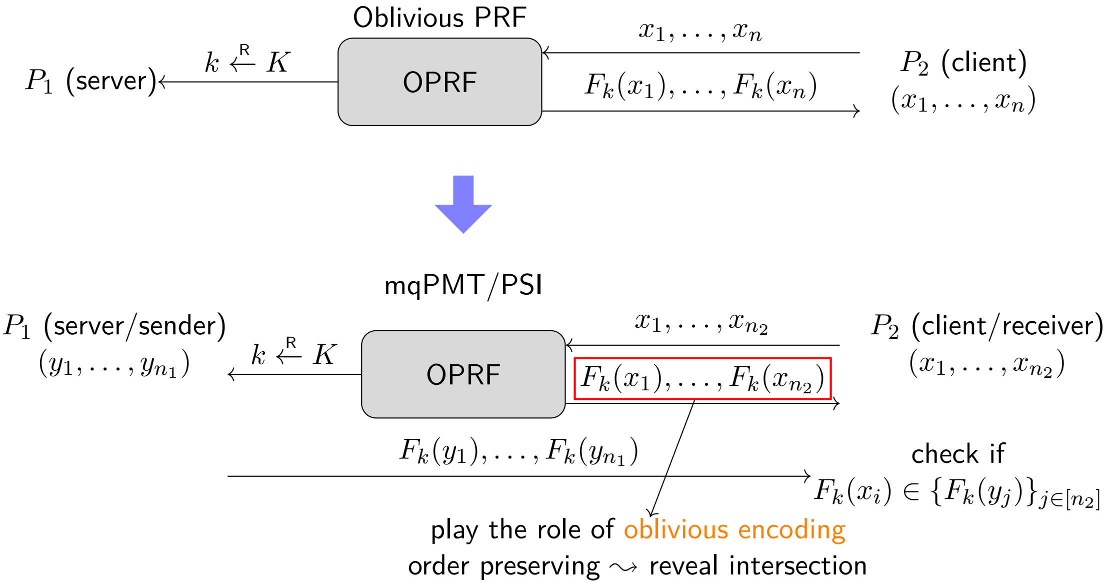
$(x_1, \ldots, x_{n_2})$

$F_k(y_1), \ldots, F_k(y_{n_1})$

check if

$F_k(x_i) \in \{F_k(y_j)\}_{j \in [n_2]}$

Starting Point: mqPMT/PSI from OPRF



Oblivious PRF

P_1 (server)

$k \xleftarrow{R} K$

OPRF

$x_1, \ldots, x_n$

$F_k(x_1), \ldots, F_k(x_n)$

P_2 (client)

$(x_1, \ldots, x_n)$

mqPMT/PSI

P_1 (server/sender)

$(y_1, \ldots, y_{n_1})$

$k \xleftarrow{R} K$

OPRF

$x_1, \ldots, x_{n_2}$

$\boxed{F_k(x_1), \ldots, F_k(x_{n_2})}$

P_2 (client/receiver)

$(x_1, \ldots, x_{n_2})$

$F_k(y_1), \ldots, F_k(y_{n_1})$

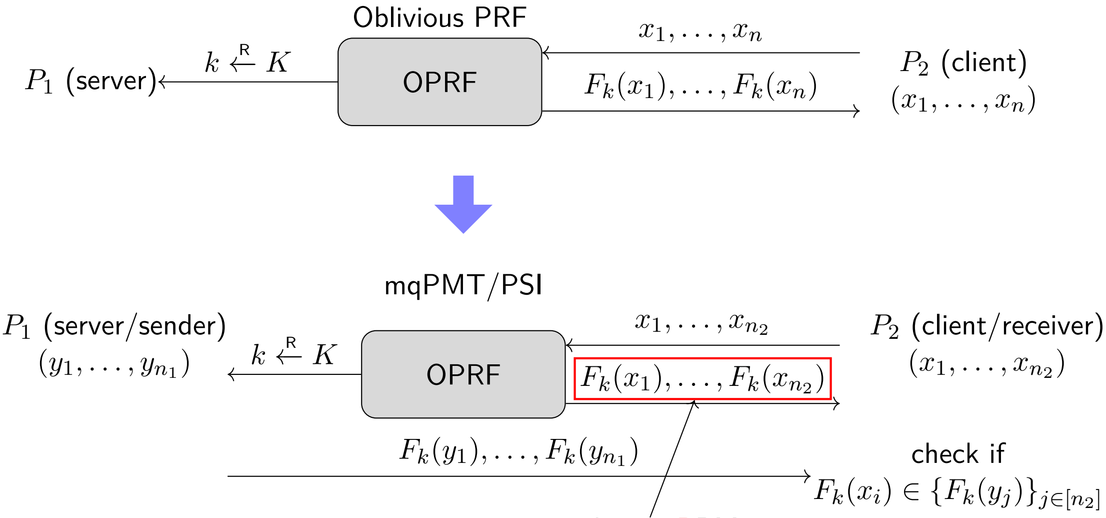
check if

$F_k(x_i) \in \{F_k(y_j)\}_{j \in [n_2]}$

play the role of oblivious encoding

order preserving $\rightsquigarrow$ reveal intersection

Starting Point: mqPMT/PSI from OPRF



Oblivious PRF

P_1 (server)

$k \overset{R}{\leftarrow} K$

OPRF

$x_1, \ldots, x_n$

$F_k(x_1), \ldots, F_k(x_n)$

P_2 (client)

$(x_1, \ldots, x_n)$

mqPMT/PSI

P_1 (server/sender)

$(y_1, \ldots, y_{n_1})$

$k \overset{R}{\leftarrow} K$

OPRF

$x_1, \ldots, x_{n_2}$

$\boxed{F_k(x_1), \ldots, F_k(x_{n_2})}$

P_2 (client/receiver)

$(x_1, \ldots, x_{n_2})$

$F_k(y_1), \ldots, F_k(y_{n_1})$

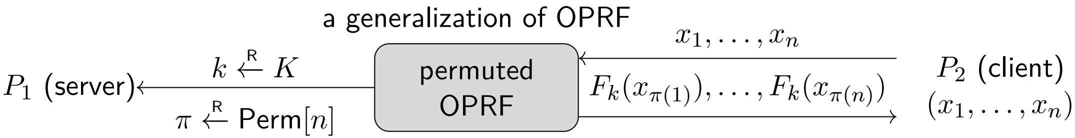
check if

$F_k(x_i) \in \{F_k(y_j)\}_{j \in [n_2]}$

to enable mqRPMT

we need permuted oblivious encoding

mqRPMT from Permuted OPRF



a generalization of OPRF

P_1 (server)

$k \xleftarrow{R} K$

$\pi \xleftarrow{R} \mathsf{Perm}[n]$

permuted OPRF

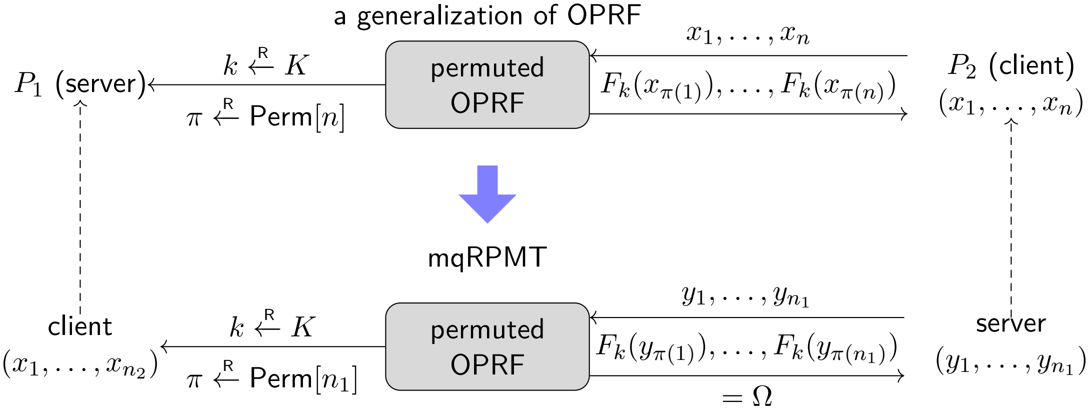
$x_1, \ldots, x_n$

$F_k(x_{\pi(1)}), \ldots, F_k(x_{\pi(n)})$

P_2 (client)

$(x_1, \ldots, x_n)$

mqRPMT from Permuted OPRF



a generalization of OPRF

P_1 (server)

$k \xleftarrow{R} K$

$\pi \xleftarrow{R} \mathsf{Perm}[n]$

permuted OPRF

$x_1, \ldots, x_n$

$F_k(x_{\pi(1)}), \ldots, F_k(x_{\pi(n)})$

P_2 (client)

$(x_1, \ldots, x_n)$

mqRPMT

client

$(x_1, \ldots, x_{n_2})$

$k \xleftarrow{R} K$

$\pi \xleftarrow{R} \mathsf{Perm}[n_1]$

permuted OPRF

$y_1, \ldots, y_{n_1}$

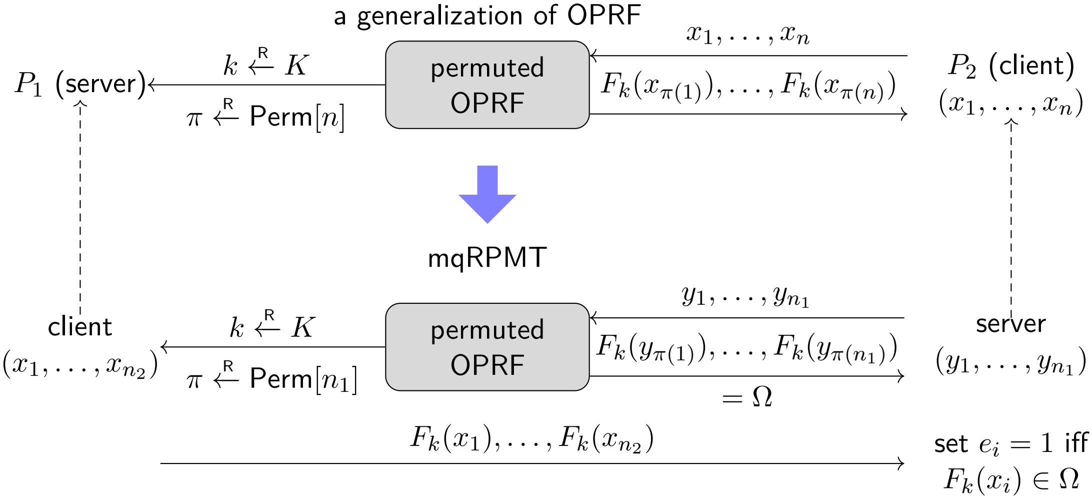
$F_k(y_{\pi(1)}), \ldots, F_k(y_{\pi(n_1)})$

$= \Omega$

server

$(y_1, \ldots, y_{n_1})$

mqRPMT from Permuted OPRF



a generalization of OPRF

P_1 (server)

$k \xleftarrow{\mathsf{R}} K$

$\pi \xleftarrow{\mathsf{R}} \mathsf{Perm}[n]$

permuted OPRF

$x_1, \dots, x_n$

$F_k(x_{\pi(1)}), \dots, F_k(x_{\pi(n)})$

P_2 (client)

$(x_1, \dots, x_n)$

mqRPMT

client

$(x_1, \dots, x_{n_2})$

$k \xleftarrow{\mathsf{R}} K$

$\pi \xleftarrow{\mathsf{R}} \mathsf{Perm}[n_1]$

permuted OPRF

$y_1, \dots, y_{n_1}$

$F_k(y_{\pi(1)}), \dots, F_k(y_{\pi(n_1)})$

$= \Omega$

server

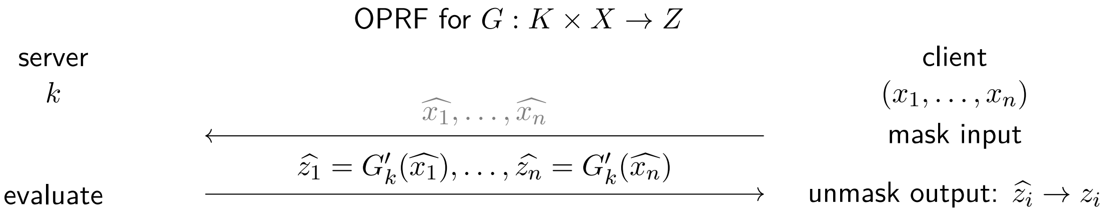
$(y_1, \dots, y_{n_1})$

$F_k(x_1), \dots, F_k(x_{n_2})$

set $e_i = 1$ iff $F_k(x_i) \in \Omega$

Build Permuted OPRF from cwPRP

A common approach to build OPRF is “mask-then-unmask” via homomorphism



OPRF for $G : K \times X \to Z$

server

k

client

$(x_1, \ldots, x_n)$

mask input

$\widehat{x_1}, \ldots, \widehat{x_n}$

$\xleftarrow{\hspace{10em}}$

evaluate

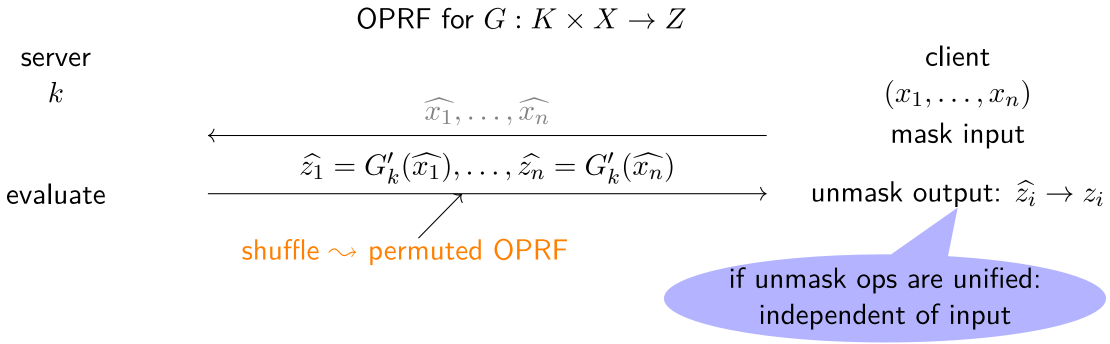
$\widehat{z_1} = G'_k(\widehat{x_1}), \ldots, \widehat{z_n} = G'_k(\widehat{x_n})$

$\xrightarrow{\hspace{10em}}$

unmask output: $\widehat{z_i} \to z_i$

Build Permuted OPRF from cwPRP

A common approach to build OPRF is "mask-then-unmask" via homomorphism



OPRF for $G : K \times X \to Z$

server

k

client

$(x_1, \ldots, x_n)$

mask input

$\widehat{x_1}, \ldots, \widehat{x_n}$

$\widehat{z_1} = G'_k(\widehat{x_1}), \ldots, \widehat{z_n} = G'_k(\widehat{x_n})$

evaluate

unmask output: $\widehat{z_i} \to z_i$

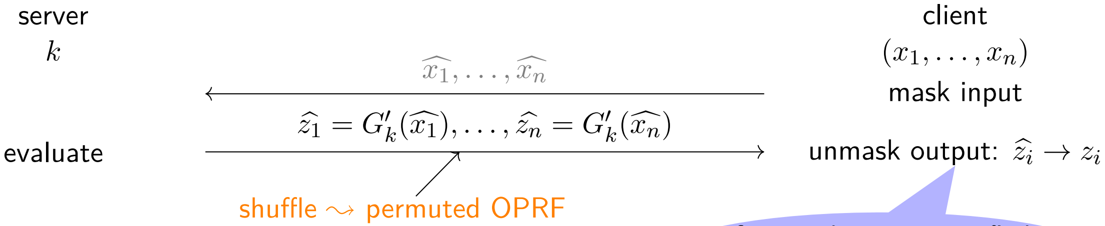
shuffle $\rightsquigarrow$ permuted OPRF

if unmask ops are unified:
independent of input

Build Permuted OPRF from cwPRP

A common approach to build OPRF is “mask-then-unmask” via homomorphism

OPRF for $G : K \times X \to Z$



server

k

evaluate

client

$(x_1, \ldots, x_n)$

mask input

$\widehat{x_1}, \ldots, \widehat{x_n}$

$\widehat{z_1} = G'_k(\widehat{x_1}), \ldots, \widehat{z_n} = G'_k(\widehat{x_n})$

unmask output: $\widehat{z_i} \to z_i$

shuffle ⇝ permuted OPRF

if unmask ops are unified: independent of input

cwPRP enables simplest unified mask-then-unmask

mask: $\hat{x} \leftarrow F_s(\mathsf{H}(x))$

evaluate: $\hat{z} \leftarrow F_k(\hat{x})$

unmask: $z \leftarrow F_s^{-1}(F_k(\hat{x})) = F_k(F_s^{-1}(\hat{x})) = F_k(\mathsf{H}(x))$

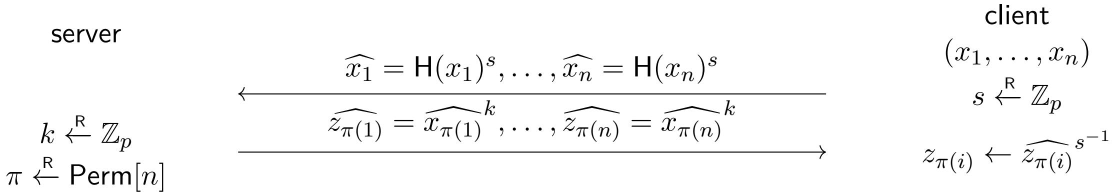
Permuted OPRF from DDH-based cwPRP

Observe that the DDH-based cwPRF is actually a cwPRP $F : \mathbb{Z}_p \times \mathbb{G} \to \mathbb{G}$.

- combine $\mathsf{H} : \{0,1\}^* \to \mathbb{G} \Rightarrow$ permuted OPRF protocol for $G : \mathbb{Z}_p \times \{0,1\}^* \to \mathbb{G}$ defined as $G_k(x) = F_k(\mathsf{H}(x))$.

---

pOPRF for $G_k(x) = F_k(\mathsf{H}(x))$



server

client

$(x_1, \ldots, x_n)$

$s \xleftarrow{\mathsf{R}} \mathbb{Z}_p$

$$\xleftarrow{\widehat{x_1} = \mathsf{H}(x_1)^s, \ldots, \widehat{x_n} = \mathsf{H}(x_n)^s}$$

$k \xleftarrow{\mathsf{R}} \mathbb{Z}_p$

$\pi \xleftarrow{\mathsf{R}} \mathsf{Perm}[n]$

$$\xrightarrow{\widehat{z_{\pi(1)}} = \widehat{x_{\pi(1)}}^k, \ldots, \widehat{z_{\pi(n)}} = \widehat{x_{\pi(n)}}^k}$$

$z_{\pi(i)} \leftarrow \widehat{z_{\pi(i)}}^{s^{-1}}$

Comparison of mqRPMT from cwPRF and pOPRF

Primitive	Assumption	implied by X25519	Bloom filter optimization
cwPRF	DDH	✓	✓
pOPRF	DDH	✗	✗

the pOPRF-based mqRPMT is more of theoretical interest

- It can be viewed as a counterpart of OPRF-based mqPMT construction
- So far, we only know how to build pOPRF based on assumptions with nice algebra structure, but not from fast primitives such as OT or VOLE.
    - This somehow explains the efficiency gap between mqPMT and mqRPMT.

Outline

1. PSO Framework from mqRPMT

2. Construction of mqRPMT
   - 1st Construction from Commutative Weak PRF
   - 2nd Construction from Permuted Oblivious PRF
   - Connection Between mqPMT and mqRPMT

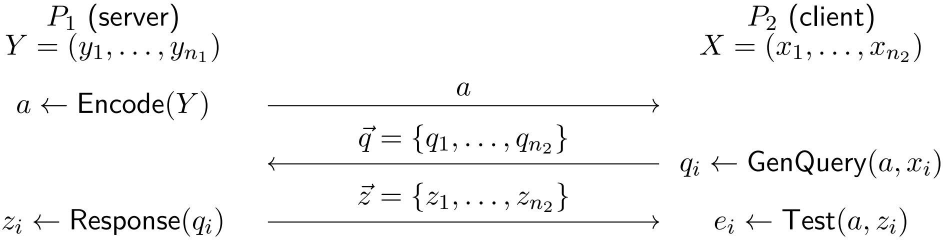
3. Comparison and Experimentation

4. Summary

Sigma-mqPMT

Given the efficiency gap between PSI and other PSO protocols, it is intriguing to study the connection between mqPMT and mqRPMT.

- Towards this goal, we first abstract a category of mqPMT called Sigma-mqPMT.



P_1 (server)
$Y = (y_1, \ldots, y_{n_1})$

P_2 (client)
$X = (x_1, \ldots, x_{n_2})$

$a \leftarrow$ Encode(Y) $\xrightarrow{a}$

$\xleftarrow{\vec{q} = \{q_1, \ldots, q_{n_2}\}}$ $q_i \leftarrow$ GenQuery(a, x_i)

$z_i \leftarrow$ Response(q_i) $\xrightarrow{\vec{z} = \{z_1, \ldots, z_{n_2}\}}$ $e_i \leftarrow$ Test(a, z_i)

- **Reusable:** a (best interpreted as encoding of Y) can be safely reused.
- **Context-independent:** q_i is only related to a, x_i under test and P_2's randomness.
- **Stateless test:** Test algorithm can work without knowing (x_i, q_i).

mqRPMT* from Sigma-mqPMT

P_1 (server)
$Y = (y_1, \ldots, y_{n_1})$

P_2 (client)
$X = (x_1, \ldots, x_{n_2})$

$$a \leftarrow \mathsf{mqPMT.Encode}(Y) \quad \xrightarrow{\qquad a \qquad}$$

$$\xleftarrow{\vec{q} = \{q_1, \ldots, q_{n_2}\}} \quad q_i \leftarrow \mathsf{mqPMT.GenQuery}(a, x_i)$$

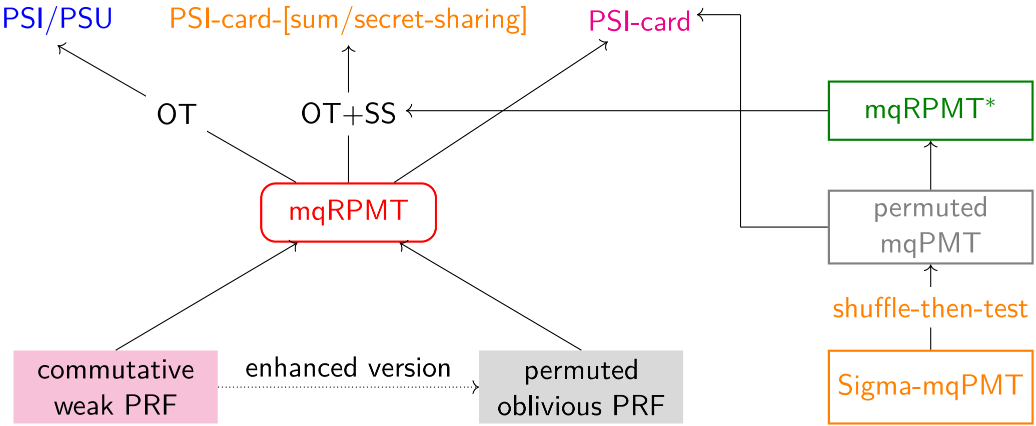
$$\pi \xleftarrow{\mathsf{R}} \mathsf{Perm}[n_2] \quad \xrightarrow{\vec{z}^* = \{z_{\pi(1)}, \ldots, z_{\pi(n_2)}\}}$$

$$\vec{e} = \{e^*_{\pi^{-1}(i)}\}_{i=1}^{n_2} \quad \xleftarrow{\vec{e}^* = \{e^*_1, \ldots, e^*_{n_2}\}} \quad e^*_i \leftarrow \mathsf{mqPMT.Test}(a, z^*_i)$$

Via the “permute-then-test” approach, we can tweak Sigma-mqPMT to mqRPMT* (additionally reveal intersection size to client).

- translate a category of PSI protocols (such as [Mea86, FIPR05, CLR17]) to other PSO protocols (allowing both parties learn the intersection size).
- make the initial step towards establishing the connection between mqRPMT and mqPMT.

Summary of Main Results



PSI/PSU

PSI-card-[sum/secret-sharing]

PSI-card

OT

OT+SS

mqRPMT*

mqRPMT

permuted mqPMT

shuffle-then-test

commutative weak PRF

enhanced version

permuted oblivious PRF

Sigma-mqPMT

Outline

1. PSO Framework from mqRPMT
2. Construction of mqRPMT
    - 1st Construction from Commutative Weak PRF
    - 2nd Construction from Permuted Oblivious PRF
    - Connection Between mqPMT and mqRPMT
3. Comparison and Experimentation
4. Summary

Cryptographic Engineering Matters

We implement our PSO framework via the following vein

EC groups DDH-based cwPRF $\rightsquigarrow$ mqRPMT $\rightsquigarrow$ PSO framework

Cryptographic Engineering Matters

We implement our PSO framework via the following vein

EC groups DDH-based cwPRF $\rightsquigarrow$ mqRPMT $\rightsquigarrow$ PSO framework

1. `NIST P-256` ♦ ▼ (also known as `secp256r1` and `prime256v1`)
   - hash-to-point operation is expensive $\approx$ non-fixed Exp
   - point compression halves communication cost

     $\rightsquigarrow$ point decompression is expensive $\approx$ non-fixed Exp

Cryptographic Engineering Matters

We implement our PSO framework via the following vein

EC groups DDH-based cwPRF $\rightsquigarrow$ mqRPMT $\rightsquigarrow$ PSO framework

1. `NIST P-256` ⧫ ▼ (also known as `secp256r1` and `prime256v1`)
   - hash-to-point operation is expensive $\approx$ non-fixed Exp
   - point compression halves communication cost
     $\rightsquigarrow$ point decompression is expensive $\approx$ non-fixed Exp
2. `Curve25519` ★ (*de facto* alternative of `NIST P-256`)
   - numerous merits: no backdoor, fast Exp, immunity against side-channel attacks
   - allow "Exp" with only X-coordinate $\rightsquigarrow$ halve communication & no decompression
   - any 32-byte bit array corresponds to the X-coordinate of a valid EC point $\rightsquigarrow$ hash-to-point operation is almost free

Cryptographic Engineering Matters

We implement our PSO framework via the following vein

EC groups DDH-based cwPRF $\rightsquigarrow$ mqRPMT $\rightsquigarrow$ PSO framework

① `NIST P-256` ⧫ ▼ (also known as `secp256r1` and `prime256v1`)

- hash-to-point operation is expensive $\approx$ non-fixed Exp
- point compression halves communication cost
  $\rightsquigarrow$ point decompression is expensive $\approx$ non-fixed Exp

② `Curve25519` ★ (*de facto* alternative of `NIST P-256`)

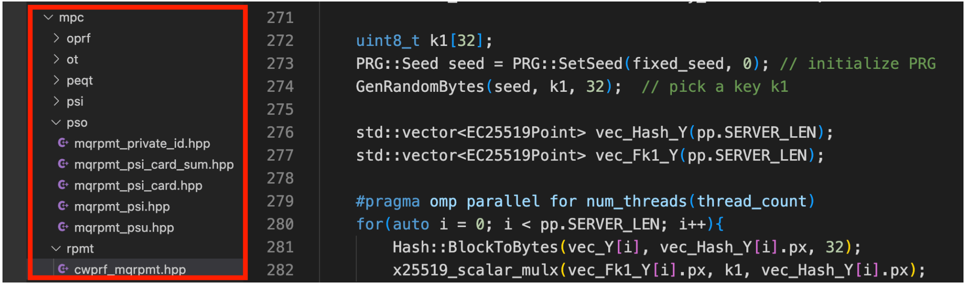
- numerous merits: no backdoor, fast Exp, immunity against side-channel attacks
- allow "Exp" with only X-coordinate $\rightsquigarrow$ halve communication & no decompression
- any 32-byte bit array corresponds to the X-coordinate of a valid EC point $\rightsquigarrow$ hash-to-point operation is almost free

For the first time, `Curve25519` fully unleashes its power in PSO area.
Correct the prejudice that "public-key operations are expensive":

- By leveraging optimized implementation, their performances are comparable with symmetric-key operations

Implementation Features

- Modular design: admit flexible combination to support various scenarios
- Minimum dependency: only require OpenSSL and OpenMP
- Multi-platforms: run smoothly on Linux and MacOS
- Rich functionality: support all PSO operations
- Highly parallelizable: scalable ⇝ support large-scale applications



```
mpc
  oprf
  ot
  peqt
  psi
  pso
    mqrpmt_private_id.hpp
    mqrpmt_psi_card_sum.hpp
    mqrpmt_psi_card.hpp
    mqrpmt_psi.hpp
    mqrpmt_psu.hpp
  rpmt
    cwprf_mqrpmt.hpp
```

```

    uint8_t k1[32];
    PRG::Seed seed = PRG::SetSeed(fixed_seed, 0); // initialize PRG
    GenRandomBytes(seed, k1, 32);  // pick a key k1

    std::vector<EC25519Point> vec_Hash_Y(pp.SERVER_LEN);
    std::vector<EC25519Point> vec_Fk1_Y(pp.SERVER_LEN);

    #pragma omp parallel for num_threads(thread_count)
    for(auto i = 0; i < pp.SERVER_LEN; i++){
        Hash::BlockToBytes(vec_Y[i], vec_Hash_Y[i].px, 32);
        x25519_scalar_mulx(vec_Fk1_Y[i].px, k1, vec_Hash_Y[i].px);
```

Implementation Details

Dev/Test environment	Other Parameters
CPU = Intel i7 2.50 GHZ	$\kappa = 128$, $\lambda = 40$
Physical core $= 8$	item length$= 128$ bits
RAM = 8GB	set sizes$= \{2^{12}, 2^{16}, 2^{20}\}$
OS = Ubuntu 20.04	LAN $= 10$Gbps, WAN $= 50$Mbps, RTT$= 80$ms

Protocols:

- mqRPMT, PSI, PSI-card, PSI-card-sum, PSU, Private-ID

Test items:

- Functionality
- Computation cost: total running time
- Communication cost: sum of two parties

Core protocol: mqRPMT

Protocol	T	Running time (s)						Comm. (MB)		
		LAN			WAN			total		
		2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
mqRPMT$^{\blacklozenge}$	1	0.50	7.20	114.16	1.39	9.68	136.27	0.52	8.35	133.6
	2	0.31	3.89	62.09	1.14	6.54	86.60			
	4	0.22	2.37	40.41	1.11	5.08	62.77			
Speedup		1.6-2.3 ×	1.9-3.0 ×	1.8-2.8 ×	1.2-1.3 ×	1.5-1.9 ×	1.6-2.2 ×	–	–	–
mqRPMT$^{\blacktriangledown}$	1	0.50	8.00	128.00	1.35	10.15	141.52	0.27	4.35	69.6
	2	0.32	5.05	80.69	1.18	7.11	94.19			
	4	0.23	3.54	58.40	1.08	5.54	71.26			
Speedup		1.6-2.2 ×	1.6-2.3 ×	1.6-2.2 ×	1.1-1.3×	1.4-1.8 ×	1.5-2 ×	–	–	–
mqRPMT$^{\bigstar}$	1	0.26	3.51	54.85	0.81	5.41	68.68	0.26	4.23	67.66
	2	0.15	1.79	28.24	0.75	3.83	41.38			
	4	0.10	1.07	15.32	0.72	3.09	28.31			
Speedup		1.7-2.6 ×	2.0-3.3 ×	1.9-3.6 ×	1.1-1.1 ×	1.4-1.8 ×	1.7-2.4 ×	–	–	–

strict linear complexity & high parallelism

2^{20} scale: #time < 15s using 4 threads on laptop, #communication < 70M

PSI: Performance and Comparison

PSI	Running time (s)						Comm. (MB)		
	LAN			WAN			total		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
[PRTY19]★	5.51	88.64	1418.20	5.82	90.79	1498.67	0.30	4.74	76.60
Our PSI⧫	**0.50**	**7.24**	**114.66**	**1.71**	**10.50**	**142.45**	**0.68**	**10.61**	**169.37**
Our PSI▼	**0.55**	**8.04**	**128.18**	**1.73**	**11.02**	**148.18**	**0.42**	**6.61**	**105.23**
Our PSI★	**0.29**	**3.56**	**55.11**	**1.19**	**6.38**	**75.56**	**0.41**	**6.48**	**103.31**
DH-PSI★	**0.22**	**3.39**	**54.79**	**0.92**	**5.57**	**69.31**	**0.28**	**4.57**	**74.1**

compared to existing DH-PSI implementation: # time speeds up $4.9\text{-}25.7\times$

PSI	Running time (ms)						Comm. (KB)		
	LAN			WAN			total		
	2^{8}	2^{9}	2^{10}	2^{8}	2^{9}	2^{10}	2^{8}	2^{9}	2^{10}
[RT21]★	50.0	71.0	147.3	224.1	260.2	457.9	17.9	34.1	66.3
Our PSI★	**41.9**	**69.5**	**99.3**	**577.0**	**582.9**	**646.1**	**38.6**	**63.5**	**113.3**
DH-PSI★	**16.49**	**31.80**	**56.91**	**210.42**	**227.33**	**252.32**	**18.48**	**36.68**	**72.8**

achieve the fastest speed in small set setting $(< 2^{10})$

PSI-card: Performance and Comparison

Our framework unifies and explains prior protocols

- DDH-cwPRF-based mqRPMT: recover PSI-card [HFH99] (add Bloom filter optimization)
- DDH-pOPRF-based mqRPMT: recover PSI-card [CGT12]

PSI-card	Running time (s)						Comm. (MB)		
	LAN			WAN			total		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
[GMR⁺21]	1.00	8.41	126.01	8.60	27.46	323.52	2.93	55.49	1030
Our PSI-card$^{\blacklozenge}$	**0.49**	**7.20**	**114.31**	**1.30**	**9.68**	**136.06**	**0.53**	**8.59**	**137.31**
Our PSI-card$^{\blacktriangledown}$	**0.53**	**8.00**	**128.00**	**1.35**	**10.16**	**141.31**	**0.28**	**4.58**	**73.20**
Our PSI-card$^{\bigstar}$	**0.27**	**3.51**	**54.89**	**0.82**	**5.42**	**68.31**	**0.27**	**4.46**	**71.30**

compared to the SOTA

$\#$ time speeds up 2.3-$10.5\times$, $\#$ communication reduces 11.3-$15.2\times$

PSI-card-sum: Performance and Comparison

PSI-card-sum	Running time (s)						Comm. (MB)		
	LAN			WAN			total		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
[IKN^{+}20]$^{\blacktriangledown}$ (deployed)	23.64	176.34	–	30.10	186.29	–	2.72	43.24	–
Our PSI-card-sum$^{\blacklozenge}$	**0.51**	**7.22**	**113.66**	**1.46**	**9.68**	**136.27**	**0.65**	**10.12**	**161.40**
Our PSI-card-sum$^{\blacktriangledown}$	**0.57**	**8.12**	**129.66**	**1.94**	**11.83**	**157.66**	**0.39**	**6.10**	**97.34**
Our PSI-card-sum*	**0.31**	**3.73**	**57.44**	**1.36**	**6.53**	**76.16**	**0.37**	**5.75**	**95.30**

google / **private-join-and-compute** Public

↓

compared to the SOTA

\# time speeds up 22.1-76.3×, # communication reduces 7.4-7.5×

PSU: Performance and Comparison

PSU	Running time (s)						Comm. (MB)		
	LAN			WAN			total		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
[GMR^{+}21]	1.16	10.06	151.34	10.34	38.52	349.43	3.85	67.38	1155
[?]♦	4.87	12.19	141.38	5.78	15.75	182.88	1.35	21.41	342.38
[?]▼	5.10	15.13	187.29	5.82	17.37	210.06	0.77	12.20	195.17
[JSZ^{+}22]	2.29	8.50	516.04	5.33	27.00	736.30	3.59	70.37	1341.55
Our PSU♦	**0.52**	**7.27**	**114.44**	**1.70**	**10.56**	**143.29**	**0.69**	**10.61**	**169.37**
Our PSU▼	**0.57**	**8.04**	**128.20**	**1.76**	**10.92**	**148.15**	**0.42**	**6.61**	**105.23**
Our PSU★	**0.30**	**3.55**	**55.48**	**1.19**	**6.38**	**74.96**	**0.41**	**6.48**	**103.31**

compared to the SOTA: first achieves strict linear complexity

$\#$ time speeds up 2.4-$17\times$, $\#$ communication reduces $2\times$

Private-ID: Performance and Comparison

Private-ID	Running time (ms)						Comm. (MB)		
	LAN			WAN			total		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
[GMR+21]	1.65	11.023	158.76	13.82	43.00	385.12	4.43	76.57	1293
[BKM+20]★	2.21	37.56	671.75	7.98	46.97	710.94	1.00	15.97	226.70
Our Private-ID♦	**0.55**	**7.28**	**115.63**	**5.34**	**14.83**	**163.43**	**3.12**	**16.91**	**237.55**
Our Private-ID▼	**0.65**	**8.43**	**134.16**	**5.69**	**15.68**	**169.05**	**2.85**	**12.91**	**173.50**
Our Private-ID★	**0.34**	**3.78**	**59.76**	**5.04**	**10.87**	**94.89**	**2.82**	**12.74**	**171.54**

- distributed OPRF: SOTA OPRF [RR22] built from VOLE and improved OKVS
- PSU protocol: cwPRF-based mqRPMT

compared to the SOTA
time speeds up $2.7\text{-}4.9\times$, # communication is slightly larger

Outline

1. PSO Framework from mqRPMT
2. Construction of mqRPMT
   - 1st Construction from Commutative Weak PRF
   - 2nd Construction from Permuted Oblivious PRF
   - Connection Between mqPMT and mqRPMT
3. Comparison and Experimentation
4. Summary

Summary of This Work

Unified PSO framework from mqRPMT

- show mqRPMT is complete for all PSO protocols
- greatly reduce the deployment and maintaining costs of PSO

Generic construction of mqRPMT

- cwPRF: demonstrate that DDH assumption is truly a golden goose
- permuted OPRF: make the concept of OPRF more useful; somewhat explain inefficiency of PSU/PCSI
- mqRPMT* from Sigma-mqPMT: an initial step towards the connection to mqPMT

Efficient implementation

- identify expensive ECC operations in cheap disguise
- find the perfect match: `Curve25519`

About Research

> *From [Grothendieck], I have learned not to take glory in the difficulty of a proof.*

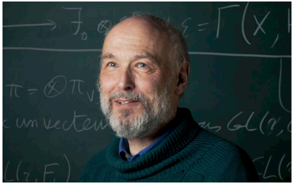


Figure: Pierre Deligne

About Research

> *From [Grothendieck], I have learned not to take glory in the difficulty of a proof.*

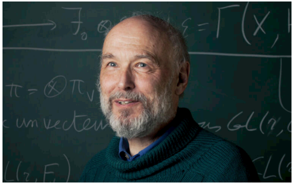


Figure: Pierre Deligne

Likewise, we do not take shame in the simplicity of our construction :-)

Simple is elegant and extremely efficient.

Thanks for Your Attention!

Any Questions?

Reference I

Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin.
Private matching for compute.
2020.
`https://eprint.iacr.org/2020/599`.

Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik.
Fast and private computation of cardinality of set intersection and union.
In *Cryptology and Network Security, 11th International Conference, CANS 2012*, volume 7712, pages 218–231. Springer, 2012.

Hao Chen, Kim Laine, and Peter Rindal.
Fast private set intersection from homomorphic encryption.
In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 1243–1255. ACM, 2017.

Reference II

Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold.
Keyword search and oblivious pseudorandom functions.
In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

Siyao Guo, Pritish Kamath, Alon Rosen, and Katerina Sotiraki.
Limits on the efficiency of (ring) lwe-based non-interactive key exchange.
J. Cryptol., 35(1):1, 2022.

Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh.
Private set operations from oblivious switching.
In *Public-Key Cryptography - PKC 2021*, volume 12711 of *Lecture Notes in Computer Science*, pages 591–617. Springer, 2021.

Reference III

- Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg.
  Enhancing privacy and trust in electronic communities.
  In *Proceedings of the First ACM Conference on Electronic Commerce (EC-99)*, pages 78–86. ACM, 1999.
- Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung.
  On deploying secure computing: Private intersection-sum-with-cardinality.
  In *IEEE European Symposium on Security and Privacy, EuroS&P 2020*, pages 370–389. IEEE, 2020.
- Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu.
  Shuffle-based private set union: Faster and more secure.
  In *USENIX 2022*, 2022.

Reference IV

Catherine A. Meadows.
A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party.
In *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, pages 134–137. IEEE Computer Society, 1986.

Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai.
Spot-light: Lightweight private set intersection from sparse OT extension.
In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference*, volume 11694 of *Lecture Notes in Computer Science*, pages 401–431. Springer, 2019.

Srinivasan Raghuraman and Peter Rindal.
Blazing fast PSI from improved OKVS and subfield VOLE.
In *ACM CCS 2022*, 2022.

Reference V

Mike Rosulek and Ni Trieu.
Compact and malicious private set intersection for small sets.
In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1166–1181. ACM, 2021.