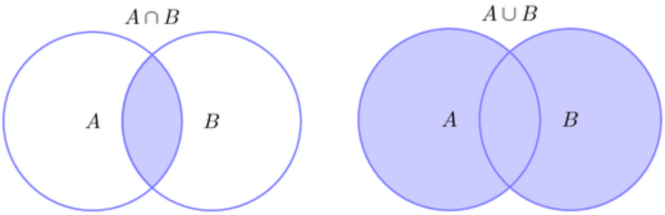
A Framework of Private Set Operations from Mulit-query Reverse Private Membership Test



Yu Chen
Shandong University
joint work with Min Zhang, Cong Zhang, Minglang Dong and Weiran Liu

Outline

Outline

Privacy Preserving Computation

国家重大战略

国务院《关于构建要素市场化的意见》
《十四五规划和 2035 年远景目标纲要》

数据是新型生产要素 $\leadsto$ 激活数据要素潜能

数据保护需求

数据泄露事件频发，损失难以估量
三法五典出台

严格保护数据安全 $\leadsto$ 数据流动性降低

Gartner 2021：变革型前沿技术 $\Rightarrow$ 破局的关键、数字经济的安全底座

$\left.\begin{array}{l}\text{高级密码方案}\\ \text{零知识证明}\\ \text{安全多方计算}\end{array}\right\}$

打破数据孤岛 $\longrightarrow$ 释放数据价值

Private Set Operations (high frequency and high value)

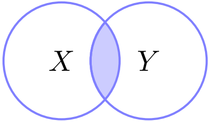
sender

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

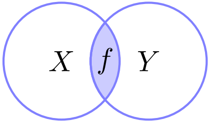
$V = \{v_1, \ldots, v_{n_2}\}$

receiver

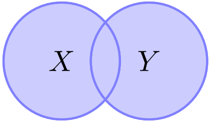
$Y = \{y_1, \ldots, y_{n_1}\}$



$\text{PSI} = X \cap Y$



$$\text{PCSI} = \begin{cases} |X \cap Y| & \text{cardinality} \\ |X \cap Y|, \sum_{x_i \in X \cap Y} v_i & \text{cardinality-sum} \\ f(X \cap Y) & \text{general computation} \end{cases}$$



$\text{PSU} = X \cup Y$

Wide Applications of PSO

PSI

- privacy-preserving location sharing
- private contact discovery
- DNA testing and pattern matching

PCSI

- measuring the effectiveness of online advertising

PSU

- IP blacklist and vulnerability data aggregation
- private DB supporting full join
- private-ID

SOTA of PSO

PSI has been extensively studied in the last two decades

- balanced setting: [KKRT16, CM20, RR22] achieves linear complexity, and almost as efficient as insecure hash protocol
- unbalanced setting: [CLR17, CHLR18, CMdG^{+}21] achieves sub-linear complexity of large set

In sharp contrast, the study of PCSI and PSU are not satisfying.

SOTA of PSO

PSI has been extensively studied in the last two decades

- balanced setting: [KKRT16, CM20, RR22] achieves linear complexity, and almost as efficient as insecure hash protocol
- unbalanced setting: [CLR17, CHLR18, CMdG+21] achieves sub-linear complexity of large set

In sharp contrast, the study of PCSI and PSU are not satisfying.

PCSI

- [HFH99, IKN+20, PSTY19] achieve linear complexity

concretely $20\times$ slower in timing and $30\times$ more communication than PSI

SOTA of PSO

PSI has been extensively studied in the last two decades

- balanced setting: [KKRT16, CM20, RR22] achieves linear complexity, and almost as efficient as insecure hash protocol
- unbalanced setting: [CLR17, CHLR18, CMdG$^+$21] achieves sub-linear complexity of large set

In sharp contrast, the study of PCSI and PSU are not satisfying.

PCSI

- [HFH99, IKN$^+$20, PSTY19] achieve linear complexity

concretely $20\times$ slower in timing and $30\times$ more communication than PSI

PSU

- [KS05, Fri07, HN10, KRTW19, JSZ$^+$22] have superlinear complexity
- [DC17, ZCL$^+$23] achieve linear complexity, but not strict (communication or computation complexity additionally depends on statistical parameter $\lambda \approx 40$)

concretely $20\times$ slower in timing and $25\times$ more communication than PSI

Motivation

Different approaches are used for different private set operations $\rightsquigarrow$ require much more engineering effort and maintaining cost

- Goal: a unified framework of PSO

[1] [GMR+21] presented a PSO framework from permuted characteristic. However, its oblivious shuffle functionality is not necessary for PSO, and incurs superlinear complexity.

Motivation

Different approaches are used for different private set operations $\rightsquigarrow$ require much more engineering effort and maintaining cost

- Goal: a unified framework of PSO

There exists huge efficiency gap between PSI and other PSO protocols

- Goal: efficient instantiations to close the gap[1]

[1] [GMR$^+$21] presented a PSO framework from permuted characteristic. However, its oblivious shuffle functionality is not necessary for PSO, and incurs superlinear complexity.

Motivation

Different approaches are used for different private set operations $\rightsquigarrow$ require much more engineering effort and maintaining cost

- Goal: a unified framework of PSO

There exists huge efficiency gap between PSI and other PSO protocols

- Goal: efficient instantiations to close the gap[1]

After $\approx$ 40 years, DH-PSI [Mea86] is still the most easily understood and implemented one among numerous PSI protocols. Surprisingly, no counterpart is known in the PSU setting yet. Existing protocols are very complicated.

- Goal: build DDH-based PSU protocol as simple as DH-PSI

[1] [GMR+21] presented a PSO framework from permuted characteristic. However, its oblivious shuffle functionality is not necessary for PSO, and incurs superlinear complexity.

Motivation

Different approaches are used for different private set operations $\rightsquigarrow$ require much more engineering effort and maintaining cost

- Goal: a unified framework of PSO

There exists huge efficiency gap between PSI and other PSO protocols

- Goal: efficient instantiations to close the gap[1]

After $\approx$ 40 years, DH-PSI [Mea86] is still the most easily understood and implemented one among numerous PSI protocols. Surprisingly, no counterpart is known in the PSU setting yet. Existing protocols are very complicated.

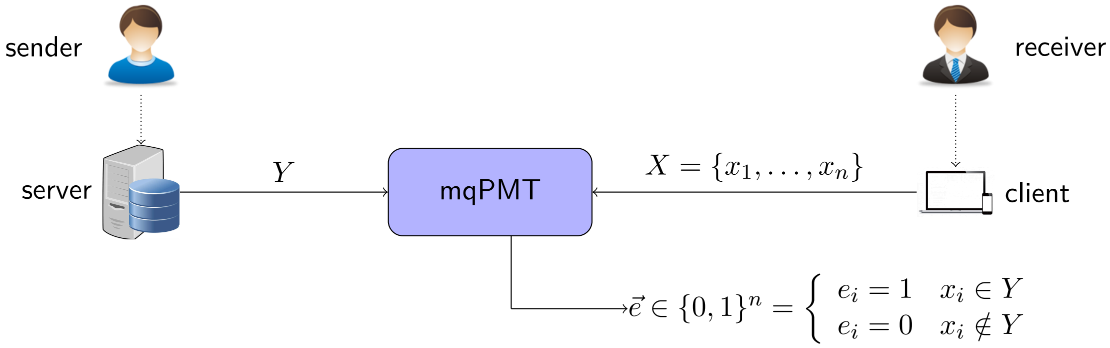
- Goal: build DDH-based PSU protocol as simple as DH-PSI

Is there a central building block that enables a unified framework for PSO?
How to give instantiations with optimal asymptotic complexity and good concrete efficiency?
Can the DDH assumption strike back with efficient PSU protocol?

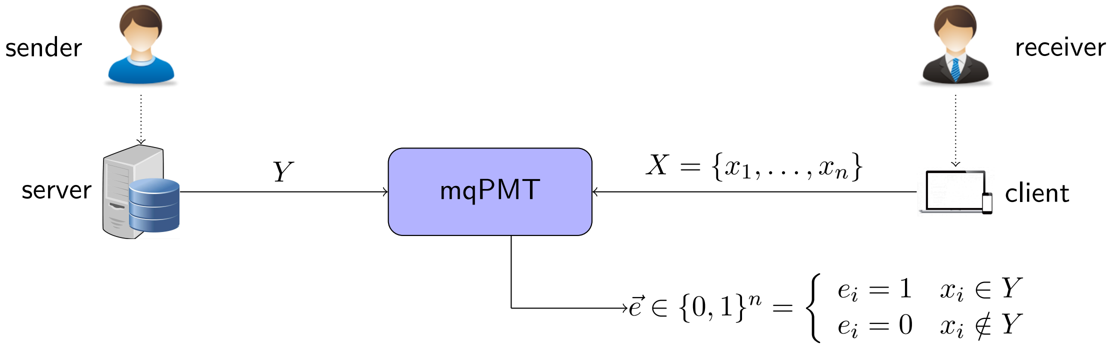
[1] [GMR+21] presented a PSO framework from permuted characteristic. However, its oblivious shuffle functionality is not necessary for PSO, and incurs superlinear complexity.

Outline

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

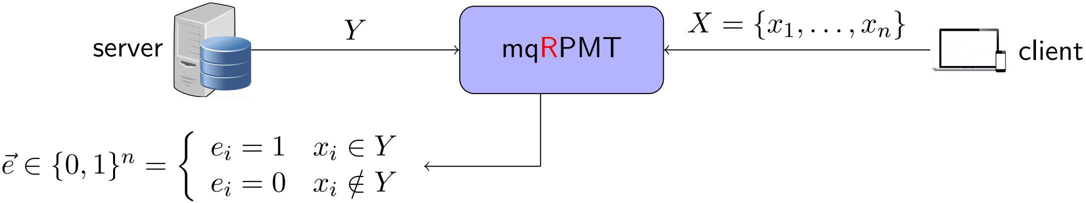


Start Point: multi-query Private Membership Test (mqPMT) underlying PSI

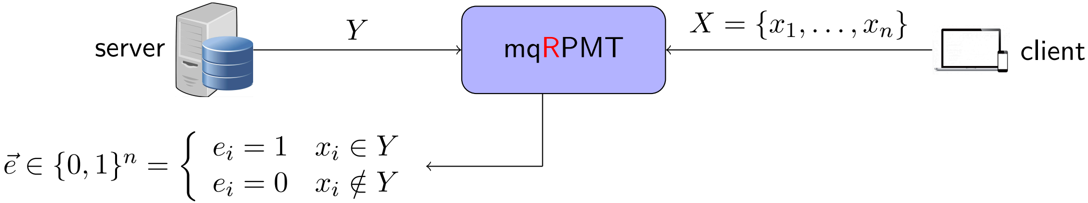


- Problem: the client learns both x_i and e_i, a.k.a. the intersection $\rightsquigarrow$ not suitable for protocols that should hide intersection, such as PCSI and PSU.

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

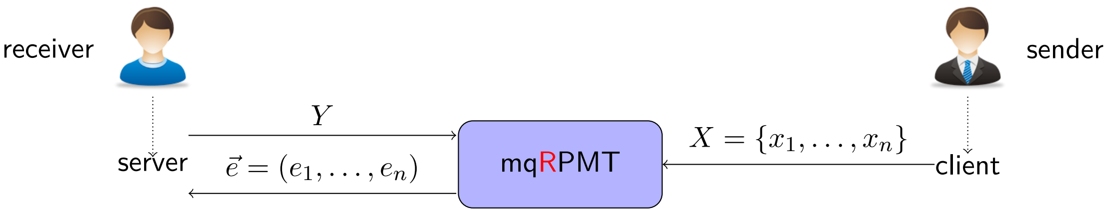


The core protocol: multi-query Reverse Private Membership Test (mqRPMT)

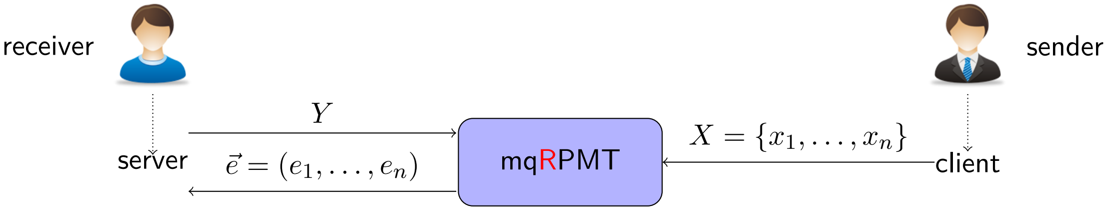


- The server learns e_i, while the client learns x_i, a.k.a. the information of intersection is shared between the two parties $\rightsquigarrow$ suitable for all PSO protocols

PSO from mqRPMT

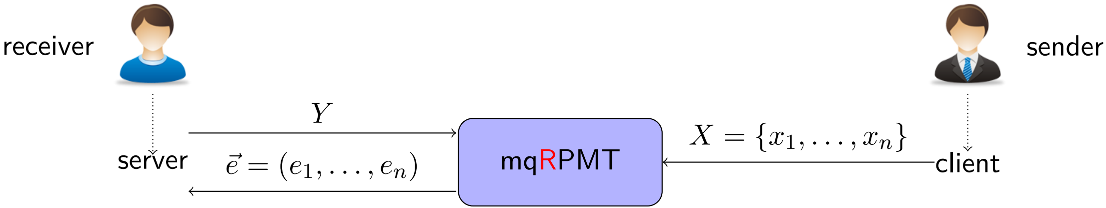


PSO from mqRPMT

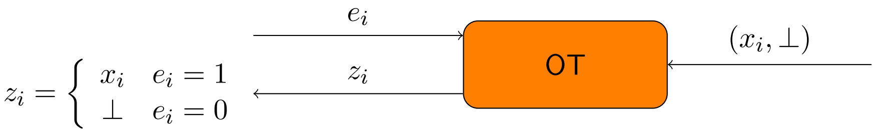


directly yields PSI-card: $|X \cap Y|$ is the Hamming weight of $\vec{e}$

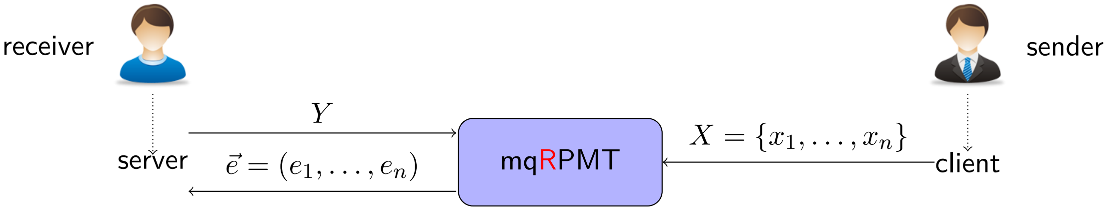
PSO from mqRPMT



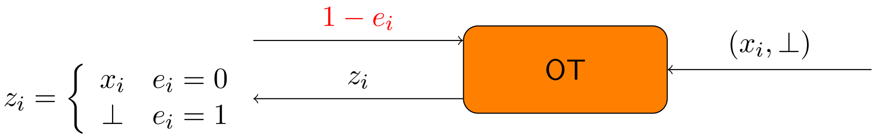
yields PSI coupled with OT: receiver obtains $X \cap Y$



PSO from mqRPMT



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



PSO from mqRPMT

receiver 

sender 

server $\xrightarrow{Y}$ **mqRPMT** $\xleftarrow{X = \{x_1, \ldots, x_n\}}$ client

server $\xleftarrow{\vec{e} = (e_1, \ldots, e_n)}$ **mqRPMT**

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

$\xrightarrow{e_i}$ **OT** $\xleftarrow{(r_i, v_i + r_i)}$ $V = \{v_1, \ldots, v_n\}$

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

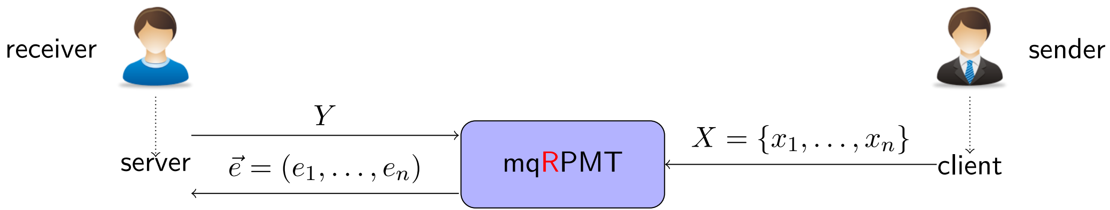
$$z_i = \begin{cases} r_i & e_i = 1 \\ v_i + r_i & e_i = 0 \end{cases} \quad \xleftarrow{z_i} \text{OT}$$

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

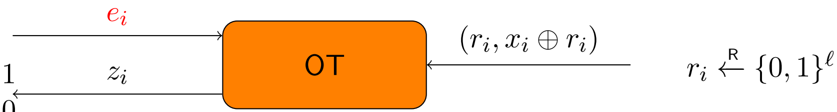
receiver obtains $|X \cap Y|$

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

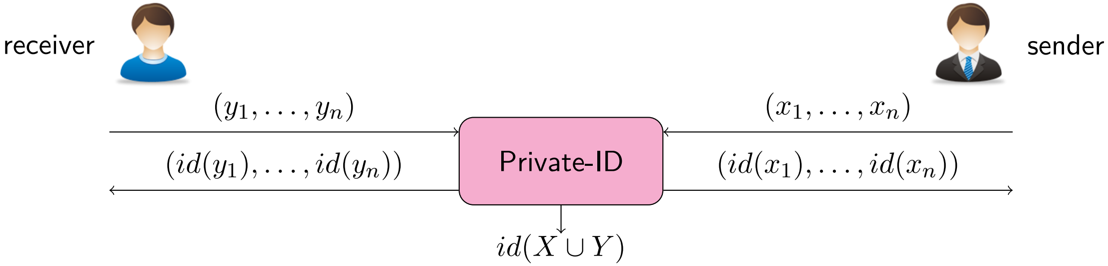


$$z_i = \begin{cases} r_i & e_i = 1 \\ x_i \oplus r_i & e_i = 0 \end{cases}$$

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

sender has $x_i \oplus r_i$

Private-ID



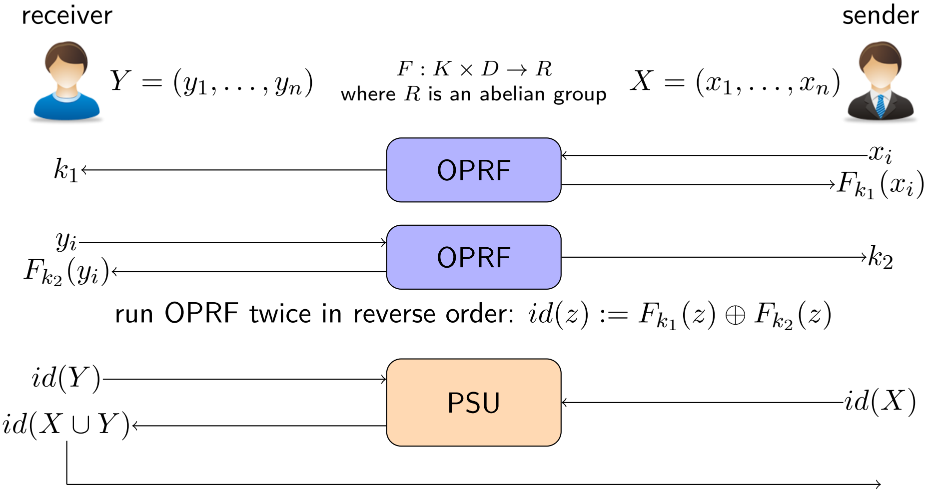
Buddhavarapu et al. [BKM+20] proposed private-ID:

- 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.



Our Construction of Private-ID

receiver

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

sender

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

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

$\{y_i\}_{i=1}^n \to$ distributed OPRF $\leftarrow \{x_i\}_{i=1}^n$

$k_1, \{G_{k_1,k_2}(y_i)\}_{i=1}^n \leftarrow$ distributed OPRF $\to k_2, \{G_{k_1,k_2}(x_i)\}_{i=1}^n$

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

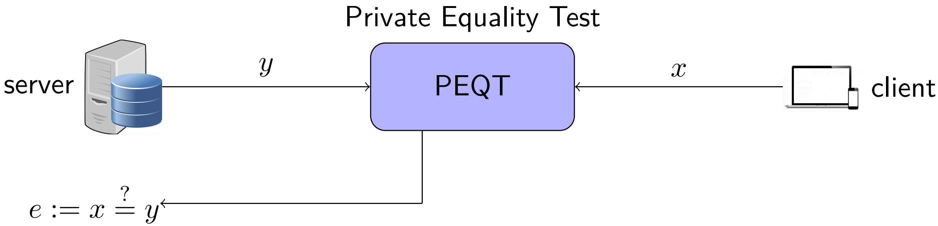
$id(Y) \to$ distributional PSU $\leftarrow id(X)$

$id(X \cup Y) \leftarrow$ distributional PSU

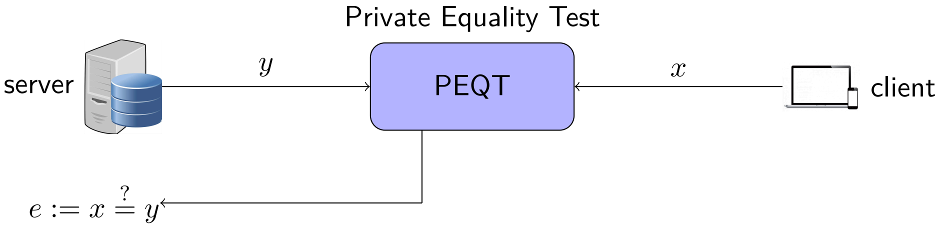
Outline

Outline

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 $R = D$) $\rightsquigarrow F_{k_1}(F_{k_2}(\cdot))$ 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 ($k \xleftarrow{R} K$, $x \xleftarrow{R} X$) 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 ($k \xleftarrow{R} K$, $x \xleftarrow{R} X$) 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 $k' \xleftarrow{R} K$, $x \xleftarrow{R} D$, 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 $k \xleftarrow{R} \mathbb{Z}_p$.
- 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 $\Rightarrow$ NIKE.

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

non-interactive key-exchange

$(a, F_a(x))$ $\qquad\qquad$ $(b, F_b(x))$

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

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

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

mqRPMT from cwPRF

PEQT

P_1 (server) — y — $k_1 \xleftarrow{R} K$

P_2 (client) — x — $k_2 \xleftarrow{R} K$

partial encoding

$$P_1 \xrightarrow{F_{k_1}(\mathsf{H}(y))} P_2$$

$$P_1 \xleftarrow{F_{k_2}(\mathsf{H}(x)),\ F_{k_2}(F_{k_1}(\mathsf{H}(y)))} P_2$$

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

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

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

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

P_2 (client)

x

$k_2 \xleftarrow{R} K$

set $e = 1$ iff

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

mqRPMT

P_1 (server)

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

$k_1 \xleftarrow{R} K$

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

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

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

P_2 (client)

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

$k_2 \xleftarrow{R} K$

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

$k_1 \xleftarrow{R} K$

partial encoding

P_2 (client)

x

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

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

$k_2 \xleftarrow{R} K$

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

set $e = 1$ iff
$F_{k_1}(F_{k_2}(\mathsf{H}(x))) \stackrel{?}{=} F_{k_2}(F_{k_1}(\mathsf{H}(y)))$

mqRPMT

P_1 (server)

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

$k_1 \xleftarrow{R} K$

P_2 (client)

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

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

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

$k_2 \xleftarrow{R} K$

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

set $e_i = 1$ iff
$F_{k_1}(F_{k_2}(\mathsf{H}(x_i))) \in \Omega$

send encoding in order reveals intersection!

mqRPMT from cwPRF

PEQT

P_1 (server) | P_2 (client)

y | x

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

partial encoding

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

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

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

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

set $e = 1$ iff

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

mqRPMT

P_1 (server) | P_2 (client)

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

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

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

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

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

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

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

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)

y

$k_1 \xleftarrow{R} K$

partial encoding

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

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

P_2 (client)

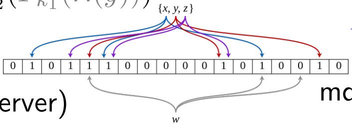
x

$k_2 \xleftarrow{R} K$

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

set $e = 1$ iff

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



mqRPMT

P_1 (server)

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

$k_1 \xleftarrow{R} K$

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

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

P_2 (client)

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

$k_2 \xleftarrow{R} K$

set $e_i = 1$ iff

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

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

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	$3n \times \|D\|$ or $2n \times \|D\| + n \cdot 1.44\lambda\ (\ll \|D\|)$

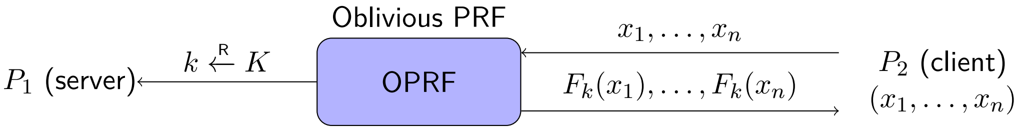
cwPRF-based mqRPMT is optimal in the sense that both computation and communication complexities are strictly linear in n

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

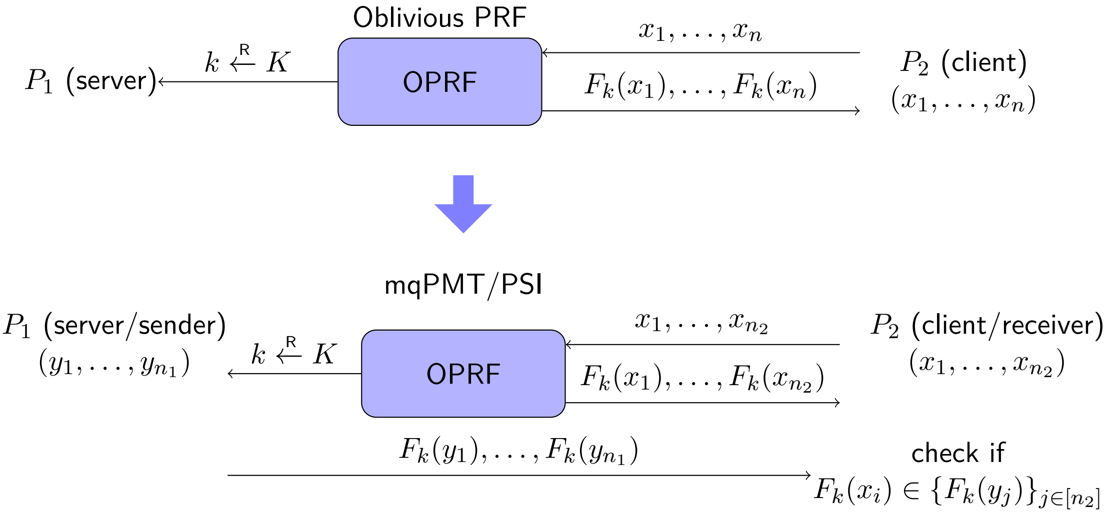
incredibly simple and efficient

Outline

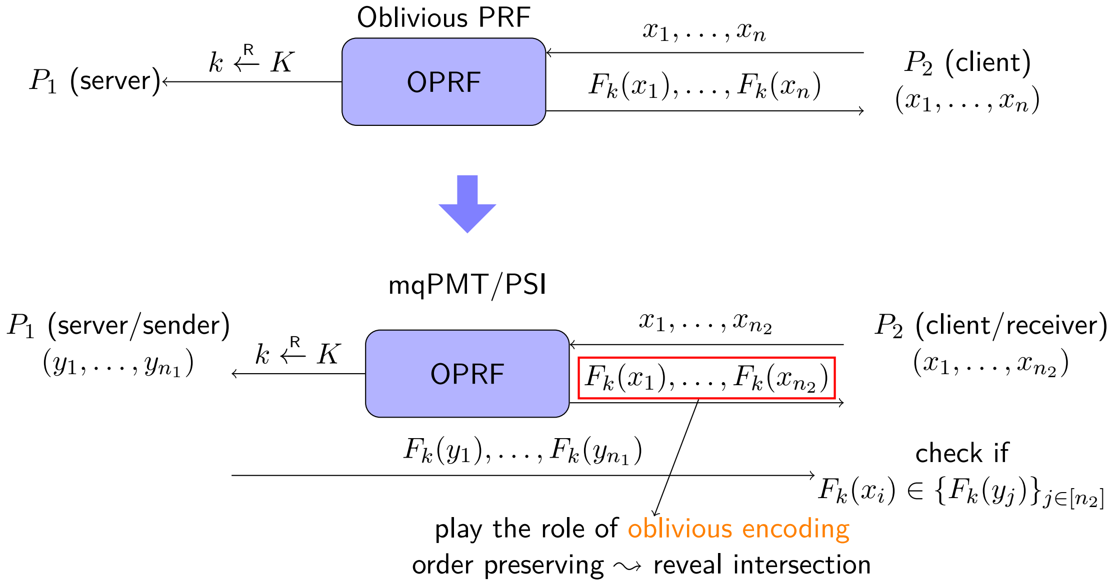
Starting Point: mqPMT/PSI from OPRF



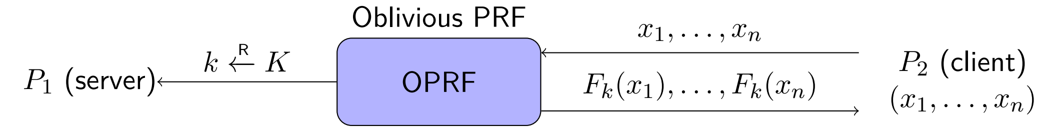
Starting Point: mqPMT/PSI from OPRF

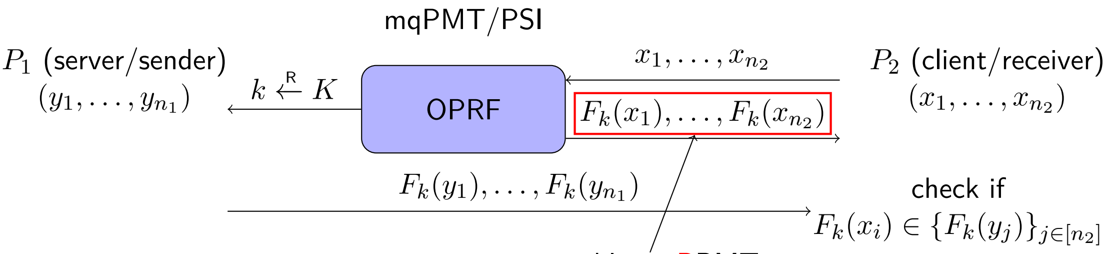


Starting Point: mqPMT/PSI from OPRF



Starting Point: mqPMT/PSI from OPRF

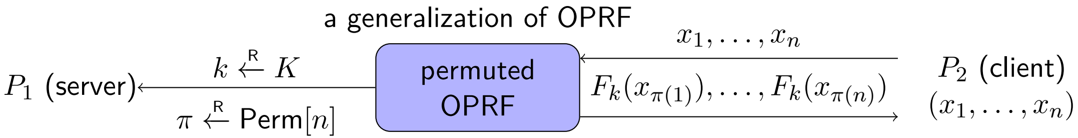




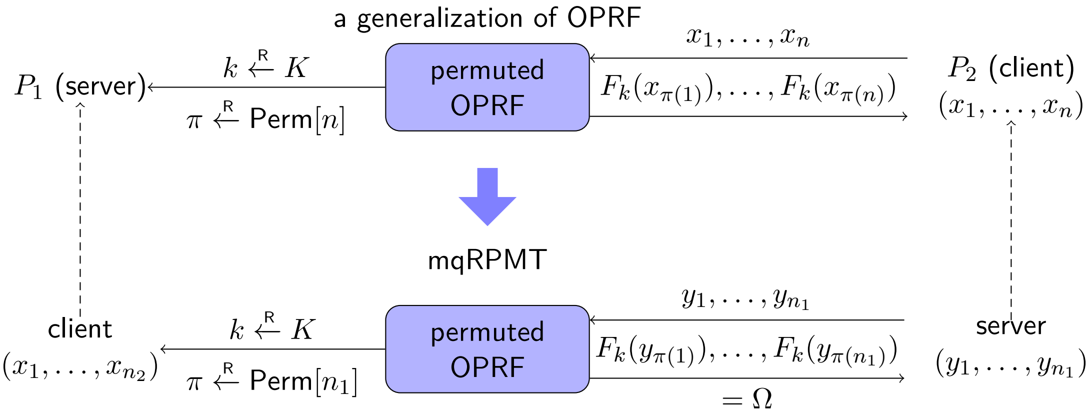
to enable mqRPMT

we need permuted oblivious encoding

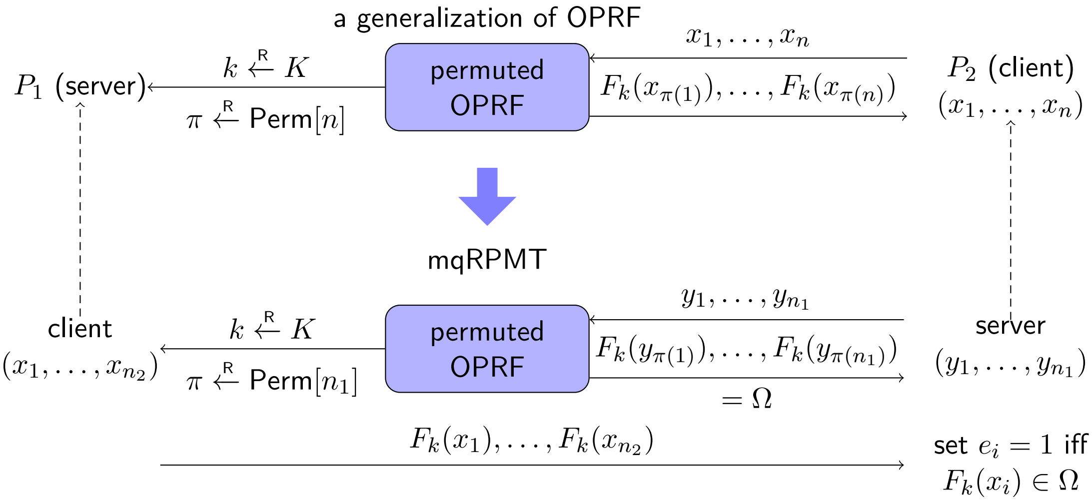
mqRPMT from Permuted OPRF



mqRPMT from Permuted OPRF



mqRPMT from Permuted OPRF



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		client
k		$(x_1, \ldots, x_n)$
	$\xleftarrow{\widehat{x_1}, \ldots, \widehat{x_n}}$	mask input
evaluate	$\xrightarrow{\widehat{z_1} = G'_k(\widehat{x_1}), \ldots, \widehat{z_n} = G'_k(\widehat{x_n})}$	unmask output: $\widehat{z_i} \to z_i$

Build Permuted OPRF from cwPRP

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

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

server		client
k		$(x_1, \ldots, x_n)$
	$\xleftarrow{\widehat{x_1}, \ldots, \widehat{x_n}}$	mask input
evaluate	$\xrightarrow{\widehat{z_1} = G'_k(\widehat{x_1}), \ldots, \widehat{z_n} = G'_k(\widehat{x_n})}$	unmask output: $\widehat{z_i} \rightarrow z_i$

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		client
k		$(x_1, \ldots, x_n)$
	$\xleftarrow{\widehat{x_1}, \ldots, \widehat{x_n}}$	mask input
evaluate	$\xrightarrow{\widehat{z_1} = G'_k(\widehat{x_1}), \ldots, \widehat{z_n} = G'_k(\widehat{x_n})}$	unmask output: $\widehat{z_i} \to z_i$

shuffle $\rightsquigarrow$ 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 acturally 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	`Curve25519`	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

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.

$$
\begin{array}{ccc}
P_1 \text{ (server)} & & P_2 \text{ (client)} \\
Y = (y_1, \ldots, y_{n_1}) & & X = (x_1, \ldots, x_{n_2}) \\
a \leftarrow \mathsf{Encode}(Y) & \xrightarrow{\quad a \quad} & \\
 & \xleftarrow{\vec{q} = \{q_1, \ldots, q_{n_2}\}} & q_i \leftarrow \mathsf{GenQuery}(a, x_i) \\
z_i \leftarrow \mathsf{Response}(q_i) & \xrightarrow{\vec{z} = \{z_1, \ldots, z_{n_2}\}} & e_i \leftarrow \mathsf{Test}(a, z_i)
\end{array}
$$

- **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).

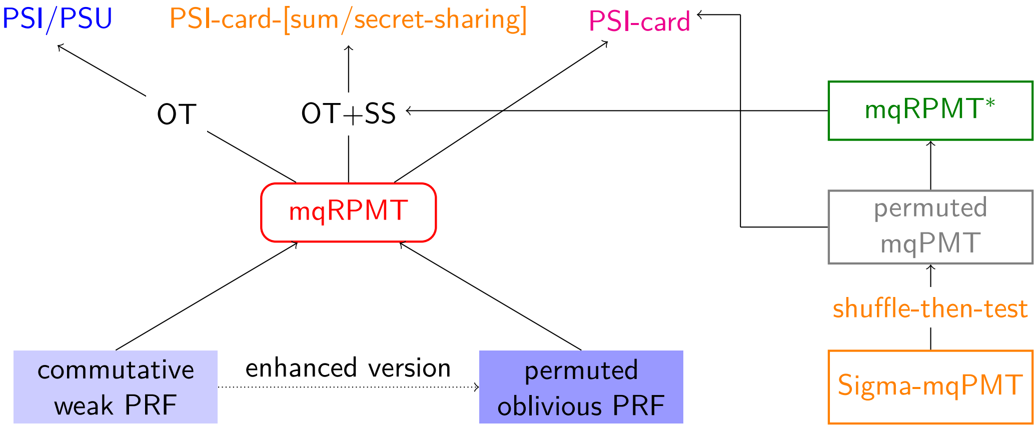
mqRPMT* from Sigma-mqPMT

P_1 (server)		P_2 (client)
$Y = (y_1, \ldots, y_{n_1})$		$X = (x_1, \ldots, x_{n_2})$
$a \leftarrow \mathsf{mqPMT.Encode}(Y)$	$\xrightarrow{\quad a \quad}$	
	$\xleftarrow{\vec{q} = \{q_1, \ldots, q_{n_2}\}}$	$q_i \leftarrow \mathsf{mqPMT.GenQuery}(a, x_i)$
$\pi \xleftarrow{\mathsf{R}} \mathsf{Perm}[n_2]$	$\xrightarrow{\vec{z}^* = \{z_{\pi(1)}, \ldots, z_{\pi(n_2)}\}}$	
$\vec{e} = \{e^*_{\pi^{-1}(i)}\}_{i=1}^{n_2}$	$\xleftarrow{\vec{e}^* = \{e^*_1, \ldots, e^*_{n_2}\}}$	$e^*_i \leftarrow \mathsf{mqPMT.Test}(a, z^*_i)$

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



Outline

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

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

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 fast Exp in compressed form $\rightsquigarrow$ halve communication & no decompression
   - any 32-byte bit string (interpreting as X-coordinate) can be ambiguously identified as a valid 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 fast Exp in compressed form $\rightsquigarrow$ halve communication & no decompression
   - any 32-byte bit string (interpreting as X-coordinate) can be ambiguously identified as a valid 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 $\rightsquigarrow$ support large-scale applications

```
mpc
  > oprf
  > ot
  > peqt
  > psi
  v pso
      mqrpmt_private_id.hpp
      mqrpmt_psi_card_sum.hpp
      mqrpmt_psi_card.hpp
      mqrpmt_psi.hpp
      mqrpmt_psu.hpp
  v 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 $\times$	1.9-3.0 $\times$	1.8-2.8 $\times$	1.2-1.3 $\times$	1.5-1.9 $\times$	1.6-2.2 $\times$	–	–	–
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 $\times$	1.6-2.3 $\times$	1.6-2.2 $\times$	1.1-1.3$\times$	1.4-1.8 $\times$	1.5-2 $\times$	–	–	–
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 $\times$	2.0-3.3 $\times$	1.9-3.6 $\times$	1.1-1.1 $\times$	1.4-1.8 $\times$	1.7-2.4 $\times$	–	–	–

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 $2.3\text{-}10.5\times$, # communication reduces $11.3\text{-}15.2\times$

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$^{\bigstar}$	**0.31**	**3.73**	**57.44**	**1.36**	**6.53**	**76.16**	**0.37**	**5.75**	**95.30**

Google

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
[ZCL+23]♦	4.87	12.19	141.38	5.78	15.75	182.88	1.35	21.41	342.38
[ZCL+23]▼	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 $2.4\text{-}17\times$, # 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

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: a 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.

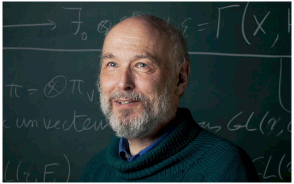


Figure: Pierre Deligne

About Research

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

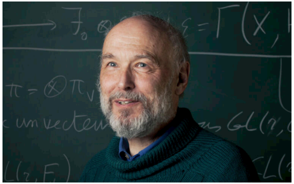


Figure: Pierre Deligne

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

Thanks for Your Attention!

Any Questions?

http://eprint.iacr.org/2022/652

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, Zhicong Huang, Kim Laine, and Peter Rindal.
Labeled PSI from fully homomorphic encryption with malicious security.
In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, pages 1223–1237. ACM, 2018.

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

Melissa Chase and Peihan Miao.
Private set intersection in the internet setting from lightweight oblivious PRF.
In *Advances in Cryptology - CRYPTO 2020*, volume 12172 of *Lecture Notes in Computer Science*, pages 34–63. Springer, 2020.

Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg.
Labeled PSI from homomorphic encryption with reduced computation and communication.
In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1135–1150. ACM, 2021.

Alex Davidson and Carlos Cid.
An efficient toolkit for computing private set operations.
In *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017*, volume 10343 of *Lecture Notes in Computer Science*, pages 261–278. Springer, 2017.

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.

Reference III

Keith B. Frikken.
Privacy-preserving set union.
In *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007*, volume 4521 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2007.

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.

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.

Carmit Hazay and Kobbi Nissim.
Efficient set operations in the presence of malicious adversaries.
In *Public Key Cryptography - PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 312–331. Springer, 2010.

Reference IV

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.

Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu.
Efficient batched oblivious PRF with applications to private set intersection.
In *CCS 2016*, pages 818–829. ACM, 2016.

Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang.
Scalable private set union from symmetric-key techniques.
In *Advances in Cryptology - ASIACRYPT 2019*, volume 11922 of *Lecture Notes in Computer Science*, pages 636–666. Springer, 2019.

Lea Kissner and Dawn Xiaodong Song.
Privacy-preserving set operations.
In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2005.

Reference V

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.

Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai.
Efficient circuit-based PSI with linear communication.
In *Advances in Cryptology - EUROCRYPT 2019*, volume 11478 of *Lecture Notes in Computer Science*, pages 122–153. Springer, 2019.

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

Reference VI

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.

Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin.
Optimal private set union from multi-query reverse private membership test.
In *USENIX 2023*, 2023.
https://eprint.iacr.org/2022/358.