Concretely Efficient Correlated Oblivious Permutation

Feng Han
Alibaba Group
fengdi.hf@alibaba-inc.com

Xiao Lan
Chinese Academy of Sciences
lanxiaoscu@gmail.com

Weiran Liu
Alibaba Group
weiran.lwr@alibaba-inc.com

Lei Zhang
Alibaba Group
zongchao.zl@taobao.com

Hao Ren
Nanyang Technological University
hao.ren@ieee.org

Lin Qu
Alibaba Group
xide.ql@taobao.com

Yuan Hong
University of Connecticut
yuan.hong@uconn.edu

Abstract

Oblivious permutation (OP) enables two parties, a sender with a private data vector $\mathbf{x}$ and a receiver with a private permutation π, to securely obtain the shares of $\pi(\mathbf{x})$. OP has been used to construct many important MPC primitives and applications such as secret shuffle, oblivious sorting, private set operations, secure database analysis, and privacy-preserving machine learning. Due to its high complexity, OP has become a performance bottleneck in several practical applications, and many efforts have been devoted to enhancing its concrete efficiency. Chase et al. (Asiacrypt'20) proposed an offline-online OP paradigm leveraging a pre-computable resource termed Share Translation. While this paradigm significantly reduces online costs, the substantial offline cost of generating Share Translation remains an area for further investigation.

In this work, we redefine the pre-computable resource as a cryptographic primitive known as Correlated Oblivious Permutation (COP) and conduct in-depth analyses and optimizations of the two COP generation solutions: network-based solution and matrix-based solution. The optimizations for the network-based solution halve the communication/computation cost of constructing a switch (the basic unit of the permutation network) and reduce the number of switches in the permutation network. The optimizations for the matrix-based solution halve the communication cost of small-size COP generation and reduce the cost of large-size COP generation with in-outside permutation decomposition.

We implement our two COP generation protocols and conduct comprehensive evaluations. Taking commonly used 128-bit input data as an example, our network-based and matrix-based solutions are up to 1.7× and 1.6× faster than baseline protocols, respectively. We further facilitate the state-of-the-art (SOTA) PSU protocols with our optimized COP, achieving over 25% reduction in communication cost and 35% decrease in execution time. This shows that our COP optimizations bring significant improvements for real-world MPC primitives.

Keywords

Oblivious permutation, secret shuffle

1 Introduction

Oblivious permutation (OP), also referred to as "permute + share", enables two parties, a sender with an input vector $\mathbf{x}$ and a receiver with a permutation π, to jointly compute the shares of $\pi(\mathbf{x})$ without revealing anything else to either party. OP is a fundamental cryptographic primitive in secure multi-party computation (MPC) and has numerous applications [1, 3, 19, 20, 24, 28, 30, 42]. Specifically, OP serves as the main building block in oblivious sorting [1], federated database analysis [33, 37, 42], privacy-preserving machine learning [30, 31], and differential privacy with shuffle model on MPC [3].

An important application of OP is to build two-party [13] and multi-party [18, 19] secret shuffle. Consider a scenario in which multiple parties jointly hold a shared data array generated through certain MPC functionalities. Directly revealing this array would expose its relationship to the original inputs. Secret shuffle can permute this array using a random permutation unknown to everyone. The typical method to implement secret shuffle is for each party to sample a random permutation and then sequentially invoke OPs to permute the shared data array under these permutations [18, 19]. Since no party can know all permutations, the relationship between the secret data array and the original inputs is obscured, thereby meeting the security requirements. The efficiency of OP directly influences the overall performance of secret shuffle as OP is the sole involved building block.

Another application of OP is to construct private set union (PSU) protocols [20, 24, 28], which enables two parties, each holding a private set of elements, to compute the union of their sets without revealing anything else. Garimella et al. [20] and Jia et al. [24] proposed PSU protocols based on OP. In their intermediate step, the receiver obtains a bit vector indicating whether the elements of the sender's set are in the receiver's set. However, since the receiver knows which element corresponds to which bit in the bit vector, directly revealing the bit vector immediately discloses the intersection to the receiver. OP is used to disrupt such correlation information to prevent this leakage. However, OP is the performance bottleneck in these PSU protocols. As shown in Table 1 and Table 2 of [20], when the set sizes reach 2^{20}, OP accounts for nearly 30% of the PSU execution time in the LAN setting. As OP incurs substantial communication costs, its impact is even more pronounced in the WAN setting, taking nearly 60% of the PSU execution time. According to our experiment, OP takes over the 80% communication cost in the PSU protocol proposed by Jia et al. [24].

Over the last decade, there have been several solutions proposed to realize OP, including network-based [20, 34], AHE-based [13, 26], matrix-based [13], and wPRF-based [36] solutions. Nevertheless, OP is still expensive for real-world applications.

One way to address the efficiency challenges in MPC is to employ an offline-online paradigm. In this paradigm, two parties can generate data-independent correlated randomness in the offline phase. Subsequently, consuming this correlated randomness can dramatically reduce the online communication and computation costs of most existing protocols. A prime example of this approach is using Beaver triples for multiplication [6, 16]. Chase et al. [13] and Peceny et al. [36] extended this paradigm to (matrix-based and wPRF-based) OP. Specifically, the resource for OP is the vectors with the correlation that $\mathbf{c} = \pi(\mathbf{a}) - \mathbf{b}$, where the sender $\mathcal{S}$ holds $(\mathbf{a}, \mathbf{b})$ and the receiver $\mathcal{R}$ holds $(\mathbf{c}, \pi)$. We denote the vectors as Correlated OP (COP) and defer the details to §2.4. We note that network-based solutions [20, 34] can also be adapted to this paradigm without increasing the communication/computing cost (detailed in §3.2).

Although the online permutation process using COP is highly efficient, one still needs to generate COP in the offline phase. Improving COP generation remains essential. Moreover, optimizing COP generation ultimately improves the efficiency of OP even without an offline-online paradigm. It raises the following question.

Is there room for improvement in COP generation?

1.1 Related work

Before answering this question, we first briefly survey the existing OP solutions.

Network-based solutions. The network-based OP solution is first introduced in [34], where the parties securely evaluate a permutation network with any generic MPC protocol, e.g., garbled circuit [45], to obtain the permuted shared vectors. A permutation network contains a set of interconnected switches, where each switch takes two secret-shared values as input and outputs them either in their original order or in swapped order according to one programming bit. The SOTA protocol [20] requires $2l$ bits communication and one correlated oblivious transfer (COT) to implement one switch with l bit inputs. As will be analyzed in §5, the network-based solution is especially efficient on input with a smaller bit length while performing poorly on input with a larger bit length. To the best of our knowledge, previous works utilized this solution to generate binary shares (defined over $\mathbb{F}_2^l$). We point out that it can also be employed to generate shares defined over any ring, such as $\mathbb{F}_{2^l}$ and $\mathbb{F}_p$, by detailing our optimizations in §3 using ring operations.

Matrix-based solutions. The core of the scheme of Chase et al. [13] for small-size COP generation is to let the sender construct a pseudorandom $n \times n$ matrix such that the receiver learns all values except the $\pi(i)$-th value in the i-th row for $1 \le i \le n$, which is fulfilled with puncturable pseudorandom function (PPRF). Therefore, we refer to the solution as the matrix-based COP. They also show how to decompose a large-size permutation into multiple small-size permutations and subsequently compile multiple small-size COPs into a large-size COP. On data with a smaller or larger bit length, the matrix-based solution behaves opposite to that of the network-based solution. As discussed in [13], the matrix-based solution can also generate shares defined over any ring.

AHE and wPRF-based solutions. The solutions based on rerandomizable additively homomorphic encryption (AHE) [13, 26] and weak pseudorandom function (wPRF) [36] achieve linear complexity in the input size and the bit length of each element. We refer to Appendix A for details. AHE-based solutions typically produce secret shares of $\pi(\mathbf{a})$ defined in the corresponding AHE ciphertext space (e.g., $\mathbb{F}_p$), while the wPRF-based solutions produce secret shares of $\pi(\mathbf{a})$ defined in $\mathbb{F}_2^l$. In practice, the most commonly used share types in secure database analysis and machine learning are arithmetic shares on $\mathbb{F}_{2^l}$. Conversions between different share types typically require additional costs [16].

Among these solutions, network-based and matrix-based solutions are flexible to support any sharing type defined over a ring and have become the most commonly used methods in practice. Network-based solutions are extensively used in applications with relatively small-bit inputs, such as PSU [20, 24] and private database analysis [37, 42]. In contrast, matrix-based solutions are more commonly utilized in scenarios with large-bit inputs, such as privacy-preserving machine learning [30]. Also, as we will detail in §3 and §4, there is significant potential for further optimizing both approaches. However, AHE-based and wPRF-based protocols are almost optimized and require post-processing to support diverse input domains. Therefore, we focus on optimizations for network-based and matrix-based solutions. We also provide performance comparisons with the SOTA wPRF-based protocol for completeness.

1.2 Our Contributions

In this paper, we address the aforementioned question in the semi-honest setting[1]. We optimize both network-based and matrix-based OP protocols. According to our experiment on 128-bit input data, depending on the input size and network environments, our network-based protocol can reduce communication costs by 50% and achieve a $1.32\times - 1.71\times$ faster execution time compared to its baseline, while our matrix-based protocol is $1.2\times - 1.6\times$ faster than its baseline. We also present a selection strategy for choosing the optimal protocol under various scenarios through theoretical and experimental analysis. The key contributions of this paper are as follows.

- We propose an optimized network-based COP generation protocol that reduces the communication/computation costs by nearly half compared to prior work. The main idea is that when using the random oblivious transfer (ROT) to implement a switch in the permutation network, the sum of the receiver's two possible output pairs for that switch remains constant regardless of his choice bit. When the receiver obtains one of two outputs of that switch, knowing the sum of the output pairs is enough for the receiver to compute the other output. The communication cost is thus reduced by a factor of 2. We also argue that our switch protocol is communication-optimal if the switch is implemented with the ROT. We further adopt the Waksman network [41], which requires fewer switches compared to the Beneš network [7] used in prior work.
- We propose an optimized matrix-based COP generation protocol. For small-size COP, we halve the communication cost by adopting the optimized half-tree single-point correlated oblivious transfer (COT) and remove the communication cost for

[1]There have been efforts to develop malicious secure OP protocols [18, 40, 43]. See §6 for details. Optimizing OP protocols in the malicious setting is more challenging, and we leave this as an open problem.

constructing the last row of matrices based on the special property of permutations. Consequently, the number of COT and the communication cost for transferring OT payloads are reduced by factors of $\frac{1}{n}$ and $\frac{n-1}{2n}$, respectively. For large-size COP, we find that decomposing the permutation from the middle layer to the outer layers (i.e., in-outside decomposition) incurs smaller costs compared to decomposing from the outer layers to the middle layer (i.e., out-inside decomposition).
- We implement both our optimized protocols and the baseline protocols. We intend to make our implementation open-source. Comprehensive comparisons and analyses are conducted based on experimental results regarding various input sizes and network environments. Based on these results, we present a strategy for selecting the optimal COP protocol. Additionally, we apply the strategy-recommended (network-based) COP protocol on the SOTA shuffle-based PSU protocols, which are the most representative applications of OP. This results in new shuffle-based PSU protocols under the offline-online paradigm with an improved OP building block, maintaining extremely fast online performance.

2 Background and Preliminary

2.1 Notation

We denote the computational security parameter as κ and the statistical security parameter as λ. Let $[a, b] := \{a, \ldots, b\}$ and $[b]$ be a shorthand for $[1, b]$. We use $\perp$ to represent null, $\bar{c}$ to denote $1 + c$ for $c \in \mathbb{F}_2$, $\mathsf{negl}(\cdot)$ to denote a negligible function, and $y \leftarrow \mathbb{Y}$ to denote random sampling of y from the domain $\mathbb{Y}$. We denote $X \overset{c}{\equiv} Y$ if two distributions X, Y are computationally indistinguishable.

We use bold lowercase letters (*e.g.*, $\mathbf{v}$) for vectors and bold uppercase letters (*e.g.*, $\mathbf{M}$) for matrices. We denote $\mathbf{v}_i$ or $\mathbf{v}[i]$ as the i-th element of $\mathbf{v}$, and $\mathbf{v}_{i:j}$ as $\{\mathbf{v}_i, \ldots, \mathbf{v}_j\}$ for $1 \le i \le j \le |\mathbf{v}|$. Given a two-dimensional matrix $\mathbf{M}$, we denote $\mathbf{M}_i$ as its i-th row vector, and $\mathbf{M}_{i,j}$ as the j-th element of its i-th row vector. $\mathbf{v} \otimes \mathbf{u}$ is the shorthand for the element-wise operation $\otimes$ of two vectors $(\mathbf{v}_1 \otimes \mathbf{u}_1, \ldots, \mathbf{v}_n \otimes \mathbf{u}_n)$, and $\mathbf{v} \otimes c$ is the shorthand for $(\mathbf{v}_1 \otimes c, \ldots, \mathbf{v}_n \otimes c)$. $I(n, i)$ denotes a length-n vector whose i-th entry is 1 and others are 0.

A size-n permutation $\pi : [n] \to [n]$ is a bijective function, and we denote $\pi(i)$ as its i-th element. Applying a permutation π to a vector $\mathbf{v}$ results in $\pi(\mathbf{v}) = (\mathbf{v}_{\pi(1)}, \ldots, \mathbf{v}_{\pi(n)})$, which we also write as $\pi \cdot \mathbf{v}$. The inverse of a permutation π is denoted by π^{-1}, satisfying $\pi^{-1}(\pi(i)) = i$ for $i \in [n]$. We denote the set of all size-n permutations as $\mathbb{P}_n$.

2.2 Security Model

This paper focuses on the semi-honest two-party computational security model [29]. We consider a static probabilistic polynomial-time (PPT) semi-honest adversary, who can corrupt either the sender $\mathcal{S}$ or the receiver $\mathcal{R}$ at the beginning of the protocol. The adversary may attempt to learn information from the transcript of the protocol while following the prescribed protocol faithfully. Let $\mathcal{P}$ be a protocol for computing a probabilistic polynomial-time functionality $\mathcal{F} : (\{0,1\}^*)^2 \to (\{0,1\}^*)^2$, and let $\mathcal{F}_i(x_0, x_1)$ denotes the i-th element of $\mathcal{F}(x_0, x_1)$. The view of party P_i during an execution of $\mathcal{P}$ on (x_0, x_1) is $\mathsf{view}_i^{\mathcal{P}}(x_0, x_1)$, including P_i's input x_i and all messages received during the protocol. The output of both parties during an execution of $\mathcal{P}$ on x_0, x_1 is denoted by $\mathsf{out}^{\mathcal{P}}(x_0, x_1)$.

Initialize: Upon receiving (init, Δ) from a sender $\mathcal{S}$ where global key $\Delta \in \mathbb{F}_{2^\kappa}$, and (init) from a receiver $\mathcal{R}$, store Δ and ignore all subsequent (init) commands.
Extend: Upon receiving (extend, n) from $\mathcal{S}$ and $(\mathsf{extend}, \mathbf{u})$ from $\mathcal{R}$ where $\mathbf{u} \in \{0,1\}^n$:
(1) If $\mathcal{S}$ is honest, sample $\mathbf{v} \leftarrow \mathbb{F}^n_{2^\kappa}$; Otherwise, receive $\mathbf{v} \in \mathbb{F}^n_{2^\kappa}$ from the adversary.
(2) If $\mathcal{R}$ is honest, Compute $\mathbf{w} = \mathbf{v} \oplus \mathbf{u} \cdot \Delta \in \mathbb{F}^n_{2^\kappa}$; Otherwise, receive $\mathbf{w} \in \mathbb{F}^n_{2^\kappa}$ from the adversary and recompute $\mathbf{v} = \mathbf{w} \oplus \mathbf{u} \cdot \Delta$.
(3) Send $\mathbf{v}$ to $\mathcal{S}$ and $\mathbf{w}$ to $\mathcal{R}$.

Figure 1: Functionality for Correlated OT $\mathcal{F}_{\mathsf{COT}}$.

Definition 1 (Simulation-based security). *A protocol $\mathcal{P}$ securely computes $\mathcal{F}$ if there exist PPT simulators Sim_i where $i \in \{0,1\}$, for every $(x_0, x_1) \in (\{0,1\}^*)^2$ such that:*

$$\{\mathsf{Sim}_i(x_i, \mathcal{F}_i(x_0, x_1)), \mathcal{F}(x_0, x_1)\} \overset{c}{\equiv} \{\mathtt{view}_i^{\mathcal{P}}(x_0, x_1), \mathtt{out}^{\mathcal{P}}(x_0, x_1)\}$$

As stated in the above definition, the simulation-based proof for a specific protocol is to construct a simulation for each party such that this simulator can generate messages indistinguishable from those produced during the execution of the real protocol.

2.3 Oblivious Transfer

Oblivious transfer (OT) [14, 32, 35] is a core building block in various MPC protocols. In a basic 1-out-of-2 OT_l, the sender inputs two l-bit strings (s_0, s_1), and the receiver inputs a bit $c \in \{0,1\}$ and receives s_c. In practical applications, OT is commonly implemented using Random OT (ROT), which returns two random strings (r_0, r_1) to the sender and r_c to the receiver. The sender should further send $(r_0 \oplus s_0, r_1 \oplus s_1)$ to the receiver to recover the true OT payload, and this process is called OT correction. As a special OT flavor, correlated OT (COT) allows the sender to input Δ and receive two correlated random strings r_0, r_1 where $r_1 = r_0 \oplus \Delta$. With COT, ROT can be computed with correlation robust hash function (CRHF) [23]. We also denote COT^n_l as n parallel COTs on l-bit strings. A variant of COT^n_l, where the Hamming weight of the choice-bit vector is 1, has been well studied as the single-point COT [10]. It can be efficiently implemented using the Goldreich-Goldwasser-Micali (GGM) tree [21] and $\mathsf{COT}^{\log n}_\kappa$ with $O(\kappa \log n)$-bit communication cost. We present the functionality of COT in Figure 1. If $\mathbf{u} = I(n, \alpha)$ where $\alpha \in [n]$, we obtain the functionality of single-point COT $\mathcal{F}_{\mathsf{spCOT}}$.

To improve the efficiency, the IKNP-style OT extension [23, 27] is proposed to generate a large number of OTs based on a few base OTs. But it still takes the communication cost of $O(n\kappa)$ bits for COT^n_κ. Recently, it has been superseded by the silent OT line of work based on primal-LPN assumption [39, 44] or dual-LPN assumption [9, 15, 38]. Compared to IKNP-style OT extension, silent OT can generate many OTs with few communication costs, but requires a relatively high computational cost. Thus, the optimal OT implementation may vary in different network and computational environments. For completeness, we will separate the cost of OT

Parameters: The target length n and the ring $\mathbb{D}$ of COP. $\mathcal{R}$ has the size-n permutation ϕ.
Functionality: Upon receiving π from $\mathcal{R}$, uniformly samples three vectors $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{D}^n$ such that $\mathbf{c} = \phi(\mathbf{a}) - \mathbf{b}$. Send $\mathbf{a}, \mathbf{b}$ to $\mathcal{S}$ and $\mathbf{c}$ to $\mathcal{R}$.

Figure 2: Functionality for generating correlated oblivious permutation $\mathcal{F}_{\text{cop}}$.

$\mathcal{S} : (\mathbf{a}, \mathbf{b}), \mathbf{x}$		$\mathcal{R} : (\mathbf{c}, \phi), \pi$
$\mathbf{o} = \mathbf{x} - \mathbf{a}$	$\overset{\mathbf{o}}{\underset{\rho}{\rightleftharpoons}}$	$\rho = \pi(\phi^{-1})$
Output $\mathbf{y} = \rho(\mathbf{b})$		Output $\mathbf{z} = \rho(\mathbf{c}) + \pi(\mathbf{o})$

Figure 3: Protocol to obtain $\mathbf{z} = \pi(\mathbf{x}) - \mathbf{y}$ based on the random COP $\mathbf{c} = \phi(\mathbf{a}) - \mathbf{b}$ with the same size. For the second case when $\pi = \phi$, ρ is $(1, \ldots, n)$ and does not need to be sent, and the outputs are $\mathbf{y} = \mathbf{b}$ and $\mathbf{z} = \mathbf{c} + \pi(\mathbf{o})$.

in the subsequent theoretical complexity analysis and evaluate our protocols with both implementations.

2.4 Online-offline Oblivious Permutation

The ideal functionality for generating COP is shown in Figure 2. In an offline-online paradigm [36], a deterministic OP that computes the share of $\pi(\mathbf{x})$ taking the input π from the receiver and $\mathbf{x}$ from the sender can be realized with a small online cost using the offline-generated COP, and there are two usage cases of COP.

(1) The permutation ϕ of the COP is a random permutation sampled by $\mathcal{R}$ and is different from the input permutation π, the online permutation protocol is described in Figure 3. It requires $n(l + \log n)$ bits of communication, where n is the input size and l is the bit length of each input element.
(2) The permutation of the COP is the same as the permutation π, then the online permutation phase only requires the communication cost of nl bits.

2.5 Network-based Oblivious Permutation

The permutation network is an arrangement of switches and wires allowing n inputs to be simultaneously connected to n outputs via edge-disjoint paths. For any one-to-one permutation π of n inputs and n outputs, there exists a set of edge-disjoint paths connecting the i-th input to the $\pi(i)$-th output for all $i \in [n]$. Here, we focus on binary permutation networks constructed from binary switches, each of which can be in one of two states: direct/crossed connection. Since there are $n!$ possible permutations with n inputs and n outputs, it follows that at least $\lceil \log_2(n!) \rceil \approx n \log_2 n - 1.443n$ switches are required to realize such permutation π. The first permutation network was designed by Beneš [7]. The network is recursively programmed, where a size-2 network requires one switch, and a size-n ($n \geq 4, n = 2^r$) network can be built with two smaller networks that support arbitrary size-$\frac{n}{2}$ permutations, along with additional n switches. As a result, Beneš network can program any size-n permutation using a total of $n \log_2 n - \frac{n}{2}$ switches. Beneš network is thus *asymptotically* optimal in terms of the switch count. Chang and

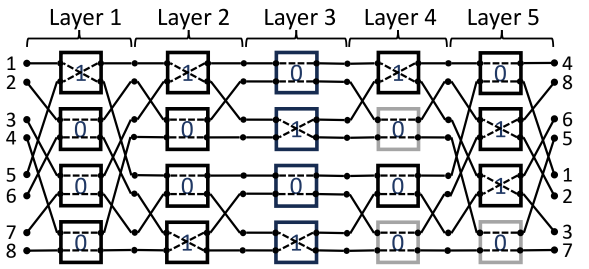


Figure 4: An example of size-8 in-place Beneš network where $\pi = (4, 8, 6, 5, 1, 2, 3, 7)$. It has 5 layers. Each box is a switch, where two dotted lines connect the input and output depending on the programmed bit (0/1). The gray switches can be omitted in the Waksman network.

Melhem [12] later generalized Beneš network for any size n, which has been used to construct efficient network-based COP protocols supporting arbitrary permutation sizes [20].

A permutation network is an in-place network if each switch rearranges each pair of input data within the same two locations. To facilitate the description of the permutation decomposition in the following matrix-based approach, we describe the in-place version of Beneš network [13] for consistency[2]. An example of size-8 in-place Beneš permutation network is shown in Figure 4.

Now, we briefly review the prior network-based OP protocol [34]. Given a size-n vector $\mathbf{x}$ from the sender $\mathcal{S}$ and a permutation $\pi \in \mathbb{P}_n$ from the receiver $\mathcal{R}$, the shares of $\pi(\mathbf{x})$ is obtained through the Beneš network in the following steps [34]. First, $\mathcal{S}$ chooses random masks for all m wires in the Beneš network. We denote these masks as a vector $\mathbf{m}$, where $\{\mathbf{m}_i\}_{i \in [n]}$ are masks for input wires, and $\{\mathbf{m}_i\}_{i \in [m-n+1, m]}$ are masks for output wires. Second, the parties perform an 1-out-of-2 OT for each switch g with input wires $i0, i1$ and output wires $j0, j1$, where $\mathcal{S}$'s input for the OT is $\big((\mathbf{m}_{j0} - \mathbf{m}_{i0} || \mathbf{m}_{j1} - \mathbf{m}_{i1}), (\mathbf{m}_{j0} - \mathbf{m}_{i1} || \mathbf{m}_{j1} - \mathbf{m}_{i0})\big)$ and $\mathcal{R}$'s input bit is the corresponding programming bit for this switch. $\mathcal{R}$ sets the masks of wires as the corresponding OT results. Next, $\mathcal{S}$ sends the masked input $\{x_i - \mathbf{m}_i\}_{i \in [n]}$ to $\mathcal{R}$, and $\mathcal{R}$ sets them as the masks of his input wires. Finally, $\mathcal{S}$ outputs the masks of output wires $\{\mathbf{m}_i\}_{i \in [m-n+1, m]}$. $\mathcal{R}$ identifies the path from the output wire to the corresponding input wire, and his output is obtained by adding all corresponding masks together. Therefore, a switch needs 1 ROT and a further $4l$ bit of communication for OT payload correction.

Instead of first choosing random masks for a switch's output wire(s) and using those to determine the OT payloads, Garimella et al. [20] suggested that the sender can use one of the two ROT results to dictate the first OT payload, and further use that to compute the values of two output wires. In this way, a switch is implemented with 1 COT, $2l$ bits communication, and 3 invocations of CRHF whose output length is $2l$ bits (used to extend the COT result into ROT result).

2.6 Matrix-based Oblivious Permutation

The matrix-based OP solution involves using puncturable pseudo-random functions (PPRF) to generate a permutation matrix.

[2]The in-place and the original versions have the same number of switches, and their structures are similar.

Definition 2 (Puncturable Pseudorandom Function (PPRF)). *A PPRF [10] with key space $\mathbb{K}$, punctured key space $\mathbb{K}_P$, size n, and range $\mathbb{Y}$, has the following syntax:*

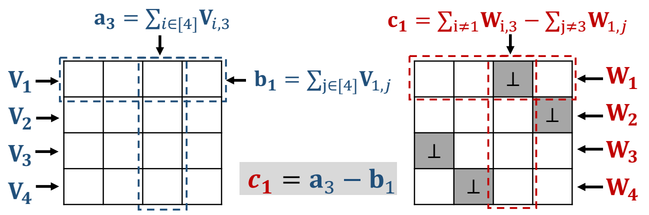
- PPRF.Gen(1^λ)*: On input* 1^λ*, output a random key* $k_{\mathsf{pprf}} \in \mathbb{K}$.
- PPRF.Punc$(k_{\mathsf{pprf}}, \alpha)$*: On input a key* $k_{\mathsf{pprf}} \in \mathbb{K}$ *and a punctured point* $\alpha \in [n]$*, output a punctured key* $k_{\mathsf{pprf}}\{\alpha\} \in \mathbb{K}_P$.
- PPRF.Eval$(\alpha, k_{\mathsf{pprf}}\{\alpha\}, x)$*: On input a punctured key* $k_{\mathsf{pprf}}\{\alpha\} \in \mathbb{K}_P$ *and a point* $x \in [n]$*, output the result* PPRF$(k_{\mathsf{pprf}}, x) \in \mathbb{Y}$ *if* $x \neq \alpha$*; otherwise output* $\perp$.

A PPRF *is secure if for any PPT adversary* Adv *and any point* $\alpha \in [n]$ *chosen by* Adv*, it holds that given* $k_{\mathsf{pprf}} \leftarrow$ PPRF.Gen(1^λ), $k_{\mathsf{pprf}}\{\alpha\} \leftarrow$ PPRF$(k_{\mathsf{pprf}}, \alpha)$*:*

$$\left| \begin{array}{l} \Pr\left[\mathsf{Adv}(1^\lambda, \alpha, k_{\mathsf{pprf}}\{\alpha\}, \mathsf{PPRF}(k_{\mathsf{pprf}}, \alpha)) = 1\right] \\ - \Pr\left[\mathsf{Adv}(1^\lambda, \alpha, k_{\mathsf{pprf}}\{\alpha\}, y^*_\alpha \leftarrow \mathbb{Y}) = 1\right] \end{array} \right| \leq \mathsf{negl}(\lambda)$$

PPRF can be built from any length-doubling PRG, using the GGM tree [21] and $\mathsf{COT}_\kappa^{\lceil \log n \rceil}$ with $2\kappa\lceil \log n \rceil$ bits communication.

Chase et al. [13] proposed the matrix-based OP using PPRF. For a size-n permutation, two parties invoke n instances of PPRF where $\mathcal{R}$ inputs $\pi(i)$ for the i-th execution. As a result, $\mathcal{S}$ and $\mathcal{R}$ obtain $n \times n$ matrices $\mathbf{V}$ and $\mathbf{W}$ respectively, where $\mathbf{V}_{i,j} = \mathbf{W}_{i,j}$ for all $i \in [n], j \neq \pi(i)$ and $\mathbf{W}_{i,\pi(i)} = \perp$. Finally, $\mathcal{S}$ outputs two vectors $\mathbf{a}, \mathbf{b}$ where $\mathbf{a}_i = \sum_{j \in [n]} \mathbf{V}_{j,i}, \mathbf{b}_i = \sum_{j \in [n]} \mathbf{V}_{i,j}$ and $\mathcal{R}$ outputs a vector $\mathbf{c}$ where $\mathbf{c}_i = \sum_{j \neq i} \mathbf{W}_{j,\pi(i)} - \sum_{j \neq \pi(i)} \mathbf{W}_{i,j}$ such that $\mathbf{c} = \pi(\mathbf{a}) - \mathbf{b}$. Figure 5 illustrates a toy example with $n = 4$.



$\mathbf{a}_3 = \sum_{i \in [4]} \mathbf{V}_{i,3}$
$\mathbf{c}_1 = \sum_{i \neq 1} \mathbf{W}_{i,3} - \sum_{j \neq 3} \mathbf{W}_{1,j}$
$\mathbf{V}_1$
$\mathbf{V}_2$
$\mathbf{V}_3$
$\mathbf{V}_4$
$\mathbf{b}_1 = \sum_{j \in [4]} \mathbf{V}_{1,j}$
$c_1 = \mathbf{a}_3 - \mathbf{b}_1$
$\perp$
$\mathbf{W}_1$
$\mathbf{W}_2$
$\mathbf{W}_3$
$\mathbf{W}_4$

Figure 5: A toy example of a matrix-based OP with $\pi = (3, 4, 1, 2)$. Two matrices, $\mathbf{V}$ and $\mathbf{W}$, are obtained by $\mathcal{S}$ and $\mathcal{R}$, respectively. $\mathcal{S}$ holds the values colored blue, and the receiver $\mathcal{R}$ holds the values colored red.

However, since the computation complexity is $O(n^2)$, such matrix-based OP protocol is computation-expensive for large n. To adapt it for large-size permutations, instead of directly generating a COP for the entire permutation π, Chase et al. [13] proposed that two parties first decompose the permutation into small sub-permutations, run COPs for each sub-permutation, and then combine them back to a large COP for the original permutation π.

3 Improving Network-based COP

We consider two optimization techniques for network-based COP. The first optimization comes from the observation that existing network-based COP does not make full use of correlations generated in ROT. In the initial network-based COP construction [34], $\mathcal{S}$ generates random masks for all wires in the permutation network. Then, for each switch, $\mathcal{S}$ and $\mathcal{R}$ perform one 1-out-of-2 OT to obliviously transfer two mask correlations. Garimella et al. [20] proposed that $\mathcal{S}$ generates random masks according to *one of its ROT outputs* to reduce the communication cost of a switch by half. This raises the possibility of further halving the communication cost by leveraging *two ROT outputs*. We show that this is feasible.

The second optimization is to use an improved permutation network with fewer switches. Since each switch requires one (R)OT invocation in network-based COP, reducing the number of switches in the permutation network structure leads to a reduction in the total number of OT invocations, thereby decreasing the overall computation/communication costs. We choose the Waksman network [5], reducing at most $\frac{n}{2} - 1$ switches compared with Beneš network [12] in the best case. The details are shown in Appendix B.

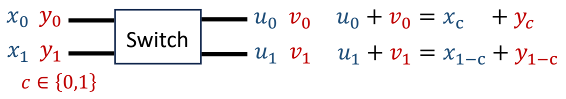


Figure 6: The input (in left) and output (in right) of a switch for secret-shared values. $\mathcal{S}$ holds the values colored blue, and $\mathcal{R}$ holds the values colored red.

3.1 Halving the Cost of Implementing Switches

Let us review the basic idea of network-based COP solutions by taking the protocol implementing the switch (shown in Figure 6) as the fundamental building block [20, 34]. The pipeline of switch protocol optimizations (including ours) is illustrated in Figure 7.

In the switch protocol, $\mathcal{S}$ holds (x_0, x_1) and $\mathcal{R}$ holds (y_0, y_1) where $(x_0 + y_0, x_1 + y_1)$ is the plaintext pair. Given a choice bit c owned by $\mathcal{R}$, the target is to let $\mathcal{S}$ and $\mathcal{R}$ respectively obtain (u_0, u_1) and (v_0, v_1), such that

- If $c = 0$, then $u_0 + v_0 = x_0 + y_0$ and $u_1 + v_1 = x_1 + y_1$ (output shares are not switched);
- If $c = 1$, then $u_0 + v_0 = x_1 + y_1$ and $u_1 + v_1 = x_0 + y_0$ (output shares are switched).

Mohassel and Sadeghian [34] implemented such a switch protocol using a single 1-out-of-2 OT. To achieve this, $\mathcal{S}$ first generates random (u_0, u_1). Two parties then invoke 1-out-of-2 OT, where $\mathcal{S}$ takes $((x_0 - u_0) \| (x_1 - u_1), (x_1 - u_0) \| (x_0 - u_1))$ as inputs and $\mathcal{R}$ takes the choice bit c as input. $\mathcal{R}$ finally sets

$$(v_0, v_1) = \begin{cases} ((x_0 - u_0) + y_0, (x_1 - u_1) + y_1) & \text{if } c = 0 \\ ((x_1 - u_0) + y_1, (x_0 - u_1) + y_0) & \text{if } c = 1 \end{cases}$$

It is easy to verify (u_0, u_1) and (v_0, v_1) satisfy the desired correlation. In implementation, the above process is realized with 1 ROT_{2l} and $4l$ bits communication for OT correction, as shown on the left side of Figure 7. One can leverage this protocol for every switch in the permutation network to obtain network-based COP.

Garimella et al. [20] identified redundancy in using ROT to instantiate the above switch protocol. ROT generates randomness (r_0, r_1) for $\mathcal{S}$, which are then used as one-time keys to encrypt $(x_0 - u_0) \| (x_1 - u_1)$ and $(x_1 - u_0) \| (x_0 - u_1)$. Given that r_0 is randomly generated via ROT, it is not necessary to use r_0 to further encrypt uniformly random $(x_0 - u_0) \| (x_1 - u_1)$. Instead, $\mathcal{S}$ can directly assign $(x_0 - u_0) \| (x_1 - u_1) = r_0$. Applying this optimization saves half the communication cost of a switch.

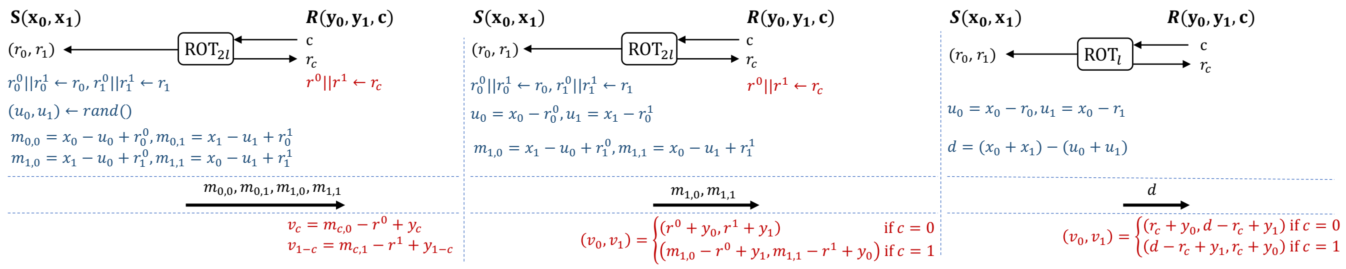

S(x0, x1)
R(y0, y1, c)
ROT2l
(r0, r1)
c
rc
r0^0||r0^1 ← r0, r1^0||r1^1 ← r1
r^0||r^1 ← rc
(u0, u1) ← rand()
m0,0 = x0 − u0 + r0^0, m0,1 = x1 − u1 + r0^1
m1,0 = x1 − u0 + r1^0, m1,1 = x0 − u1 + r1^1
m0,0, m0,1, m1,0, m1,1
vc = mc,0 − r^0 + yc
v1−c = mc,1 − r^1 + y1−c
S(x0, x1)
R(y0, y1, c)
ROT2l
(r0, r1)
c
rc
r0^0||r0^1 ← r0, r1^0||r1^1 ← r1
r^0||r^1 ← rc
u0 = x0 − r0^0, u1 = x1 − r0^1
m1,0 = x1 − u0 + r1^0, m1,1 = x0 − u1 + r1^1
m1,0, m1,1
(v0, v1) = (r^0 + y0, r^1 + y1) if c = 0; (m1,0 − r^0 + y1, m1,1 − r^1 + y0) if c = 1
S(x0, x1)
R(y0, y1, c)
ROTl
(r0, r1)
c
rc
u0 = x0 − r0, u1 = x0 − r1
d = (x0 + x1) − (u0 + u1)
d
(v0, v1) = (rc + y0, d − rc + y1) if c = 0; (d − rc + y1, rc + y0) if c = 1

Figure 7: Comparison of the construction for a switch in three network-based methods: MS13 [34], GMR21 [20] and our $\mathcal{P}_{\text{cop-net}}$, from left to right. For MS13 and GMR21, $((r_0, r_1), r_c)$ is the output of ROT_{2l} where the choice bit of $\mathcal{R}$ is c and the bit length of input is l; while for our $\mathcal{P}_{\text{cop-net}}$, they are output of ROT_l. The calculations of $\mathcal{S}$ and $\mathcal{R}$ are marked in blue and red, respectively.

Parameters: The size of input n. The field of input $\mathbb{D}$. A CRHF $H : \{0,1\}^\kappa \to \mathbb{D}$.
Input: $\mathcal{R}$ holds a permutation $\pi \in \mathbb{P}_n$.
Protocol:

(1) **[Programming]** $\mathcal{R}$ programs a Waksman network ω with the size-n permutation π, where the number of switches in ω is $W(n)$.
(2) Let p be a public function that maps each switch g to a distinct index $p(g) \in [W(n)]$, $\mathcal{R}$ obtains a length-$W(n)$ vector $\mathbf{c}$ such that the sign for each switch gate g is placed into $\mathbf{c}_{p(g)}$.
(3) **[OT]** Two parties initialize $\mathcal{F}_{\text{COT}}$ with sender sampled Δ. Then they invoke $\mathcal{F}_{\text{COT}}$ where $\mathcal{S}$ acts as sender and $\mathcal{R}$ acts as the receiver with input $\mathbf{c}$. As the result, $\mathcal{S}$ receives length-$W(n)$ vector $\mathbf{r}$ and $\mathcal{R}$ receives two length-$W(n)$ vectors $\mathbf{r}^0, \mathbf{r}^1$.
(4) **[Network construction]** Two parties construct own network. Let $\mathbf{u}$ and $\mathbf{v}$ be the values of wires held by $\mathcal{S}$ and $\mathcal{R}$ respectively. For each input wires in the first layer i, $\mathcal{S}$ samples $\mathbf{u}_i \leftarrow \mathbb{D}$, $\mathcal{R}$ sets $\mathbf{v}_i = 0$.
(5) In topological order of wires: let g be a switch with input wires $i0, i1$ and output wires $j0, j1$.
    (a) $\mathcal{S}$ computes $s_b \leftarrow H(\mathbf{r}^b_{p(g)})$ for $b \in \{0,1\}$, and sets $\mathbf{u}_{j0} = \mathbf{u}_{i0} - s_0, \mathbf{u}_{j1} = \mathbf{u}_{i0} - s_1$.
    (b) $\mathcal{S}$ computes and sends $d = \mathbf{u}_{i0} + \mathbf{u}_{i1} - \mathbf{u}_{j0} - \mathbf{u}_{j1}$ to $\mathcal{R}$.
    (c) $\mathcal{R}$ computes $s \leftarrow H(\mathbf{r}_{p(g)})$. If $\mathbf{c}_{p(g)} = 0$, $\mathcal{R}$ sets $\mathbf{v}_{j0} = s + \mathbf{v}_{i0}, \mathbf{v}_{j1} = d - s + \mathbf{v}_{i1}$; Otherwise, $\mathcal{R}$ sets $\mathbf{v}_{j0} = d - s + \mathbf{v}_{i1}, \mathbf{v}_{j1} = s + \mathbf{v}_{i0}$.

Output: $\mathcal{S}$ outputs length-n vectors $\mathbf{a}, \mathbf{b}$, where $\mathbf{a}_i$ is the value of the i-th input wire of the first layer and $\mathbf{b}_i$ is the value of i-th output wire of the last layer. $\mathcal{R}$ outputs length-n vectors $\mathbf{c}$ where $\mathbf{c}_i$ is the value of i-th output wire of the last layer.

Figure 8: COP generation protocol based on Waksman network $\mathcal{P}_{\text{cop-net}}$.

$$\underbrace{(x_0 - u_0 || x_1 - u_1)}_{r_0}, (x_1 - u_0 || x_0 - u_1)$$

We further observe that there is another correlation between $(x_0 - u_0)||(x_1 - u_1)$ and $(x_1 - u_0)||(x_0 - u_1)$, that is,

$$(x_0 - u_0) + (x_1 - u_1) = (x_1 - u_0) + (x_0 - u_1)$$

We can leverage this correlation to further halve the communication cost by determining (u_0, u_1) on both (r_0, r_1) instead of solely on r_0. Specifically, $\mathcal{S}$ computes $u_0 = x_0 - r_0$, $u_1 = x_0 - r_1$. Subsequently, $\mathcal{S}$ only need to send $d = x_0 + x_1 - u_0 - u_1$ to $\mathcal{R}$.

$$\overbrace{(\underbrace{x_0 - u_0}_{r_0} || x_1 - u_1)}^{sum=d}, \overbrace{(x_1 - u_0 || \underbrace{x_0 - u_1}_{r_1})}^{sum=d}$$

$\mathcal{R}$ finally sets $(v_0, v_1) = \begin{cases} (r_c + y_0, d - r_c + y_1) & \text{if } c = 0 \\ (d - r_c + y_1, r_c + y_0) & \text{if } c = 1 \end{cases}$.

It is easy to verify that (u_0, u_1) and (v_0, v_1) satisfy the desired correlation. $\mathcal{S}$ only needs to send $d = x_0 + x_1 - u_0 - u_1$, which further halves the communication cost compared to sending the encrypted $(x_1 - u_0)||(x_0 - u_1)$ as in [20]. The randomness of the outputs (u_0, u_1) and (v_0, v_1) for two parties is guaranteed by the randomness of (r_0, r_1). The message d sent by $\mathcal{S}$ can be inferred from the result that $\mathcal{R}$ should get, thereby demonstrating the security.

Now, we informally analyze the lower bound of the communication cost for implementing a switch based on a ROT result. First, the receiver knows nothing about (x_0, x_1) before the switch protocol. After the switch is somewhat done with a ROT result, the receiver can obtain $\sum x_i - \sum u_i$ by computing $\sum v_i - \sum y_i$. This implies that the receiver learns l-bit information of the sender's input (x_0, x_1) and output (u_0, u_1). Second, the input (x_0, x_1) of the sender can be any two l-bit random values, and the ROT result is random and independent with (x_0, x_1), so $\sum x_i - \sum u_i$ can be any l-bit value no matter how the sender generates random values (u_0, u_1). Therefore, if security is not compromised, the message transmitted during the switch protocol must be no less than l-bit from the perspective of information theory. Therefore, we argue that the lower bound of the communication cost for implementing a switch for l-bit input based on an ROT result is l bits, and our protocol is communication-optimal.

3.2 Our network-based protocol

Our optimized network-based COP is formally described in Figure 8. The protocol follows the above idea but uses slightly different notations. Note that in the permutation network, all switches are organized into layers, with the output of switches in each layer serving as the input to switches in the next layer. Therefore, we use vectors $\mathbf{u}$ and $\mathbf{v}$ to denote wire values held by $\mathcal{S}$ and $\mathcal{R}$, respectively, and subscript indexes to identify wire values for a particular switch g. In this way, the input (and output) of a switch $(\mathbf{u}_{i0}, \mathbf{u}_{i1}), (\mathbf{v}_{i0}, \mathbf{v}_{i1})$ (and $(\mathbf{u}_{j0}, \mathbf{u}_{j1}), (\mathbf{v}_{j0}, \mathbf{v}_{j1})$) correspond to $(x_0, x_1), (y_0, y_1)$ (and $(u_0, u_1), (v_0, v_1)$) as described in §3.1. We also leverage CRHF H to extend (r_0, r_1) generated by COT to the ROT result with the same length l as the switch input.

If the offline-online paradigm is not adopted and direct computation of the share of $\pi(\mathbf{x})$ is required, only the following operations need to be added at the end of the protocol in Figure 8: (1) $\mathcal{S}$ computes and sends $\mathbf{x} - \mathbf{a}$ to $\mathcal{R}$, and outputs $\mathbf{b}$; (2) $\mathcal{R}$ computes and outputs $\pi(\mathbf{x} - \mathbf{a}) + \mathbf{c}$. Note that these operations are identical to those of the online phase in the offline-online paradigm when the permutation of offline generated COP is π. Therefore, the cost of permuting $\mathbf{x}$ remains unchanged regardless of whether the offline-online paradigm is used.

4 Improving Matrix-based COP

We consider two optimizations for matrix-based COP. The first optimization is for small-size permutations with lower communication costs, while the second focuses on more efficient permutation decomposition for generating COP in large-size permutations.

4.1 Improving Small-size Permutation

Chase's protocol [13] relies on PPRF constructed using the standard GGM tree. Recently, Guo et al. proposed a new primitive named pseudorandom correlated GGM (pcGGM) tree [22]. By plugging the pcGGM tree into PPRF, the computation/communication costs are reduced by nearly one-quarter and one-half, respectively. A naive way to optimize small-size permutation is thus directly adopting the more efficient pcGGM tree-based PPRF, achieving $n\kappa(\log n + 1)$ (rather than $2n\kappa \log n$) bits of communication.

Our further optimization exploits the property that the permutation $\pi = (\pi(1), \ldots, \pi(n))$ must traverse the entire set $[n]$. This implies that $\pi(n)$ *can be uniquely determined from* $\{\pi(1), \ldots, \pi(n-1)\}$. In the matrix $\mathbf{W}$, this property indicates that the punctured index of the n-th row is the index that has not appeared in the preceding $n-1$ rows. It seems possible to utilize the values of the first $n-1$ rows in $\mathbf{V}$ and $\mathbf{W}$ to determine the entries in their n-th row.

However, directly using PPRF to generate each row in the matrix is not feasible for the above optimization. If all values in punctured points are pseudorandom and unknown to $\mathcal{R}$, for the i-th column with $\pi(n) \neq i$, $\mathcal{S}$ knows the values of the first $n-1$ columns $\{\mathbf{V}_{j,i} | 1 \leq j < n\}$ while $\mathcal{R}$ only learns $n-2$ values of them except $\mathbf{V}_{k,i}$ where $\pi(k) = i$ $(1 \leq k < n)$. Without interaction and $\mathcal{S}$ knowing the index k, two parties cannot obtain the same pseudorandom value for $\mathbf{V}_{n,i}$ and $\mathbf{W}_{n,i}$.

Our solution is to create two matrices $\mathbf{V}$ and $\mathbf{W}$ such that only the pairs of values at punctured positions $(i, \pi(i))$ exhibit a correlation $\mathbf{W}_{i,\pi(i)} = \mathbf{V}_{i,\pi(i)} \oplus \Delta$, where $\mathcal{S}$ knows Δ and $\mathcal{R}$ knows punctured indexes $\pi(i)$. All other pairs of values are the same in $\mathbf{V}$ and $\mathbf{W}$. This allows two parties to compute the same values for columns $i \neq \pi(n)$, which is just enough for the permutation. Subsequently, two parties can calculate the values of the last row and then break that correlation using CRHF to make the column-sum or row-sum pseudorandom, thus meeting the security requirements.

Specifically, two parties now obtain two matrices that satisfy $\mathbf{W}_i = \mathbf{V}_i \oplus I(n, \pi(i)) \cdot \Delta$ for $i \in [n-1]$. Let $\mathbf{W}_n$ and $\mathbf{V}_n$ be the column-wise sum of $\mathbf{W}_{1:n-1}$ and $\mathbf{V}_{1:n-1}$, i.e., $\mathbf{W}_{n,i} = \bigoplus_{j \in [n-1]} \mathbf{W}_{j,i}$ and $\mathbf{V}_{n,i} = \Delta \oplus (\bigoplus_{j \in [n-1]} \mathbf{V}_{j,i})$. If $i \neq \pi(n)$, let $\pi(k) = i$, then $\mathbf{V}_{n,i} = \Delta \oplus (\bigoplus_{j \in [n-1]} \mathbf{V}_{j,i}) = \Delta \oplus \mathbf{V}_{\pi(k),i} \oplus (\bigoplus_{j \in [n-1], j \neq k} \mathbf{W}_{j,i}) = \bigoplus_{j \in [n-1]} \mathbf{W}_{j,i} = \mathbf{W}_{n,i}$. If $i = \pi(n)$, then obviously $\mathbf{V}_{n,i} = \Delta \oplus \mathbf{W}_{n,i}$. Therefore, the constraint of the n-th row $\mathbf{W}_n = \mathbf{V}_n \oplus I(n, \pi(n)) \cdot \Delta$ holds. Finally, n^2 computations of CRHF H on the values of the matrices are performed to break the correlations and ensure that the result vectors $\mathbf{a}', \mathbf{b}', \mathbf{c}'$ are pseudorandom. $\{\mathbf{W}_{i,\pi(i)}\}_{i \in [n]}$ are independent due to the security of $\mathcal{F}_{\text{spCOT}}$, and $\{H(\mathbf{V}_{i,\pi(i)})\}_{i \in [n]}$ are pseudorandom to $\mathcal{R}$ given $\mathbf{V}_{i,\pi(i)} = \mathbf{W}_{i,\pi(i)} \oplus \Delta$ without knowing Δ according to the correlation robustness. Therefore, although the obtained values $\mathbf{W}_n, \mathbf{V}_n$ depend on the first $n-1$ rows and are not random, the obtained $\mathbf{a}', \mathbf{b}'$ are still pseudorandom to $\mathcal{R}$.

Two parties can adopt the half-tree-optimized single-point COT to obtain these two matrices, which requires $\text{COT}_\kappa^{\log n}$ and the communication cost of $\kappa \log n$ bits for a size-n spCOT. The total number of COTs and the communication cost for transferring OT payloads is reduced to $(n-1)\log n$ (instead of $n \log n$) and $(n-1)\kappa \log n$ (instead of $2n\kappa \log n$) bits, respectively. The detailed protocol is formally described in Figure 9. Note that the mask operation in step (7) of Figure 9 is essential to achieve simulation-based security. For a large-size COP generated with permutation decomposition, the same mask operation should be performed on the compiled output vectors. Theorem 1 formally shows the security of our optimization for small-size permutation. The proof is shown in Appendix C.

Theorem 1. *The protocol in Figure 9 securely realizes the functionality in Figure 2 in the $\mathcal{F}_{\text{spCOT}}$-hybrid world.*

4.2 Improving Permutation Decomposition

The other crucial technique in matrix-based COP is to avoid $O(n^2)$ complexity by first decomposing a large-size permutation into multiple small-size sub-permutations, followed by compiling the COPs corresponding to those sub-permutations into the required large-size COP [13]. This section reviews the permutation decomposition method, along with our optimization.

Out-inside Permutation Decomposition. There are two steps to decompose a size-n permutation [13]. The first step is to program the Beneš (or Waksman) network with $2\log n - 1$ layers. Given the upper bound of the size of the decomposed small permutation is T where T is a power of 2, the second step is to split the adjacent layers of the permutation network *from the outer layers to the middle layer*, ensuring that the number of layers in each subnetwork is $\log T$ except for the middle subnetwork. It will result in $d = 2\lceil \frac{\log n}{\log T} \rceil - 1$ subnetworks $\{\omega_1, \ldots, \omega_d\}$. Each ω_i corresponds to a size-n permutation π_i $(i \in [d])$ and $\pi = \pi_d \cdot \ldots, \cdot \pi_1$. We refer to this decomposition strategy as *out-inside decomposition*. The left part of Figure 10 provides a toy example of the out-inside decomposition for a 5-layer network. The network is divided into $2\lceil \frac{\log 8}{\log 4} \rceil - 1 = 3$ subnetworks, where ω_1 or ω_3 is the subnetwork consisting of the switches in the left or right ($\log T = 2$) layers, and ω_2 is a network with the switches in the middle layer.

Parameters: The length of COP $n > 2$. The field of input $\mathbb{D}$. A CRHF $H : \{0,1\}^\kappa \to \mathbb{D}$.
Input: $\mathcal{R}$ holds a permutation $\pi \in \mathbb{P}_n$.
Protocol:

(1) **[spCOT]** $\mathcal{S}$ samples $\Delta \in \mathbb{F}_{2^\kappa}$ and sends (init, Δ) to $\mathcal{F}_{\text{spCOT}}$. $\mathcal{R}$ sends (init) to $\mathcal{F}_{\text{spCOT}}$.
(2) For $i \in [n-1]$: $\mathcal{S}$ sends (extend, n) and $\mathcal{R}$ sends $(\text{extend}, I(n, \pi(i)))$ to $\mathcal{F}_{\text{spCOT}}$, as the result, $\mathcal{S}$ receives length-n vector $\mathbf{V}_i$ and $\mathcal{R}$ receives length-n vector $\mathbf{W}_i$.
(3) **[$\mathcal{S}$ local computation]** $\mathcal{S}$ fill the last row of matrix $\mathbf{V}$. For $i \in [n]$: $\mathbf{V}_{n,i} = \Delta \oplus (\bigoplus_{j\in[n-1]} \mathbf{V}_{j,i})$.
(4) $\mathcal{S}$ computes two length-n vectors $\mathbf{a}', \mathbf{b}'$. For $i \in [n]$: $\mathbf{a}'_i = \sum_{j\in[n]} H(\mathbf{V}_{j,i})$, $\mathbf{b}'_i = \sum_{j\in[n]} H(\mathbf{V}_{i,j})$.
(5) **[$\mathcal{R}$ local computation]** $\mathcal{R}$ fill the last row of matrix $\mathbf{W}$. For $i \in [n]$: $\mathbf{W}_{n,i} = \bigoplus_{j\in[n-1]]} \mathbf{W}_{j,i}$.
(6) $\mathcal{R}$ computes a length-n vector $\mathbf{c}'$. For $i \in [n]$: $\mathbf{c}'_i = \sum_{j\neq i} H(\mathbf{W}_{j,\pi(i)}) - \sum_{j\neq\pi(i)} H(\mathbf{W}_{i,j})$.
(7) **[Mask]** $\mathcal{S}$ samples two length-n vectors $\mathbf{x}, \mathbf{y}$ and sends them to $\mathcal{R}$. $\mathcal{S}$ computes two vectors $\mathbf{a} = \mathbf{a}' + \mathbf{x}$, $\mathbf{b} = \mathbf{b}' + \mathbf{y}$ and $\mathcal{R}$ computes $\mathbf{c} = \mathbf{c}' + \pi(\mathbf{x}) - \mathbf{y}$.

Output: $\mathcal{S}$ outputs $\mathbf{a}, \mathbf{b}$. $\mathcal{R}$ outputs $\mathbf{c}$.

Figure 9: Optimized matrix-based COP generation protocol $\mathcal{P}_{\text{cop-mat}}$.

Depending on whether the switches are interconnected by wires, each subnetwork can be further divided into multiple smaller subnetworks, with the number of input wires for each small subnetwork not exceeding T. For example, the left subnetwork ω_1 on the left subfigure of Figure 10 can be further divided into two disjoint subnetworks (marked in yellow and blue). Each small subnetwork corresponds to a sub-permutation. The same division can be done on the right part of Figure 10. Following these steps, the large permutation can be decomposed into the sub-permutations corresponding to these small subnetworks.

After generating COP for each sub-permutation, the next step is to compile those COPs into the required COP for the original size-n permutation.

(1) For $i \in [d]$, $\mathcal{S}$ and $\mathcal{R}$ combine the COPs of sub-permutations in ω_i to obtain size-n COP. Let $\mathbf{c}^i = \pi_i(\mathbf{a}^i) - \mathbf{b}^i$ be the result.
(2) $\mathcal{S}$ sends $\{\alpha^i = \mathbf{a}^{i+1} - \mathbf{b}^i\}_{i\in[d-1]}$ to $\mathcal{R}$.
(3) $\mathcal{R}$ initializes $\beta^1 = \mathbf{c}^1$ and computes $\beta^{i+1} = \pi_{i+1}(\beta^i - \alpha^i) + \mathbf{c}^{i+1}$ for $i \in [d-1]$.
(4) $\mathcal{S}$ and $\mathcal{R}$ perform the mask operation on $\mathbf{a}^1, \mathbf{b}^d$ and β^d as shown in Figure 9 and output the result vectors.

The correctness of $\beta^d = \pi(\mathbf{a}^1) - \mathbf{b}^d$, where $\pi = \pi_d \cdot \ldots \cdot \pi_1$, can be proven by induction that if $\beta^i = (\pi_i \cdot \ldots \cdot \pi_1)(\mathbf{a}^1) - \mathbf{b}^i$ holds, then

$$\begin{aligned}\beta^{i+1} &= \pi_{i+1}(\beta^i - \alpha^i) + \mathbf{c}^{i+1}\\ &= \pi_{i+1}((\pi_i \cdot \ldots \cdot \pi_1)(\mathbf{a}^1) - \mathbf{b}^i - \mathbf{a}^{i+1} + \mathbf{b}^i) + \mathbf{c}^{i+1}\\ &= (\pi_{i+1} \cdot \ldots \cdot \pi_1)(\mathbf{a}^1) - \pi_{i+1}(\mathbf{a}^{i+1}) + \mathbf{c}^{i+1}\\ &= (\pi_{i+1} \cdot \ldots \cdot \pi_1)(\mathbf{a}^1) - \mathbf{b}^{i+1}.\end{aligned}$$

In-outside Permutation Decomposition. The out-inside permutation decomposition correctly decomposes a size-n permutation into $\frac{nd}{T}$ disjoint size-T sub-permutations when $\log n$ is an integer multiple of $\log T$, i.e., $\log n \bmod \log T = 0$. However, when $s = (\log n \bmod \log T) > 0$, the middle subnetwork $\omega_{\frac{d+1}{2}}$ contains fewer than $2\log T - 1$ layers. Figure 10 demonstrates a toy example where $s = \log 8 \bmod \log 4 = 1$. In this case, $\pi_{\frac{d+1}{2}}$ can be divided into multiple sub-permutations with size $S = 2^s < T$. Since computing $\frac{n}{S}$ size-S COPs requires $n \log S$ OTs and $n\kappa \log S$ bits of communication, compared to $n \log T$ OTs and $n\kappa \log T$ bits of communication for computing $\frac{n}{T}$ size-T COPs, generating these smaller COPs becomes more efficient.

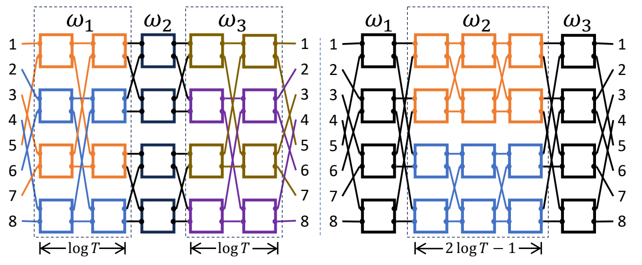


Figure 10: Examples of permutation decomposition given n=8 and T=4. The out-inside method (left) splits into 4 size-4 permutations (in different colors) and 4 size-2 permutations (in black). Our in-outside method (right) splits into 2 size-4 permutations and 8 size-2 permutations.

This observation motivates us to decompose a permutation into as many smaller sub-permutations as possible while keeping the number of decomposed subnetworks d unchanged. When $\log n$ is not an integer multiple of $\log T$, if we adopt the *in-outside decomposition strategy*, where the permutation network is split from the middle layer to the outer layers, it results in 2 subnetworks with sizes less than $\log T$ in the outermost layers, instead of just 1 subnetwork in the middle layer. Therefore, the total cost to construct COP with $\mathcal{P}_{\text{cop-mat}}$ is reduced.

Specifically, let $n = 2^r$, the upper bound of the size of a decomposed small permutation be $T = 2^t$, $s = r - t\lceil \frac{r}{t} \rceil + t$, $S = 2^s$, and $\pi_{i:j}$ denote the permutation corresponding to the subnetwork consisting of layers i to j of the in-place permutation network. Recall the structure of the in-place Beneš network, a switch in the i-th or the $(2r-i)$-th layers ($i \le \log n$) will swap two values that are 2^{r-i} apart. A permutation represented by α layers in Beneš network can be decomposed into $2^{r-\alpha}$ size-2^α sub-permutations. Specifically, the permutation $\pi_{r-j:r-i}$ or $\pi_{r+i:r+j}$ with $0 \le i \le j \le r$ can be decomposed into $2^{r-(j-i+1)}$ sub-permutations. The x-th permutation acts on the data with indexes $\{f_{gid}(x,i) \cdot 2^j + p \cdot 2^i + f_{ind}(x,i)\}_{0 \le p < 2^{j-i+1}}$ where $f_{gid}(x,i) = (x-1)/2^i$, $f_{ind}(x,i) = x - f_{gid}(x,i) \cdot 2^i$ represents which group x will be assigned to when it is grouped by the distance

Table 1: Theoretical complexity comparison. The input size is n and the bit length is l. $B(n)$ and $W(n)$ are the number of switches in the Beneš network and the Waksman network, respectively, for the input size n. T is the size of small-size COP in the matrix-based protocols. $u = \lceil l/\kappa \rceil$, $d = 2\lceil \frac{\log n}{\log T} \rceil - 1$. The computation/communication costs are the total cost of both parties, excluding the cost of COT.

Protocols	No. of COT	No. of AES	Other comm. cost
wPRF-based [36]	$4nu\kappa$	$15nu\kappa$	$20nu\kappa$
Network-based [20]	$B(n)$	$6B(n)u$	$2B(n)l$
Matrix-based [13]	$dn\log T$	$(2u+4)dnT$	$(d+1)nl + 2n\kappa d \log T$
$\mathcal{P}_{\mathsf{cop-net}}$	$W(n)$	$3W(n)u$	$W(n)l$
$\mathcal{P}_{\mathsf{cop-mat}}$	$n\frac{T-1}{T}(2\log n - \log T)$	$(2u+2)dnT$	$(d+1)nl+$ $n\kappa(2\log n - \log T)$

of 2^i, as well as its index within that group. And the permutation of the middle $2i-1$ layers $\pi_{r-(i-1):r+(i-1)}$ is the combination of 2^{r-i} size-2^i permutations, where the x-th permutation acts on the values with indexes $\{(x-1)\cdot 2^i + p\}_{p\in[2^i]}$.

Our in-outside decomposition first decompose the size-n permutation π into d permutations $\pi_{2r-s:2r-1} \cdot \cdots \cdot \pi_{r-(t-1):r+(t-1)} \cdot \ldots \pi_{1:s}$. Then, two permutations $\pi_{1:s}, \pi_{2r-s:2r-1}$ can be decomposed into size-S sub-permutations and the others can be decomposed into size-T sub-permutations. Compared to the out-inside decomposition strategy in [13], the in-outside decomposition strategy yields a lower cost when $s < t$.

5 Theoretical and Experimental Analysis

5.1 Theoretical Comparison

We analyze the complexity of our protocols and compare them with existing methods. The result is shown in Table 1. We list the number of COTs separately since the cost of generating COT in or not in a silent way is different. We measure the computation cost with the number of AES invocations since AES is the base operation of CRHF and PRG. Theoretically, the cost of $\mathcal{P}_{\mathsf{cop-net}}$ is smaller than $\mathcal{P}_{\mathsf{cop-mat}}$ for small input bit length l, and it is reversed for large l.

Network-based COP. Our baseline is the protocol in [20], which requires $B(n)$ COT, where $B(2)=1, B(3)=3$ and $B(n) = B(\lceil \frac{n}{2} \rceil) + B(\lfloor \frac{n}{2} \rfloor) + 2\lfloor \frac{n}{2} \rfloor$. As reviewed in §2.5, a switch needs 3 invocations of CRHF whose output bit length is $2l$ and the transmitted message with $2l$ bits, which results in $6B(n)u$ invocations of AES and $2B(n)l$ bits communication in total, where $u = \lceil \frac{l}{\kappa} \rceil$. $\mathcal{P}_{\mathsf{cop-net}}$ only requires 3 invocations of CRHF whose output bit length is l and l bit message for a switch, thus halving the cost compared to [20]. The number of switches in the Waksman network for a size-n permutation is $W(n)$, where $W(2)=1$, $W(3)=3$ and $W(n) = W(\lceil \frac{n}{2} \rceil) + W(\lfloor \frac{n}{2} \rfloor) + n - 1$ for $n \geq 4$. Therefore, the switches in the Waksman network are *strictly* fewer than the Beneš network when $n \geq 4$.

Matrix-based COP. The protocol in [13] decomposes the size-n permutation into $\frac{nd}{T}$ size-T permutation, and each COP for size-T permutation needs $T\log T$ invocations of OT, $4T^2$ invocations of AES, and $2T\log T\kappa$ bits communication to generate PPRF. With the AES to extend the bit length, a COP requires $(2u+4)T^2$ AES. Combining the communication cost of $(d+1)nl$ bits in the permutation composition process, the number of OT is $dn\log T$, the number of AES invocation is $(2u+4)dnT$, and the total communication cost is $(d+1)nl + 2dn\kappa\log T$ bits.

The reduction of AES compared to [13] comes from the adoption of half-tree GGM construction. Let $t = \log T$, $s = \log n - t\lceil \frac{\log n}{t} \rceil + t$, $S = 2^s$, our smooth permutation decomposition results in 2 s-layers networks and $(d-2)$ t-layers networks, requiring $2ns\frac{S-1}{S}$ and $(d-2)nt\frac{T-1}{T}$ OT, respectively. Therefore, the number of OT and the communication cost in constructing spCOT with half-tree are $n\frac{T-1}{T}(2\log n - \log T)$ and $n\kappa(2\log n - \log T)$ bits.

5.2 Implementation and Configuration

We set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$. As mentioned in §2.3, we evaluated our protocols with two OT implementations: OT extension [4] and silent OT [44].

Implementation details. We fully implement our optimizations using Java programming language, and our code is open source as a submodule in mpc4j[3]. Note that the implementation of the network-based baseline protocol in [20] is publicly available[4]. However, their implementation represents inputs as uint64_t, thus only supporting input bit length $l \leq 64$. We re-implement it to support arbitrary input bit length l. Recently, the implementation of the wPRF-based protocol [36] (based on C++) has been open-sourced[5]. After a thorough analysis of its implementation, we found that it primarily focuses on *computational* performance estimation that supports single-machine evaluations. To facilitate fair and comprehensive comparisons, we implement all protocols and focus on *end-to-end* performance estimation with communication support using Netty[6].

We remark that our implementation of the wPRF-based protocol is slower compared to the original C++ version. On our platform, the C++ version requires approximately 5.6s and 256MB communication cost for generating a 128-bit length COP with size 2^{20}. The performance gap comes from the following aspects. First, Java performs less efficiently than C++ in atom operations, such as XOR operations on byte arrays and AES[7]. Second, although we use Java Virtual Machine (JVM) that supports auto-vectorized (SIMD) optimizations, there remains an efficiency gap when compared to native C++ invoking SIMD directly. Third, our evaluation involves two machines, whereas the C++ version runs on a single machine, rendering the communication overhead almost negligible in their implementation. Fourth, the original C++ version is heavily optimized. While our implementation incorporates some of these optimizations, we have sacrificed some performance for improved code readability. Finally, the communication cost differs from the values reported in Table 2, due to differences in the underlying silent OT protocols. Specifically, we use Ferret OT [44] whereas their implementation uses EcCode [38][8].

[3]https://github.com/alibaba-edu/mpc4j. Our code can be found in the package "osn".
[4]https://github.com/osu-crypto/PSI-analytics
[5]https://github.com/Visa-Research/secure-join
[6]https://netty.io/
[7]On our platform, C++ takes 84ms and 52ms for 2^{25} AES and XOR on 16-byte arrays respectively, while Java takes 609ms and 429ms.
[8]We benchmark the OT generation on one of our machines. When a total of 2^{25} OTs are generated, the EcCode takes 3.5KB on average to generate 2^{20} OT, while Ferret OT takes 157KB to get the same amount of OTs. However, we chose Ferret OT instead of EcCode because Ferret OT is faster. Specifically, the EcCode implemented with

Table 2: Performance for fixed input bit length $l = 128$ bits and different input size n. T is the size of small-size COP in the matrix-based protocols. Since the wPRF-based protocol strongly relies on silent OT, we only report the result with silent OT. The best protocol within a setting is marked in green.

Protocol \ Data size n			Time (s) in LAN setting				Time (s) in WAN setting				Communication cost (MB)			
			2^{12}	2^{16}	2^{20}	2^{24}	2^{12}	2^{16}	2^{20}	2^{24}	2^{12}	2^{16}	2^{20}	2^{24}
OT extension [4]	Matrix-based [13]	$T = 16$	0.1	2.3	54.9	1738	2.4	13.2	208.6	4137	4.3	96.7	1984	38747
		$T = 64$	0.2	2.8	78.4	2291	2.0	12.7	218.4	4438	3.8	100.9	2253	36058
		$T = 256$	0.4	5.5	124.9	3309	1.8	15.1	254.6	5640	4.9	79.9	2119	33906
	Network-based [20]		0.05	0.5	10.1	265.8	1.3	6.2	87.3	1718	2.3	48.8	981.5	18925
	$\mathcal{P}_{\mathrm{cop-mat}}$	$T = 16$	0.1	1.7	38.2	1329	2.1	9.1	135.3	2618	2.7	59.7	1220	23775
		$T = 64$	0.3	2.0	54.0	1447	1.9	9.3	145.8	2846	2.5	56.6	1173	23272
		$T = 256$	0.3	4.1	93.2	2673	1.8	11.1	179.0	4049	2.2	52.9	1112	22288
	$\mathcal{P}_{\mathrm{cop-net}}$		0.05	0.34	6.0	180.3	1.2	4.7	57.1	1125	1.4	31.4	637.5	12348
silent OT [44]	wPRF-based [36]		1.0	5.16	93.4	1516	1.9	12.2	201	3190	1.7	19.4	311	4976
	Matrix-based [13]	$T = 16$	0.9	2.5	59.1	1765	2.2	10.0	174.1	3781	3.6	67.9	1386	27040
		$T = 64$	0.9	3.7	97.2	1853	1.8	10.0	185.6	4095	3.2	70.0	1555	24883
		$T = 256$	1.1	6.0	172.2	3082	1.9	12.8	223.0	5293	4.0	55.3	1454	23264
	Network-based [20]		0.8	1.2	13.0	316.4	1.7	4.6	70.4	1398	2.1	33.2	659.8	12721
	$\mathcal{P}_{\mathrm{cop-mat}}$	$T = 16$	0.7	2.0	41.9	1086	1.9	7.4	108.1	2377	2.2	36.7	743.1	14463
		$T = 64$	0.8	2.7	56.1	1338	1.8	7.0	117.6	2586.9	2.0	33.5	689	13420
		$T = 256$	0.8	4.8	93.8	2554	1.7	8.9	154.1	3747	1.9	30.0	636.4	12477
	$\mathcal{P}_{\mathrm{cop-net}}$		0.7	1.0	9.8	236.0	1.3	3.0	39.0	817.7	1.3	16.4	324.2	6277

Although there are differences in performance, we argue that the performance trends with changes in input data and network bandwidth are consistent, and similar conclusions can be drawn.

Evaluation environment and setup. All protocols are evaluated on two physical machines with Intel® Core™ i9-9900K 3.60GHz CPU and 128GB RAM. The number of available threads for each party is 15. The network settings include actual LAN (where two machines are connected directly by 2.5Gbps network cards and the ping command shows that the RTT latency is 0.4ms), and WAN (100Mbps bandwidth with 80ms RTT latency limited by tc command). We use GraalVM 22.0.2 as our JVM to run our experiments.

Since the difference among COP for various rings of the same bit length lies in the local computation on plaintext (e.g., XOR for $\mathbb{F}_2^l$ and ADD for $\mathbb{F}_{2^l}$) and those computations are very fast, we only evaluate the protocols for data defined in $\mathbb{F}_2^l$ as a representative case. We first evaluate our protocols and baseline protocols with OT extension [4] or with silent OT [44] for the fixed input bit length $l = 128$ (which is the most commonly used setting in the related works [36]) and different input sizes n, the result is shown in Table 2. Then, we evaluate our protocol under various bandwidths and input bit lengths, and the result is shown in Figure 11.

5.3 Performance Evaluation of COP

5.3.1 Network-based COP. As analyzed in §5.1, in addition to COT, the communication/computation cost of $\mathcal{P}_{\mathrm{cop-net}}$ are half of that of the baseline. Therefore, the ratio of communication cost between $\mathcal{P}_{\mathrm{cop-net}}$ and the baseline [20] is approximately 1/2 when using silent OT, or 2/3 when using OT extension. The network-based COP stands out with the minimal requirement for COT among all methods, making its performance less susceptible to the performance variations of the underlying OT, which can be observed by comparing the execution time shown in Table 2. On the other hand, compared to other methods, it has the highest coefficient associated with input bit length l in terms of communication complexity, which is nearly $n \log n$. It significantly associates its performance with the network environment, leading to a substantial performance difference under the LAN and WAN settings.

5.3.2 Matrix-based COP. Our protocol outperforms the baseline, with notable superiority in cases where either (1) the input bit length l is small, or (2) $\log n$ is not a multiple of $\log T$.

Table 3: Communication cost (MB) of matrix-based COP using silent OT for fixed $n = 2^{16}$.

Bit length		2^7	2^9	2^{11}	2^{13}	2^{15}
Baseline [13]	$T = 16$	67.9	93.1	193.7	596.4	2207
	$T = 256$	55.3	67.9	118.2	319.6	1124
$\mathcal{P}_{\mathrm{cop-mat}}$	$T = 16$	36.7	61.9	162.6	565.2	2175
	$T = 256$	30.0	42.6	92.9	294.3	1099

Theoretically, When $l = \kappa = 128$, the additional communication cost besides OT of $\mathcal{P}_{\mathrm{cop-mat}}$ is approximately $\frac{\log T+1}{2\log T+1}\times$ of that of the baseline matrix-based protocol. Our result shown in Table 2 demonstrates it. For example, the ratio of communication cost between $\mathcal{P}_{\mathrm{cop-mat}}$ and the baseline is 0.54 when $n = 2^{24}$ and $T =$

C++ takes 46ms on average to generate 2^{20} OT, while the EcCode and the Ferret OT implemented with Java take 936ms and 246ms, respectively. Therefore, for a fair comparison, we use the fastest one in our implementation, the Ferret OT, as the underlying silent OT protocol for all protocols.

256. Notably, this gain diminishes as l grows, as evidenced in Table 3, e.g., the ratio becomes 0.98 when $n = 2^{16}$, $l = 2^{15}$ and $T = 256$.

Our in-outside decomposition reduces unnecessary computation/communication costs by splitting the permutation network from the middle to both sides and splitting both-side sub-networks into minimal permutations. When l is small, the performance gain of communication cost is apparent. To illustrate this, we configure $\mathcal{P}_{\text{cop-mat}}$ with two decomposition methods and list the ratio of communication costs when $l = 128$ in Table 4. On the other hand, when l is large, the communication cost is dominated by dnl. As shown in Table 3, when $l \geq 2^{13}$, the communication cost of our protocols and baseline are close. Nonetheless, the save of computation by in-outside decomposition leads to faster execution, especially for large T in the LAN setting.

Table 4: The ratio of communication cost of $\mathcal{P}_{\text{cop-mat}}$ using in-outside decomposition vs. using out-inside decomposition, where $l = 128$ bits and the size of small-size COP $T = 256$.

Input size (n)	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
Ratio with silent OT	0.77	0.86	1	0.72	0.81	0.91
Ratio without silent OT	0.67	0.84	1	0.70	0.80	0.90

Among $\mathcal{P}_{\text{cop-mat}}$ with various T, it is obvious that the larger T is, the smaller the communication cost is and the larger the computation cost is. However, the specific execution time is affected by the network bandwidth, data size n, and the bit length l. When l is small (*e.g.*, $l = 128$ or $l = 256$), the difference of communication cost for protocols with various T is not significant, so $\mathcal{P}_{\text{cop-mat}}$ with smaller T is faster as long as the bandwidth is not extremely low as shown in Table 2 and Figure 11 due to small computation cost. When l is large, the computation cost dominates the execution time in the LAN setting, such that $\mathcal{P}_{\text{cop-mat}}$ with smaller T also performs better, as shown in Figure 11. However, determining which configuration is more efficient becomes ambiguous in the WAN setting since the relative influence of computation cost and communication latency on execution time becomes intricate to discern. But based on theoretical analysis and experimental results, we can conclude that the performance of $\mathcal{P}_{\text{cop-mat}}$ with larger T gradually becomes the best as the bandwidth decreases.

5.3.3 Comparison among Various Protocols. When the input bit length l is small, less than $2\kappa = 256$ more precisely, $\mathcal{P}_{\text{cop-net}}$ always outperforms $\mathcal{P}_{\text{cop-mat}}$ in our evaluation and theoretical comparison. For example, $\mathcal{P}_{\text{cop-mat}}$ with $T = 16$ requires nearly 2× as much OT, 2× as much communication, and 4× as much computation compared to $\mathcal{P}_{\text{cop-net}}$ when $l = 128$, leading to longer execution time regardless of the network setting and the underlying OT protocol. Note that the communication/OT complexity of $\mathcal{P}_{\text{cop-net}}$ are approximately linearly with $n \log n$, while those of the wPRF-based protocol are linearly with n. As the n increases, the communication cost of $\mathcal{P}_{\text{cop-net}}$ gradually exceeds that of the wPRF-based protocol as shown in Table 2, and their execution time gradually approaches. Although it is foreseeable that when n becomes sufficiently large, the execution of the wPRF-based protocol will be faster than $\mathcal{P}_{\text{cop-net}}$, we can still conclude that $\mathcal{P}_{\text{cop-net}}$ is the optimal choice for input with small bit length when n is not excessively large.

Table 5: Execution time (s) of protocols using silent OT for fixed input size $n = 2^{16}$ in the LAN setting.

Bit length (l)	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
$\mathcal{P}_{\text{cop-mat}}$, T=16	9.9	18.2	42.9	117.4	377.8	494.4	761.4
$\mathcal{P}_{\text{cop-net}}$	7.8	14.0	33.2	72.6	226.6	729.5	1136

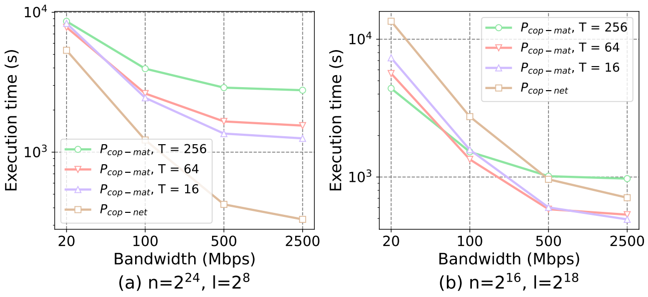


Figure 11: Execution time of protocols using silent OT under various bandwidths and unlimited latency. The left is the result on data with small bit length $l = 2^8$, while the right is the result on data with small bit length $l = 2^{18}$.

When the input bit length l is large, the ratio of communication cost between $\mathcal{P}_{\text{cop-mat}}$ and $\mathcal{P}_{\text{cop-net}}$ is nearly $\frac{2}{\log T}$, and the ratio of computation cost is nearly $\frac{4T}{3 \log T}$. Therefore, the superiority of $\mathcal{P}_{\text{cop-net}}$ and $\mathcal{P}_{\text{cop-net}}$ is closely related to network bandwidth, and $\mathcal{P}_{\text{cop-mat}}$ performs best in the WAN setting. Theoretically, in the LAN setting and with sufficiently large l, the relative speed between $\mathcal{P}_{\text{cop-mat}}$ and $\mathcal{P}_{\text{cop-net}}$ – which one is faster – should remain nearly constant as l increases, since the ratio of communication/computation cost does not vary significantly with different l. However, our experimental results deviate from this theory, as illustrated in Table 5. Even when l is large enough compared to $\kappa = 128$, $\mathcal{P}_{\text{cop-mat}}$ progressively becomes faster than $\mathcal{P}_{\text{cop-net}}$ as l increases. The issue stems from the inefficient use of bandwidth caused by the frequent IO operations and the thread scheduling of RPC (realized with Netty in our code).

We now roughly get the suggestions for selecting the optimal protocol for different cases as follows.

(1) When l is small ($\leq 2\kappa$), $\mathcal{P}_{\text{cop-net}}$ is always the best.
(2) Otherwise, as the bandwidth decreases, the optimal protocol changes from $\mathcal{P}_{\text{cop-mat}}$ with smaller T to that with larger T.

5.4 Efficiency Improvement in Applications

We implemented the SOTA shuffle-based PSU protocols to demonstrate the improvements our optimized network-based protocols bring to PSU protocols, serving as an example application of COP[9]. The baseline protocols include ZCL23-PK [46], ZCL23-PK-NO (which corresponds to the version of ZCL23-PK that does not perform point compression)[10], and two shuffle-based protocols GMR21 [20] and JSZ22-R [24], whose shuffling is instanced with the network-based [20] protocol. We remark that recently, Jia et al. [25] proposed a new

[9]We select the most representative COP applications to show the efficiency improvements. Similar efficiency gains in other downstream applications can also be achieved.
[10]We do not use ZCL23-SK as a baseline since it requires heavy-cost multiplication triples. For example, it takes 1210s to generate the triples in the LAN setting when $n = 2^{20}$ as reported in [46].

Table 6: Communication cost (in MB) and running time (in seconds) of PSU protocols with OT extension [4] for fixed input bit length $l = 64$ bits and different input size n. Communication cost of $\mathcal{S}/\mathcal{R}$ indicates the outgoing communication from $\mathcal{S}/\mathcal{R}$ to the other party. The best protocol within a setting is marked in green.

n	Protocol	Communication cost (MB)					Running time (s)											
		$\mathcal{R}$		$\mathcal{S}$		total	LAN (2.5Gbps)						WAN (100Mbps)					
							Thread number = 1			Thread number = 15			Thread number = 1			Thread number = 15		
		offline	online	offline	online		offline	online	total	offline	online	total	offline	online	total	offline	online	total
2^{16}	ZCL23-PKC	0	6.49	0	4.62	11.11	1.47	16.05	17.52	1.48	4.59	6.07	1.66	17.63	19.29	1.64	8.02	9.66
	ZCL23-PKC-NO	0	11.82	0	8.62	20.44	1.5	12.86	14.36	1.44	3.12	4.56	1.64	16.72	18.36	1.64	7.24	8.88
	GMR21	0.01	33.28	0.01	25.2	58.52	0.15	5.23	5.38	0.04	2.59	2.63	1.28	15.15	16.43	1.06	12.56	13.62
	GMR21-OO	19.82	13.48	19.82	5.4	58.52	1.38	3.33	4.71	1.25	1.3	2.55	8.03	8.27	16.30	7.9	5.69	13.59
	GMR21-OO*	9.22		18.42		46.52	0.84	3.41	4.25	0.48	0.98	1.46	8.76	7.5	16.26	8.15	3.23	11.38
	JSZ22-R	0.01	25.45	0.01	21.99	47.46	0.15	1.53	1.68	0.04	1.04	1.08	0.83	8.24	9.07	0.7	10.4	11.1
	JSZ22-R-OO	19.81	5.65	19.81	2.19	47.46	1.39	0.42	1.81	0.76	0.35	1.11	7.43	2.55	9.98	6.82	2.44	9.26
	JSZ22-R-OO*	9.21		18.41		35.46	0.73	0.41	1.14	0.5	0.2	0.7	8.14	1.81	9.95	7.62	1.55	9.17
2^{18}	ZCL23-PKC	0	26.13	0	18.5	44.63	1.45	56.34	57.79	1.51	15.71	17.22	1.71	61.14	62.85	1.6	26.83	28.43
	ZCL23-PKC-NO	0	47.59	0	34.5	82.09	1.47	50.37	51.84	1.44	10.02	11.46	1.68	57.99	59.67	1.64	17.23	18.87
	GMR21	0.01	142.11	0.01	110.7	252.8	0.13	20.37	20.5	0.02	7.84	7.86	1.19	44.17	45.36	1.05	32.77	33.82
	GMR21-OO	88.82	53.31	88.82	21.9	252.8	5.65	14.61	20.26	3.91	4.19	8.10	22.38	24.16	46.54	21.11	12.99	34.1
	GMR21-OO*	41.62		83.22		200.0	2.98	14.79	17.77	2.01	3.68	5.69	19.32	22.77	42.09	17.29	11.93	29.22
	JSZ22-R	0.01	111.7	0.01	98.3	210.0	0.15	7.08	7.23	0.01	4.15	4.16	0.85	25.56	26.41	0.7	24.16	24.86
	JSZ22-R-OO	88.81	22.9	88.81	9.5	210.0	5.46	1.71	7.17	3.17	0.96	4.13	21.31	6.53	27.84	19.78	5.55	25.33
	JSZ22-R-OO*	41.61		83.21		157.2	2.9	1.79	4.69	1.96	0.98	2.94	16.95	6.47	23.42	15.95	5.24	21.19
2^{20}	ZCL23-PKC	0	104.7	0	74.0	178.7	1.44	222.4	223.9	1.49	62.04	63.53	1.67	239.2	240.9	1.65	74.64	76.29
	ZCL23-PKC-NO	0.0	190.8	0.0	138.0	328.8	1.46	206.4	207.8	1.48	40.63	42.11	1.67	229.8	231.4	1.63	60.82	62.45
	GMR21	0.01	603.9	0.01	482.4	1086	0.29	88.52	88.81	0.03	38.98	39.01	1.24	173.3	174.5	1.05	128.6	129.6
	GMR21-OO	393.6	210.3	393.6	88.8	1086	29.25	63.31	92.56	19.78	17.53	37.31	88.71	84.21	172.9	84.34	42.32	126.7
	GMR21-OO*	185.6		371.2		855.9	13.29	63.59	76.88	8.76	16.88	25.64	55.7	83.86	139.6	54.69	42.74	97.43
	JSZ22-R	0.01	485.2	0.01	431.6	916.8	0.21	34.31	34.52	0.05	22.64	22.69	0.9	102.8	103.7	0.71	97.07	97.78
	JSZ22-R-OO	393.6	91.6	393.6	38.0	916.8	29.42	8.59	38.01	19.29	4.14	23.43	87.65	21.95	109.6	82.19	18.47	100.7
	JSZ22-R-OO*	185.6		371.2		686.4	14.0	8.41	22.41	8.68	3.68	12.36	54.71	21.02	75.73	53.68	17.1	70.78
2^{22}	ZCL23-PKC	0	419.7	0	296.0	715.7	1.44	951.7	953.1	1.47	260.6	262.1	1.62	973.6	975.2	1.65	299.2	300.9
	ZCL23-PKC-NO	0	764.5	0	552.0	1316	1.48	859.3	860.8	1.46	172.3	173.8	1.67	923.9	925.6	1.64	239.1	240.7
	GMR21	0.01	2564	0.01	2088	4652	0.18	448.9	449.1	0.02	198.2	198.2	1.29	760.0	761.3	1.04	553.5	554.5
	GMR21-OO	1728	836.2	1728	360.0	4652	186.8	287.6	474.4	110.4	86.31	196.7	409.2	344.1	753.3	380.8	164.86	545.7
	GMR21-OO*	819.2		1638		3654	97.06	288.6	385.6	50.42	84.8	135.2	240.5	345.7	586.2	237.6	163.1	400.7
	JSZ22-R	0.01	2099	0.01	1892	3991	0.18	207.83	208.0	0.03	124.6	124.6	0.86	490.2	491.1	0.7	432.2	432.9
	JSZ22-R-OO	1728	371.2	1728	164.0	3991	182.6	45.53	228.1	107.9	19.32	127.3	409.7	86.61	496.3	378.0	65.06	443.0
	JSZ22-R-OO*	819.2		1638		2993	95.55	45.77	141.3	48.63	21.04	69.67	240.01	85.74	325.8	237.4	65.09	302.5

PSU protocol that eliminates unnecessary *during-execution leakage*, that is, ensuring $\mathcal{S}$ has the information after execution rather than during execution. Their construction achieves enhanced security with AHE with additional costs. Here, we focus on efficient PSU constructions under the standard PSU functionality.

We employ COP to transform two shuffle-based PSU protocols into the offline-online PSU. The offline phase generates COP and performs necessary operations before the actual protocol execution, e.g., key distribution. The online phase executes subsequent protocols. The result protocols are denoted as GMR21-OO, GMR21-OO*, JSZ22-R-OO and JSZ22-R-OO*, where the COP generation in GMR21-OO and JSZ22-R-OO still use the network-based protocol [20] and that in GMR21-OO* and JSZ22-R-OO* use our $\mathcal{P}_{\text{cop-net}}$. Since the permutation in the shuffle-based PSU protocols is randomly sampled, the receiver can directly use the random permutation of the offline-generated COP as the permutation randomly sampled during the online phase, making the usage of COP correspond to the second case as mentioned in §2.4. Meanwhile, since the cost of network-based COP generation involves less communication than directly invoking oblivious permutation by a factor of nl, the total cost of offline generating COP and performing online permutation with it is the same as directly executing oblivious permutation on input.

Note that except for operations such as parameter negotiation that have minimal cost, most of the cost in the offline phase comes from COP generation. According to the offline/online cost shown in Table 6, the cost of oblivious permutation accounts for a relatively large proportion of the shuffle-based PSU protocols, e.g., oblivious permutation taking 67.7% − 74.3% and 83.5% − 86.8% of the communication cost in the original GMR21 and JSZ22-R protocols in our experiments. Therefore, with an offline-online paradigm, shuffle-based protocols outperform the baseline in terms of both online communication cost and online execution time. JSZ22-OO (or JSZ22-OO*) [24] have the lowest online cost. Further, our network-based COP generation protocol $\mathcal{P}_{\text{cop-net}}$ can greatly reduce the cost of the offline phase of the shuffle-based PSU protocols, making their total execution time lead in all settings except WAN with multi-thread.

6 Extension and malicious security of COP

COP Extension. To generate a COP for data with a larger bit length, we can first generate a COP for input with a smaller bit length and subsequently map the elements of the output vectors into the larger target domain. Two approaches have been proposed for domain extension, namely the PCG-based protocol in [36] and the LWR-based protocol in [30]. The PCG-based protocol in [36] is based on function secret sharing (FSS) [11] and the LPN assumption [10]. The parties first generate a public bit matrix such that syndrome decoding is hard. The extension step is done by using FSS to produce a secret-shared sparse bit vector with the initial COP as input and then multiplying the public bit matrix and those sparse bit vectors. The LWR-based protocol in [30] is designed for applications with a certain degree of error tolerance, which introduces one bit of error to COP, such that $\mathbf{c} = \pi(\mathbf{a}) - \mathbf{b} + \epsilon$, where $\epsilon \in \{0,1\}^n$.

Our optimizations are generic and fundamental, enabling fast COP generation with smaller bit lengths. Then, the result can be extended to the required domain with a larger bit length using the above-mentioned scheme, thereby accelerating the overall process.

Maliciously secure permutation. There have been many efforts to construct malicious secure two-party permutation protocol [18, 40, 43]. The protocol in [40] is built on the matrix-based approach [13], where the malicious security is guaranteed by malicious secure PPRF and checks on the matrix constructed by multiple PPRF results. The protocol in [43] adopts cut-and-choose to check each switch in the network-based approach is correctly performed.

7 Conclusion

We propose optimizations for two types of solutions for generating Correlated Oblivious Permutation (COP). We implement all the protocols for fair comparison and conclude strategies for selecting the optimal protocol under different input and network environments through theoretical analysis and experiment. Looking ahead, we aim to explore efficient malicious secure COP construction and domain extension approaches for COP.

Acknowledgment

Weiran Liu is the corresponding author of this paper. This work is supported by the National Key Research and Development Program of China (Grant No. 2022YFB3102500), and the Major Programs of the National Social Science Foundation of China (Grant No. 22&ZD147). The authors used ChatGPT4 to revise the text in §1, §2, and §4 to correct any typos and grammatical errors.

References

[1] Amit Agarwal, Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Mahimna Kelkar, and Yiping Ma. 2024. Secure sorting and selection via function secret sharing. In *2024 ACM SIGSAC Conference on Computer and Communications Security (CCS 2024)*. 3023–3037.

[2] Navid Alamati, Guru-Vamsi Policharla, Srinivasan Raghuraman, and Peter Rindal. 2024. Improved alternating-moduli PRFs and post-quantum signatures. In *Annual International Cryptology Conference (CRYPTO 2024)*. Springer, 274–308.

[3] Erik Anderson, Melissa Chase, F Betül Durak, Kim Laine, and Chenkai Weng. 2024. Precio: Private aggregate measurement via oblivious shuffling. In *2024 ACM SIGSAC Conference on Computer and Communications Security (CCS 2024)*. 1819–1833.

[4] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2013. More efficient oblivious transfer and extensions for faster secure computation. In *2013 ACM SIGSAC Conference on Computer and Communications Security (CCS 2013)*. 535–548.

[5] Bruno Beauquier and Éric Darrot. 2002. On arbitrary size Waksman networks and their vulnerability. *Parallel Processing Letters* 12, 03n04 (2002), 287–296.

[6] Donald Beaver. 1991. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology (CRYPTO 1991)*. Springer, 420–432.

[7] Václad E Beneš. 1964. Optimal rearrangeable multistage connecting networks. *Bell System Technical Journal* 43, 4 (1964), 1641–1656.

[8] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J Wu. 2018. Exploring crypto dark matter: New simple PRF candidates and their applications. In *Theory of Cryptography Conference (TCC 2018)*. Springer, 699–729.

[9] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. 2022. Correlated pseudorandomness from expand-accumulate codes. In *Annual International Cryptology Conference (CRYPTO 2022)*. Springer, 603–633.

[10] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. 2019. Efficient two-round OT extension and silent non-interactive secure computation. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS 2019)*. 291–308.

[11] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques (EUROCRYPT 2015)*. Springer, 337–367.

[12] Chihming Chang and Rami Melhem. 1997. Arbitrary size Benes networks. *Parallel Processing Letters* 7, 03 (1997), 279–284.

[13] Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. 2020. Secret-shared shuffle. In *26th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2020)*. Springer, 342–372.

[14] Tung Chou and Claudio Orlandi. 2015. The simplest protocol for oblivious transfer. In *4th International Conference on Cryptology and Information Security in Latin America (LATINCRYPT 2015)*. Springer, 40–58.

[15] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. 2021. Silver: silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *Annual International Cryptology Conference (CRYPTO 2021)*. Springer, 502–534.

[16] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation.. In *22th Annual Network and Distributed System Security Symposium (NDSS 2015)*.

[17] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. 2021. MPC-friendly symmetric cryptography from alternating moduli: candidates, protocols, and applications. In *41st Annual International Cryptology Conference (CRYPTO 2021)*. Springer, 517–547.

[18] Saba Eskandarian and Dan Boneh. 2021. Clarion: Anonymous communication from multiparty shuffling protocols. *Cryptology ePrint Archive* (2021).

[19] Jiacheng Gao, Yuan Zhang, and Sheng Zhong. 2024. Multiparty Shuffle: Linear Online Phase is Almost for Free. *Cryptology ePrint Archive* (2024).

[20] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. 2021. Private set operations from oblivious switching. In *IACR International Conference on Public-Key Cryptography (PKC 2021)*. Springer, 591–617.

[21] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1986. How to construct random functions. *J. ACM* 33, 4 (1986), 792–807.

[22] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. 2023. Half-tree: Halving the cost of tree expansion in cot and dpf. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2024)*. Springer, 330–362.

[23] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending oblivious transfers efficiently. In *23rd Annual International Cryptology Conference (CRYPTO 2023)*. Springer, 145–161.

[24] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. 2022. Shuffle-based private set union: Faster and more secure. In *31st USENIX Security Symposium (USENIX Security 2022)*. 2947–2964.

[25] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, and Dawu Gu. 2024. Scalable private set union, with stronger security. In *33rd USENIX Security Symposium (USENIX Security 2024)*.

[26] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A low latency framework for secure neural network inference. In *27th USENIX security symposium (USENIX Security 2018)*. 1651–1669.

[27] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2015. Actively secure OT extension with optimal overhead. In *35th Annual Cryptology Conference (CRYPTO 2015)*. Springer, 724–741.

[28] Jiseung Kim, Hyung Tae Lee, and Yongha Son. 2024. Revisiting shuffle-based private set unions with reduced communication. *Cryptology ePrint Archive* (2024).

[29] Yehuda Lindell. 2017. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich* (2017), 277–346.

[30] Yang Liu, Bingsheng Zhang, Yuxiang Ma, Zhuo Ma, and Zecheng Wu. 2023. iPrivJoin: An ID-private data join framework for privacy-preserving machine learning. *IEEE Transactions on Information Forensics and Security (TIFS 2023)* 18 (2023), 4300–4312.

[31] Qiyao Luo, Yilei Wang, Zhenghang Ren, Ke Yi, Kai Chen, and Xiao Wang. 2021. Secure machine learning over relational data. *arXiv preprint arXiv:2109.14806* (2021).

[32] Daniel Mansy and Peter Rindal. 2019. Endemic oblivious transfer. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS 2019)*. 309–326.

[33] Payman Mohassel, Peter Rindal, and Mike Rosulek. 2020. Fast database joins and PSI for secret shared data. In *2020 ACM SIGSAC Conference on Computer and Communications Security (CCS 2020)*. 1271–1287.

[34] Payman Mohassel and Saeed Sadeghian. 2013. How to hide circuits in MPC an efficient framework for private function evaluation. In *32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2013)*. Springer, 557–574.

[35] Moni Naor and Benny Pinkas. 2001. Efficient oblivious transfer protocols. In *12th Annual Symposium on Discrete Algorithms (SODA 2021)*, Vol. 1. 448–457.

[36] Stanislav Peceny, Srinivasan Raghuraman, Peter Rindal, and Harshal Shah. 2024. Efficient permutation correlations and batched random access for two-party computation. *Cryptology ePrint Archive* (2024).

[37] Xinyu Peng, Feng Han, Li Peng, Weiran Liu, Zheng Yan, Kai Kang, Xinyuan Zhang, Guoxing Wei, Jianling Sun, and Jinfei Liu. 2024. MapComp: A secure view-based collaborative analytics framework for join-group-aggregation. *arXiv preprint arXiv:2408.01246* (2024).

[38] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. 2023. Expand-convolute codes for pseudorandom correlation generators from LPN. In *Annual International Cryptology Conference (CRYPTO 2023)*. Springer, 602–632.

[39] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. 2019. Distributed vector-OLE: Improved constructions and implementation. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS 2019)*. 1055–1072.

[40] Xiangfu Song, Dong Yin, Jianli Bai, Changyu Dong, and Ee-Chien Chang. 2023. Secret-shared shuffle with malicious security. *Cryptology ePrint Archive* (2023).

[41] Abraham Waksman. 1968. A permutation network. *Journal of the ACM (JACM)* 15, 1 (1968), 159–163.

[42] Yilei Wang and Ke Yi. 2021. Secure yannakakis: Join-aggregate queries over private data. (2021), 1969–1981.

[43] Qiuhong Anna Wei. 2024. Malicious secure oblivious shuffling from Beneš network. (2024).

[44] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. 2020. Ferret: Fast extension for correlated OT with small communication. In *2020 ACM SIGSAC Conference on Computer and Communications Security (CCS 2020)*. 1607–1626.

[45] Andrew C Yao. 1982. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 160–164.

[46] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. 2023. Linear private set union from multi-query reverse private membership test. In *32nd USENIX Security Symposium (USENIX Security 2023)*. 337–354.

Appendix

A AHE-based and wPRF-based Protocols

In this section, we briefly review the construction of the AHE-based and wPRF-based oblivious permutation protocols. Notably, these protocols have the same communication/computation complexity of $O(nl)$, where n is the data size and l is the bit length of each element. We highlight a commonality between the two protocols that the permutation process is done by the receiver solely on "somewhat public" data.

AHE-based protocol. At the beginning of the protocol, two parties agree on the rerandomizable additive homomorphic encryption algorithm, e.g., ElGamal encryption. W.L.O.G., We denote the encryption and decryption algorithms as E and D, the result of encryption and decryption of data $x \in \mathbb{D}$ with the private key sk as $E_{sk}(x)$ and $D_{sk}(x)$, respectively.

(1) The sender generates the private key sk and the public key pk. The sender uses that key to encrypt each element of his vector $\mathbf{a} \in \mathbb{D}^n$ to obtain $\{E_{sk}(\mathbf{a}_i)\}_{i\in[n]}$. Then, the sender sends the public key pk and the encrypted result $\{E_{sk}(\mathbf{a}_i)\}_{i\in[n]}$ to the receiver.

(2) The receiver permutes the received encrypted data and samples a random vector $\mathbf{c} \in \mathbb{D}^n$ to perform the element-wise homomorphic addition, and obtains $\{E_{sk}(\mathbf{a}_{\pi(i)} - \mathbf{c}_i)\}_{i\in[n]}$. Then, the receiver sends $\{E_{sk}(\mathbf{a}_{\pi(i)} - \mathbf{c}_i)\}_{i\in[n]}$ to the receiver.

(3) The sender decrypts the received data with his private key to obtain $\mathbf{b}_i = D_{sk}(E_{sk}(\mathbf{a}_\pi - \mathbf{c}_i)) = \mathbf{a}_{\pi(i)} - \mathbf{c}_i$ for $i \in [n]$.

(4) The sender outputs $\mathbf{a}, \mathbf{b}$. The receiver outputs $\mathbf{c}$.

In the above protocol, the encrypted result $\{E_{sk}(\mathbf{a}_i)\}_{i\in[n]}$ can be seen as "public" data. Although the receiver does not know the corresponding plaintext, he has the ability to permute them and add masks to them.

Since the AHE-based solution is computationally inefficient, we do not compare our protocols with it in our experiment.

wPRF-based protocol. Before we go into the details, we first present the definition of a Weak Pseudorandom Function.

Definition 3 (Weak Pseudorandom Function). *A function $F_{wp} : \mathbb{K} \times \mathbb{D} \to \mathbb{Y}$ with key space $\mathbb{K}$, domain $\mathbb{X}$ and output space $\mathbb{Y}$ is said to be a Weak Pseudorandom Function (wPRF) if for $k \leftarrow \mathbb{K}$, $x_i \leftarrow \mathbb{D}$, $y_i \leftarrow \mathbb{Y}$, $\{(x_i, F_{wp}(k, x_i))\}_{i\in[q]} \overset{c}{\equiv} \{(x_i, y_i)\}_{i\in[q]}$ holds.*

The wPRF-based protocol relies on the shared oblivious weak Pseudorandom function $\mathcal{F}_{\mathsf{swPRF}}$, which takes data $x \in \mathbb{D}$ from the receiver and a private key $k \in \mathbb{K}$ from the sender and returns the share of $F_{wp}(k, x)$ to parties. Recently, the alternating moduli paradigms [2, 8, 17] are proposed and they can be used to efficiently construct various symmetric key primitives, including wPRF. The common characteristic of those paradigms is that the inputs are multiplied by two linear maps over different moduli $\mathbb{F}_2$ and $\mathbb{F}_3$. Since the computation over the moduli $\mathbb{F}_2$ and $\mathbb{F}_3$ can be securely evaluated with OT, the alternating moduli paradigms can be used to efficiently construct the $\mathcal{F}_{\mathsf{swPRF}}$. In our implementation, we adopt the approach proposed in [2] to instantiate $\mathcal{F}_{\mathsf{swPRF}}$.

The wPRF-based protocol proceeds as follows:

(1) The receiver samples $t \leftarrow \mathbb{F}_{2^\kappa}$ and sends t to the sender. Two parties locally compute $\mathbf{x} = \{H(t,i)\}_{i\in[n]}$.
(2) The sender samples $k \leftarrow \mathbb{K}$, and computes the vector $\mathbf{a}$, where $\mathbf{a}_i = F_{wp}(k, x_i)$.
(3) Two parties invoke $\mathcal{F}_{\mathsf{swPRF}}$, where the receiver inputs the private key k and the sender inputs $\{\mathbf{x}_{\pi(i)}\}_{i\in[n]}$. As the result, the sender learns the vector $\mathbf{b}$ and the receiver learns the vector $\mathbf{c}$, where $\mathbf{b}_i \oplus \mathbf{c}_i = F_{wp}(k, x_{\pi(i)})$ holds for $i \in [n]$.
(4) The sender outputs $\mathbf{a}, \mathbf{b}$. The receiver outputs $\mathbf{c}$.

If we analogize wPRF to an encryption algorithm, in a sense, the wPRF-based solution [36] can be considered as a "reverse" version of the AHE-based solution. The reason is that the "public" data here is the plaintext vector $\mathbf{x} = \{H(t,i)\}_{i\in[n]}$ (in contrast to the encrypted vector $\{E_{sk}(\mathbf{a}_i)\}_{i\in[n]}$ in AHE-based protocol), while the output vectors $\mathbf{b}, \mathbf{c}$ are the shares of the PRF result of this public vector (in contrast to the decryption result of the public vector in AHE-based protocol).

B Programming the Waksman Network

Since the basic idea of network-based COP is to invoke one OT for each switch in the permutation network [20, 34], using an improved permutation network with fewer switches results in fewer OT invocations, thus improving the efficiency. Waksman network [41] is an optimized network that removes one redundant switch at each step of the recursive construction of Beneš networks when $n = 2^r$. Consequently, the total number of switches is reduced to $n\log_2 n - n + 1$ (for $n = 2^r$), making it *practically* optimal.

It is desirable to use the Waksman network for permutations of arbitrary size n. Beauquier et.al. [5] generalized Waksman network to handle arbitrary size n with $W(n)$ switches, where $W(2) = 1$, $W(3) = 3$ and $W(n) = W(\lceil \frac{n}{2} \rceil) + W(\lfloor \frac{n}{2} \rfloor) + n - 1$ for $n \geq 4$. Therefore, the Waksman network enables us to obtain permutation networks with *strictly* fewer switches than the Beneš network when $n \geq 4$, thereby reducing the costs of network-based COP. The construction of the Waksman network is shown in Figure 12.

The principle for programming the switches is to ensure that for $x \in \lceil \frac{n}{2} \rceil$, two input wires with indexes x and $x + \lfloor \frac{n}{2} \rfloor$ are permuted with a switch to become the x-th inputs of two subnetworks, respectively, while maintaining the constraint that input wires with indexes $\pi(x), \pi(x + \lfloor \frac{n}{2} \rfloor)$ are routed to two different subnetworks. The key to reducing the switches is to deterministically route the $\pi(n)$-th input wire to the second subnetwork. This allows the $\frac{n}{2}$-th output wires of two subnetworks can be directly connected to the $\frac{n}{2}$-th and the n-th output wires when n is even. Compared to the Beneš network, an example of which is shown in Figure 4, the difference is that the switch at the bottom right corner of each subnetwork whose size is even and greater than or equal to 4 is removed.

Here we formally describe the algorithm. For simplicity, we further define the correlated values for $x \in [n]$: we denote $x^{\triangleright n}$ as $x + n^{\triangledown}$ if $x \leq n^{\triangledown}$, and $x - n^{\triangledown}$ otherwise, representing x's neighbor value under the domain size n. We denote $x^{\vdash n}$ as $min(x, x^{\triangleright n})$. For some $n \in \mathbb{N}$, we denote $n^{\vartriangle}$ as $\lceil \frac{n}{2} \rceil$, $n^{\triangledown}$ as $\lfloor \frac{n}{2} \rfloor$. Given a permutation network, we denote the input wires of the switches in the first layer as the initial wires and the output wires of the switches in the last layer as the result wires. Alg. 1 generates an in-place

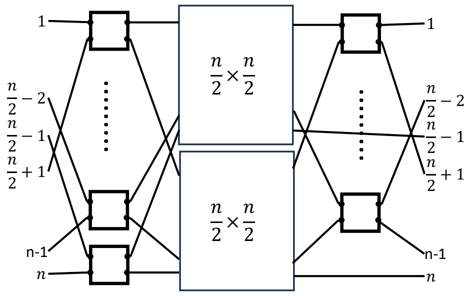
Algorithm 1 Waksman network $\omega \leftarrow \Omega(\pi)$

Input: A permutation $\pi \in \mathbb{P}_n$.
Output: The programmed Waksman network ω.

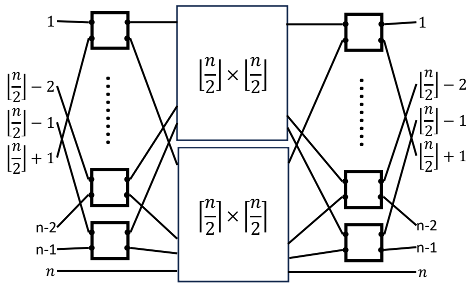
```
1:  if $n \leq 3$ then
2:      Return a network $\omega$ by looking up the table.
3:  end if
4:  Initialize two size-$n$ vectors of wires $I, O$.
5:  Initialize $\mathbf{f} = \{\perp\}^n$, a stack $T = \emptyset$, a set $P = [2n^{\triangledown}] \backslash \{\pi(n)\}$, and
    $\pi_0 = \{\perp\}^{n^{\triangledown}}$, $\pi_1 = \{\perp\}^{n^{\vartriangle}}$.
6:  if $n$ is odd then
7:      Set $\mathbf{f}[n] = 1$, $\pi_1(\pi^{-1}(n)^{\vdash n}) = n^{\vartriangle}$.
8:      If $\pi(n) \neq n$, push $(\pi(n), 1)$ into $S$.
9:  else
10:     Push $(\pi(n), 1)$ into $S$.
11: end if
12: while $P \neq \emptyset$ or $T \neq \emptyset$ do
13:     if $T = \emptyset$ then
14:         Remove an element $x$ from $P$, let $c = 0$.
15:     else
16:         Pop $(x, c)$ from $T$.
17:     end if
18:     Let $y = \pi^{-1}(x)$, set $\mathbf{f}[x] = c$, $\pi_c(y^{\vdash n}) = x^{\vdash n}$.
19:     If $x^{\triangleright n} \in P$, remove $x^{\triangleright n}$ from $P$ and push $(x^{\triangleright n}, \bar{c})$ into $T$.
20:     If $\pi(y^{\triangleright n}) \in P$, remove $\pi(y^{\triangleright n})$ from $P$ and push $(\pi(y^{\triangleright n}), \bar{c})$
        into $T$.
21: end while
22: Recursion: Invoke $\omega_0 \leftarrow \Omega(\pi_0)$, $\omega_1 \leftarrow \Omega(\pi_1)$.
23: Let $I^i, O^i$ be the vectors of initial wires and result wires for $\omega_i$
    respectively for $i \in \{0, 1\}$.
24: for $x \in [n^{\triangledown}]$ do
25:     Add a switch with choice bit $\mathbf{f}[x]$, input wires $I_x, I_{x+n^{\triangledown}}$ and
        output wires $I^0_x, I^1_x$;
26:     Add a switch with choice bit $\mathbf{f}[\pi(x)]$, input wires $O^0_x, O^1_x$
        and output wires $O_x, O_{x+n^{\triangledown}}$.
27: end for
28: if $n$ is even then
29:     Directly link $O^0_{n^{\triangledown}}$ and $O_{n^{\triangledown}}$, $O^1_{n^{\triangledown}}$ and $O_n$.
30: else
31:     Directly link $I_n$ and $I^1_{n^{\vartriangle}}$, $O^1_{n^{\vartriangle}}$ and $O_n$.
32: end if
33: return $\omega$ with initial wires $I$, result wires $O$.
```

Waksman network in the recursive style using $W(n)$ switches for size-n input, where $W(n)$ satisfies $W(2) = 1, W(3) = 3$ and $W(n) = W(n^{\vartriangle}) + W(n^{\triangledown}) + n - 1$ for $n > 3$. Let a length-n vector $\mathbf{f}$ and two permutations π_0, π_1, whose size are $n^{\triangledown}, n^{\vartriangle}$ respectively, be the intermediate variables of the above programming. $\mathbf{f}$ indicates which sub-network should the input be switched to, and π_0, π_1 are the target permutations of ω_0, ω_1. The x-th input will be switched into the $x^{\vdash n}$-th input of $\omega_{\mathbf{f}[x]}$ and the x-th output is the $x^{\vdash n}$-th output of $\omega_{\mathbf{f}[\pi(x)]}$.

If n is even, the algorithm first deterministically program $\mathbf{f}[\pi(n)] = 1$ representing switch $I_{\pi(n)}$ to ω_1, and the switch for $O_{n^{\triangledown}}$ and O_n is removed finally. If n is odd, besides from the $I_{\pi(n)}$ is assigned to ω_1, I_n is also directly link $n^{\vartriangle}$-th initial wire of ω_1. Given a wire I_i has



(a) n is even


1
$\lceil\frac{n}{2}\rceil-2$
$\lceil\frac{n}{2}\rceil-1$
$\lceil\frac{n}{2}\rceil+1$
n-2
n-1
n
$\lfloor\frac{n}{2}\rfloor\times\lfloor\frac{n}{2}\rfloor$
$\lceil\frac{n}{2}\rceil\times\lceil\frac{n}{2}\rceil$
1
$\lfloor\frac{n}{2}\rfloor-2$
$\lfloor\frac{n}{2}\rfloor-1$
$\lfloor\frac{n}{2}\rfloor+1$
n-2
n-1
n

(b) n is odd

Figure 12: Construction of the Waksman network.

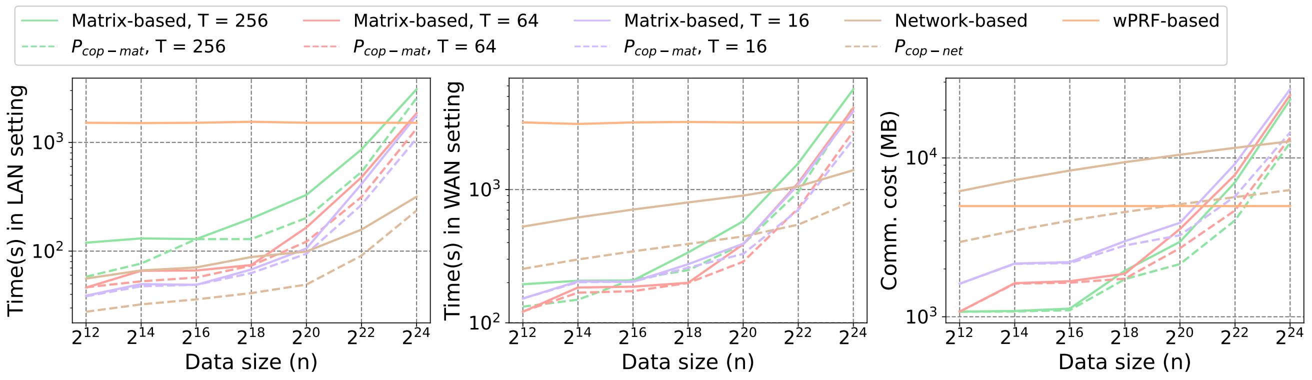


Figure 13: Performance using silent OT [44] when the total amount of input data is fixed to $nl = 2^{31}$ (256MB data in total).

been assigned to a specific sub-network ω_c, its neighbor input wire $I_{i^{\triangleright n}}$ of the same switch would be assigned to $\omega_{\bar{c}}$. Meanwhile, let O_y be the the result wire that I_i should finally link where $y = \pi^{-1}(i)$, its neighbor output wire $O_{y^{\triangleright n}}$ should be linked to a wire in $\omega_{\bar{c}}$, which means that $I_{\pi(y^{\triangleright n})}$ should be assigned to $\omega_{\bar{c}}$. Let P store all unassigned wires, and stack T store wires that have just been assigned. The algorithm first traverses the unassigned wires correlated to the wires in T until a closed loop is formed and repeats the process with an unassigned wire until all wires are assigned.

C Proof of Theorem 1

Proof. We construct simulator Sim_0 for $\mathcal{S}$ as follows.

(1) Sim_0 invokes $\mathcal{F}_{\mathsf{copv}}$ and receives $\tilde{\mathbf{a}}, \tilde{\mathbf{b}}$.
(2) Sim_0 samples $\tilde{\Delta}$ and sends $(\mathsf{init}, \tilde{\Delta})$ to $\mathcal{F}_{\mathsf{spCOT}}$.
(3) Sim_0 random samples $n-1$ length-n vectors $\{\tilde{\mathbf{V}}_i\}_{i\in[n-1]}$ where $\tilde{\mathbf{V}}_i \xleftarrow{\$} \mathbb{F}^n_{2^\kappa}$. Sim_0 invokes the simulator of $\mathcal{F}_{\mathsf{spCOT}}$ with $\{\tilde{\mathbf{V}}_i\}_{i\in[n-1]}$ and append the output to the view.
(4) Sim_0 computes two length-n vectors $\tilde{\mathbf{a}}', \tilde{\mathbf{b}}'$ with the same computation in step 3-4.
(5) Sim_0 computes $\tilde{\mathbf{x}} = \tilde{\mathbf{a}} + \tilde{\mathbf{a}}'$, $\tilde{\mathbf{y}} = \tilde{\mathbf{b}} + \tilde{\mathbf{b}}'$.

The correctness of Sim_0 directly follows the security of the underlying protocol realizing $\mathcal{F}_{\mathsf{spCOT}}$. We construct simulator Sim_1 for $\mathcal{R}$ as follows.

(1) Sim_1 sends (init) to $\mathcal{F}_{\mathsf{spCOT}}$.
(2) Sim_1 receives $\mathcal{R}$'s input π and random samples $n-1$ length-n vectors $\{\tilde{\mathbf{W}}_i\}_{i\in[n-1]}$ where $\tilde{\mathbf{W}}_i \xleftarrow{\$} \mathbb{F}^n_{2^\kappa}$. Sim_1 invokes the simulator of $\mathcal{F}_{\mathsf{spCOT}}$ with $\{(\pi(i), \tilde{\mathbf{W}}_i)\}_{i\in[n-1]}$ and append the output to the view.
(3) Sim_1 computes length-n vector $\tilde{\mathbf{c}}'$ with the same computation in step 5-6.
(4) Sim_1 invokes $\mathcal{F}_{\mathsf{copv}}$ with input π and receives $\tilde{\mathbf{c}}$.
(5) Sim_1 samples $\tilde{\mathbf{x}} \xleftarrow{\$} \mathbb{F}^n_{2^\kappa}$ and computes $\tilde{\mathbf{y}} = \tilde{\mathbf{c}}' + \pi(\tilde{\mathbf{x}}) - \tilde{\mathbf{c}}$.

First, if the computed result of $\{H(\mathbf{V}_{i,\pi(i)})\}_{i\in[n]}$ in step 4 is replaced by random sampled $\mathbf{r} \leftarrow \mathbb{F}^n_{2^\kappa}$, the result view of an execution view^1 is identical to $\mathsf{view}^{\mathcal{P}}_1$ due the correlation robustness property of H. Since $\{H(\mathbf{V}_{i,\pi(i)})\}_{i\in[n]} = \{H(\mathbf{W}_{i,\pi(i)} \oplus \Delta)\}_{i\in[n]}$ are pseudorandom without knowing Δ. We next analyze $\{\mathsf{view}^1, \mathsf{output}^{\mathcal{P}}(\perp, \pi)\}$ and $\{\mathsf{Sim}_1(\pi, \tilde{\mathbf{c}}), \mathcal{F}_{\mathsf{copv}}(\perp, \pi)\}$. It is obvious that $\{\pi, \mathbf{W}, \mathbf{c}'\}$ and $\{\pi, \tilde{\mathbf{W}}, \tilde{\mathbf{c}'}\}$ are indistinguishable due to the security of the underlying protocol of $\mathcal{F}_{\mathsf{spCOT}}$. Also, considering the randomness of $(\mathbf{x}, \mathbf{y})$ and $(\tilde{\mathbf{x}}, \tilde{\mathbf{c}})$, it is obvious that $\{\mathbf{x}, \mathbf{y}, \mathbf{c}', \mathbf{c}, \pi\} \overset{c}{\equiv} \{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{c}'}, \tilde{\mathbf{c}}, \pi\}$ due to their correlation. Therefore, Sim_1 is indistinguishable from the real one, and this completes the proof. □

D Additional experimental result

We further evaluate protocols with the fixed total amount nl of input data, the result is shown in Figure 13.