Efficient 2PC for Constant Round Secure Equality Testing and Comparison

Tianpei Lu[1]†, Xin Kang[2]†, Bingsheng Zhang[13]§, Zhuo Ma[2]§, Xiaoyuan Zhang[1], Yang Liu[2], Kui Ren[1] and Chun Chen[1]
[1]The State Key Laboratory of Blockchain and Data Security, Zhejiang University, {lutianpei, bingsheng, zhangxiaoyuan,kuiren,chenc}@zju.edu.cn,
[2]Xidian University, kangxin@stu.xidian.edu.cn, mazhuo@mail.xidian.edu.cn, bcds2018@foxmail.com,
[3]Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security

Abstract

Secure equality testing and comparison are two important primitives widely used in many secure computation scenarios, such as privacy-preserving machine learning, private set intersection, and secure data mining, etc. This work proposes new constant-round two-party computation (2PC) protocols for secure equality testing and comparison. Our protocols are designed in the online/offline paradigm. For 32-bit inputs, the online communication cost of our equality testing protocol and secure comparison protocol are as low as 76 bits (1% of ABY) and 384 bits (5% of ABY) , respectively. Our benchmarks show that (i) for 32-bit equality testing, our scheme performs 9× faster than the Guo *et al.* (EUROCRYPT 2023) and 15× of the garbled circuit (GC) with the half-gate optimization (CRYPTO 2015). (ii) for 32-bit secure comparison, our scheme performs 3× faster than Guo *et al.* (EUROCRYPT 2023), 6× faster than both Rathee *et al.* (CCS 2020) and GC with the half-gate optimization.

1 Introduction

Secure multiparty computation (MPC) [7, 26, 53] enables several untrusted parties to perform joint computations without revealing their private inputs. In the early stages, general-purpose MPC protocols [26, 31, 53] were widely studied and significantly improved in terms of performance. Recently, research focus has been moved to designing tailor-made protocol for specific tasks to achieve better performance [40,46,56]. The secure comparison and equality testing protocols are two widely used fundamental primitives in federated learning, privacy-preserving machine learning, private set intersection, advertising bidding systems, biometric authentication, and so on. These are two-party protocols in which one party inputs a and the other party inputs b. Together, they jointly evaluate whether $a > b$ or $a = b$ without disclosing private a and b. Hereby, we provide thereafter a non-exhaustive list of applications for secure comparison or equality testing.

- *Privacy-preserving machine learning.* Secure comparison is an important component for privacy-preserving machine learning [13, 17], especially for non-linear functions such as ReLU and MaxPool [56]. Furthermore, a line of research [31, 34, 39] aim to evaluate non-linear functions, such as Sigmod, GeLU, and Softmax, through secure comparison and secure polynomial evaluation.
- *Private set intersection.* Generally speaking, Private Set Intersection (PSI) [15, 35, 48] is a widely used protocol that enables two parties to securely compute a function over the intersected part of their shared datasets and has been a significant research focus over the years. Currently, in most PSI schemes, equality testing accounts for more than 50% of the total communication cost of the protocol [45]. Therefore, optimizing the communication cost of equality testing is of great importance for PSI.
- *Secure Data Mining.* Secure data mining [29, 43] can facilitate the identification of the most relevant items or patterns without exposing raw data. Secure comparison is often used in data mining tasks such as identifying the *top-k* items [24], outlier detection [50], and other analytics, where comparisons are necessary to draw insights from distributed datasets without compromising data privacy. Therefore, optimizing the performance of secure comparison can benefit numerous secure data mining application.

Moreover, the performance improvement of the secure comparison can also benefit a wide range of MPC applications. The efficiency of secure comparison in the multi-party setting [40, 56] has been significantly improved in the past years; whereas, in two-party computation (2PC) setting, secure comparison and equality testing are still major performance bottlenecks in practice. Indeed, as reported by the state-of-the-art (SOTA) [15, 31, 44] 2PC platforms, secure

†Tianpei Lu and Xin Kang contributed equally to this work.
§Bingsheng Zhang and Zhuo Ma are co-corresponding authors.

comparison/equality-testing is magnitude slower than other secure linear operations, e.g., secure multiplication.

A sequence of efforts [20,44] has been made to optimize the communication of the comparison or equality test. However, these proposed protocols are not constant rounds and suffer from poor performance in network scenarios with a large delay. The other approaches focus on the constant-round protocols. The typical solutions are based on the garbled circuit [49, 54] or the function secret sharing (FSS) [10, 12]. The garbled circuit scheme requires massive communication and computation in circuit evaluation (in the online phase), leading to a lower practical performance than protocols with logarithmic rounds. FSS gains better online communication efficiency compared to the garbled circuit scheme. Nevertheless, its online computation cost is close to that of a garbled circuit (GC). Only considering the online phase, to the best of our knowledge, FSS is the most efficient solution for both equality testing and secure comparison. However, conventional FSS is performed on the three-party scenario, which requires the third party to generate the correlated keys. Moving to the 2PC setting, the correlated key generation steps should be securely evaluated under 2PC (the user needs to evaluate massive PRGs), which is beyond practical. Recently, a line of works [21, 27] design correlated keys generation protocols that can eliminate the secure PRGs evaluation. As a trade-off, its computation cost is exponential to the input size n. Considering a large n, it is even impossible to terminate.

So far as we know, there does not exist a practical constant-round secure comparison or equality testing protocol that offers an efficient online phase with a practical offline phase.

1.1 Our Result.

In this work, we focus on secure equality testing and comparison in the two-party computation setting, i.e., Alice and Bob hold the secret input, respectively, and look for the shared result of the comparison or equality testing on their inputs. We also explain how to construct comparison and equality testing protocols for secret-shared values input using the protocols with private input. We design low (constant) communication-round protocols with relatively efficient offline phases, improving overall performance. We show that our protocols are secure against passive adversaries in the universal composability framework by Canetti [14].

2-round equality testing. Unlike the approach of bit-by-bit comparison a and b to obtain the result of the equality testing, we consider the problem as an oblivious retrieval problem. Specifically, the parties share a look-up table $\vec{T}$ in which only one random position has a value of 1, and all other positions are 0. The index of this position is shared between both parties, denoted as ε_0 and ε_1. Therefore, the parties can obtain the result of equality testing by locally selecting the shared value at the $(a-b+\varepsilon_0+\varepsilon_1)^{\mathsf{th}}$ position of $\vec{T}$. However, this approach requires a lookup table of length 2^n, which is impractical for real-world applications when a and b are n-bit integers. To address this problem, we propose a dimension reduction technique that reduces the problem of checking $a=b$ to the problem of checking $a'=b'$, where a' and b' are $\log n$ bits. This approach reduces the length of the required lookup table from 2^n and n.

In this work, we focus on optimizing the performance of the online phase while maintaining a reasonable offline phase overhead compared to FSS-based schemes. We emphasize the importance of the online phase, as many real-world applications prioritize online efficiency to achieve real-time responsiveness, specifically in scenarios demanding high responsiveness, such as PPML and PSI. In contrast, offline computations can be carried out during idle periods. By substantially reducing the online computational load and communication latency, our protocols provide practical advantages that outweigh the increased offline costs in such contexts. Furthermore, it is worth noting that our offline phase significantly outperforms FSS and remains competitive with OT-based solutions.

Secure comparison. We propose a novel 3-round secure comparison protocol Π_{cmp} in the semi-honest setting. Intuitively, our comparison protocol starts by extracting the first different-bit of inputs a and b from the big-endian. It is easy to see that the value of a on the position of different-bit corresponds to the result of the comparison $a>b$. We construct a secret shared list $\{s\}_n$ based on a transformation ϕ (cf § 4.1) highlighting such a position ζ. In particular, $s_\zeta=0$ and $s_i>0$ for other position i. Without security, we can directly reveal $\{s\}_n$ to P_0 for checking a_ζ of the highlighting position ζ. In § 4, we introduce a new primitive – Oblivious Selective Zero Check, to detect a_ζ without revealing $\{s\}_n$.

Performance. Table 3 depicts the performance comparison between our protocols and SOTA 2PC.

Our equality testing protocol requires 2 rounds of $O(n)$-bit communication in the online phase, which is close to FSS [21, 27]; note that our protocol does not invoke heavy PRFs, so the computational cost is much smaller than FSS, leading to a faster online phase, i.e. the running time of FSS-based equality testing is over $7\times$ more than ours, in the LAN/MAN/WAN setting. Moreover, compared to FSS, the offline of our protocol is $1000\times$ more efficient. For the other baseline, the garbled circuit–based equality testing [53,54], the online phase communication of our construction is less than 1%. Specifically, our benchmark shows that in the MAN setting, our protocol achieves $15\times$ better performance.

Our secure comparison protocol requires 3 rounds of $2n+2n\log n$ bits of communication in the online phase. Similarly to equality testing, our protocol outperforms the SOTA 2PC protocols. Compared to FSS-based comparison [27], our protocol achieves over $3\times$ online performance improvement, and over $1000\times$ offline performance improvement. For the garbled circuit-based comparison, the online phase communication of our comparison is less than 5%. Compared to the SOTA comparison CrypTFlow2 [47], our protocol achieves over 6× improvement in both MAN and WAN settings.

Paper Organization. § 2 introduces the preliminary including notations and the primitives to construct our protocols. The rest of the paper is organized as follows. In § 3, we propose our equality testing protocol involving one-round and two-round construction. In § 4, we introduce our three-round secure comparison protocol. § 5 conducts the performance evaluation of our equality testing and secure comparison protocols.

1.2 Related work

The concept of secure comparison was first proposed by Yao [53], a.k.a, millionaire's problem. Subsequently, equality testing called socialist millionaires' problem [33] has been successively proposed. The research in the areas has experienced rapid and consistent development. Due to the primitive similarities between secure protocols for equality tests and comparisons, we provide a unified representation. We categorize the works into five types based on the involved fundamental building blocks: GC-based-CMP/EQ, HE-based-CMP/EQ, OT-based-CMP/EQ, FSS-based-CMP/EQ, and Generic Two-Party Computation. In the following, we let n denote the input length.

GC-based-CMP/EQ. The secure comparison and equality testing protocols were initially constructed by Yao circuits [53]. Kolesnikov *et al.* [38] proposed a protocol for constructing universal circuits almost exclusively composed of XOR gates, which relies on the random oracle (RO) assumption. Then, they [37] optimize the assumption by allowing one party to garble circuits containing comparison gates, achieving secure comparison through AND gates. Zahur *et al.* [55] introduced an approach to garbling AND gates using two ciphertexts and XOR gates using zero ciphertexts concurrently, resulting in half the communication cost to compute AND gates. Despite the constant round complexity protocol realized, their communication amount is usually significant.

HE-based-CMP/EQ. The beginning of solving the millionaire problem from homomorphic encryption (HE) can be traced back to the protocol proposed by Blake *et al.* [9]. Subsequently, Garay *et al.* [25] proposed a secure comparison scheme based on threshold homomorphic encryption. However, the comparison can only be performed by a trusted third party. Cheon *et al.* [18] proposed a comparison scheme based on HE by using a composite polynomial approximation to obtain an approximate comparison result. However, this scheme is unable to achieve equality testing.

OT-based-CMP/EQ. When multiple instances of secure comparison or equality testing are needed, the approach based on oblivious transfer extension is commonly used. The method requires a constant number of public key operations and only inexpensive symmetric operations for each invocation. Couteau [19] proposed a scheme that relied on oblivious transfer (OT) to securely perform a bitwise comparison with n AND gates. Rathee *et al.* proposed a framework named CrypTFlow2 [47], which recursively equated the comparison of two integers to the comparison of sub-integers of length $(m \leq n)$. The sub-integer comparison was facilitated by 1-out-of-2^m OT.Therefore, the comparison could be implemented through $n/m-1$ AND gates. Subsequently, Chandran *et al.* [15] extends the idea to equality testing. Huang *et al.* [31] further optimized communication cost in CrypTFlow2 [47] by replacing the OT with VOLE-type OT.

FSS-based-CMP/EQ. Function secret sharing (FSS) [10, 12] allows two parties to evaluate a secure function with correlated keys locally, and output a shared result, whereas the typical solution requires a third party to generate the corresponding keys. The distributed point function (DPF) [12] can be used to realize the equality test directly and the distributed comparison function (DCF) [10] can be used to realize secure comparison. The correlated keys generation scheme [21, 27] employs FSS on the two parties' computation.

Generic Two-Party Computation. Generic two-party computation techniques enable secure computation of functions expressed as boolean circuits. Demmler *et al.* [20] presented a framework named ABY that efficiently combines arithmetic sharing, Boolean sharing, and Yao's garbled circuits to perform secure two-party computation. Secure comparison and equality testing could be efficiently instantiated by ABY. The process involved initially converting the secret input from arithmetic to Boolean form (A2B), followed by conducting bitwise comparisons, and finally reversing the transformation (B2A). Patra *et al.* [44] optimized multiplication computations in ABY2.0 by depending on function precomputation, reducing the communication cost during the online phase to half of that in ABY.

2 Preliminaries

Notation. Let $\mathcal{P} := \{P_0, P_1\}$ be the two MPC parties. We denote a vector $\{a_0, \ldots, a_{n-1}\}$ as $\vec{A}$, and a_i be the i^{th} element of $\vec{A}$. We denote $[n]$ as the index set $\{0, \ldots, n-1\}$, and $[1,n]$ as the index set $\{1, \ldots, n-1\}$. Let $\mathbf{1}\{b\}$ denote the indicator function that is 1 when b is true and 0 when b is false. Let $(1,n)$-OT denote the 1-out-of-n OT. We define $\mathsf{shift}(\vec{X}, i)$ as the operation that circularly shifts the vector $\vec{X}$ to the right by an offset of i. In addition, we define $[\cdot]^p$ over finite field $\mathbb{Z}_p$ as $[x]^p := ([x]_1 \in \mathbb{Z}_p, [x]_2 \in \mathbb{Z}_p)$ where $x = [x]_1 + [x]_2 \pmod{p}$. P_i for $i \in \{0,1\}$ hold share $[x]_i$. We use bold letters to denote matrices, e.g. $\mathbf{M}$, and the element in the i^{th} row and j^{th} column of $\mathbf{M}$ is denoted as $m_{(i,j)}$.

Threat model and security. Our equality testing and comparison protocols ensure security within the standard semi-honest setting. In this scenario, the adversary may attempt to extract private information from legitimate messages but must adhere strictly to the protocol's procedure. The security proof is

Table 1: Comparison with the state-of-the-art secure comparison and equality testing protocols. λ is the computational security parameter; μ is ECC group representation length and $\mu = 256$; n is the length of the element to be compared.

Approach	**Protocol**	**Offline**	**Online**	
		Communication	**Communication**	**#Round**
Equality Testing				
GC-based-EQ	Yao [52, 53]	$2n\lambda$	$2n\lambda$	2
Generic Two-Party Computation	ABY [20]	$6\lambda n + n$	$2\lambda n + 6n$	$\log n + 5$
	ABY2.0 [44]	$5\lambda n + 2n$	$\lambda n + 6n$	$\log n + 4$
FSS-based-EQ	Half-Tree [27]	$(n+2)\lambda^{+}$	$2n$	1
	DPF [12]	$4n(\lambda+1)+\lambda+n^{\dagger}$	$2n$	1
OT-based-EQ	CO [19]	$3\lambda n$	$2n + 2\log n + 10$	$\log^* n + 1$
	CGS [15]	$\frac{3}{4}\lambda n + 8n$	$5n - 4$	$\log n + 4$
	Π_{eq_2} (§ 3)	$\lambda \log n + n \log n + 3n + \lambda$	$2n + 2\log n + 2$	2
Secure Comparison				
GC-based-CMP	Yao [52, 53]	$2n\lambda$	$2n\lambda$	2
HE-based-CMP	GSV [25]	-	$18\mu n + 8\mu$	9
Generic Two-Party Computation	ABY [20]	$6\lambda n + 17\lambda + n$	$2\lambda n + 20n$	$\log n + 5$
	ABY2.0 [44]	$5\lambda n + 17\lambda + 2n$	$\lambda n + 9n$	$\log n + 4$
FSS-based-CMP	Half-Tree [27]	$(n+2)\lambda^{+}$	$2n$	1
	DCF [10]	$4n(\lambda+1)+\lambda+n^{\dagger}$	$2n$	1
OT-based-CMP	CO [19]	$6\lambda n$	$8n + 2\log n$	$4\log^* \lambda + 5$
	Cryptflow2 [47]	$\lambda n + 16n$	$10n - 8$	$\log n + 4$
	Π_{cmp}(§ 4)	$\approx 2n\lambda \log 2n + n \log^2 n$	$(2n+3)(\log n + 2)$	3

* $\log^*$ represents the iterated logarithm.
+ Under correlated keys generation scheme which performs $O(2^n)$ times Hash locally.
† Under a trusted third-party dealer.

based on the Universal Composability (UC) framework [14], which follows the simulation-based security paradigm. In the UC framework, protocols are executed across multiple inter-connected machines. The network adversary $\mathcal{A}$ is allowed to partially control the communication tapes of all uncorrupted machines, observing messages sent to/from uncorrupted parties and influencing message sequences. Then, a protocol Π is considered UC-secure in realizing a functionality $\mathcal{F}$ if, for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ targeting an execution of Π, there exists another PPT adversary known as a simulator $\mathcal{S}$ attacking the ideal execution of $\mathcal{F}$ such that the executions of Π with $\mathcal{A}$ and that of $\mathcal{F}$ with $\mathcal{S}$ are indistinguishable to any PPT environment $\mathcal{Z}$.

The idea world execution $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda)$. In the ideal world, the parties $\mathcal{P} := \{P_0, P_1\}$ only communicate with the ideal functionality $\mathcal{F}^f_{\mathsf{2pc}}$ with the excuted function f. Both parties send their share to $\mathcal{F}^f_{\mathsf{2pc}}$, and $\mathcal{F}^f_{\mathsf{2pc}}$ calculates and output the result to P_0 and P_1.

The real world execution $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\lambda)$. In the real world, the parties $\mathcal{P} := \{P_0, P_1\}$ communicate with each other, it executes the protocol Π. Our protocols work in the pre-processing model, but we analyze the offline and online protocols together as a whole.

Definition 1. We say protocol Π UC-secure realizes functionality $\mathcal{F}$ if for all PPT adversaries $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that for all PPT environment $\mathcal{Z}$, it holds:

$$\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\lambda) \approx \mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda)$$

Functionality $\mathcal{F}^f_{\mathsf{2pc}}$

$\mathcal{F}^f_{\mathsf{2pc}}$ interacts with P_0, P_1 and the adversary $\mathcal{S}$. Let f denote the functionality to be computed.

Input:

- Upon receiving $(\mathsf{Input}, \mathsf{sid}, a)$ from P_0, record a and send $(\mathsf{Input}, \mathsf{sid}, P_0)$ to $\mathcal{S}$, where $a \in \{0,1\}^n$.
- Upon receiving $(\mathsf{Input}, \mathsf{sid}, b)$ from P_1, record b and send $(\mathsf{Input}, \mathsf{sid}, P_1)$ to $\mathcal{S}$, where $b \in \{0,1\}^n$.

Execution:

- If both a, b are recorded, compute $(y_0, y_1) = f(a,b)$.
- Send (Output, y_0) to P_0 and (Output, y_1) to P_1.

Figure 1: The Ideal Functionality $\mathcal{F}^f_{\mathsf{2pc}}$.

Oblivious Transfer. For an instance of $(1,2)$-OT [22, 23], the sender sends the strings m_0 and $m_1 \in \{0,1\}^\ell$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$, and the receiver sends a select bit $i \in \{0,1\}$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$. As a result, the receiver obtains m_i from the $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$. Random OT (ROT) [8] is a special case of OT in which the message and the select bit are picked by $\mathcal{F}_{(1,2)\text{-ROT}}$ rather than input by parties. The sender receives two random strings r_0 and $r_1 \in \{0,1\}^l$, while the receiver obtains a bit $i \in \{0,1\}$ and m_i. A $n-1$-out-of-n ROT, its functionality $\mathcal{F}_{(n-1,n)\text{-ROT}}$ [16], sends list $\{m_0,\ldots,m_n\}$ to the sender, meanwhile, sends $b \in [n]$ and $\{m_i\}$ for $i \in [n]\backslash\{b\}$ to the receiver. We also utilize $(1,n)$-OT [42, 51]. Its functionality $\mathcal{F}_{(1,n)\text{-OT}}$ receives n strings $\{m_0,\cdots,m_{n-1}\}$ from sender, and the select index $i \in [n]$ from receiver. Subsequently, $\mathcal{F}_{(1,n)\text{-OT}}$ sends m_i to the receiver.

Oblivious Linear Evaluation. Oblivious Linear Evaluation (OLE) [6, 36] is a foundational component in various secure computation protocols [30, 46, 48]. In our protocol, we utilize its randomized variant – Random Oblivious Linear Evaluation (ROLE). In the standard ROLE protocol [6], P_0 receives random values a and b from the functionality $\mathcal{F}_{\text{ole}}$, while P_1 receives a random value u and $w = au+b$ from the functionality $\mathcal{F}_{\text{ole}}$. As the vectorize version – vector OLE (VOLE) [48], its functionality $\mathcal{F}_{\text{vole}}$ sends a random value u and a random vector $\vec{B}$ to P_0, at the same time, $\mathcal{F}_{\text{vole}}$ sends a random vector $\vec{A}$ and the vector $\vec{V}$ to P_1. It holds that $= \vec{A}u+\vec{B}$, i.e. $v_i = a_i u + b_i$ for each vector element $v_i \in \vec{V}, a_i \in \vec{A}, b_i \in \vec{B}$.

Secure permutation. The secure permutation [16] is a protocol that allows two parties, one of the parties holds the permutation and the other party holds the list, to jointly permute the list and obtain additive secret shares of the permutated list. Although this problem could be addressed using generic MPC, the most efficient implementation [16] currently is constructed by OT. We define the functionality $\mathcal{F}^{n,p}_{\text{Permute}}$ for n-dimension vector with element range $\mathbb{Z}_p$ as follow: the P_0 inputs a permutation π, and P_1 inputs a list $\vec{X} := \{x_0,\cdots,x_{n-1}\}$ where $x_i \in \mathbb{Z}_p$. After the protocol, they obtain the secret shares of the permuted list $\{x_{\pi(0)},\cdots,x_{\pi(n-1)}\}$.

3 Equality Testing

In the equality testing, P_0 inputs an integer $a \in \{0,1\}^n$ and P_1 inputs an integer $b \in \{0,1\}^n$. Both parties then receive a boolean share of $\mathbf{1}\{a=b\}$, which is equal to 1 if and only if $a=b$, and 0 otherwise.

In this section, we first design a one-round equality testing protocol with a communication complexity of $O(n)$ during the online phase. However, this design leads to an $O(2^n)$ communication complexity in the offline phase. To overcome the drawbacks, we propose a dimension reduction scheme to optimize the protocol. This optimization allows for two rounds of communication in the online phase while reducing the communication complexity in the offline phase from $O(2^n)$ to $O(n\log n)$.

Note that the equality testing over shared value $[a]$ and $[b]$ can be reduced to the equality testing over the private input. For the shared version, P_0 holds $[a]_0$ and $[b]_0$, and P_1 holds $[a]_1$ and $[b]_1$, where $a = [a]_0+[a]_1$ and $b = [b]_0+[b]_1$. We let P_0

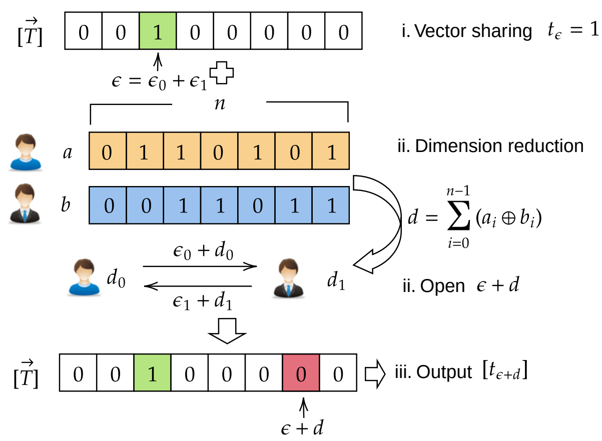


Figure 2: The Overview of Equality Testing

computes $a' = [a]_0 - [b]_0$ and P_1 computes $b' = [b]_1 - [a]_1$ as inputs for private-input equality testing. It works since $a = b$ implys $[a]_0+[a]_1 = [b]_0+[b]_1$, and thus $a' = [a]_0-[b]_0 = [b]_1-[a]_1 = b'$.

3.1 One-round equality testing

Starting point. Different from the idea of bit-by-bit comparison in related works, we consider equality testing as an oblivious retrieval problem. Specifically, P_0 generates a binary vector $\vec{T}$ as the look-up table, such that only the a^{th} value of $\vec{T}$ is 1, while the value of other positions are 0. P_1 then uses b to privately retrieves the b^{th} value from $\vec{T}$, denoted as t_b. Clearly, $t_b = 1$ if and only if $a = b$. To enumerate all strings of length n, the size of $\vec{T}$ is 2^n. For convenience, we define $N = 2^n$. However, the basic idea reveals the result of the equality testing to P_1 instead of sharing the result between P_0 and P_1. To keep t_b private to P_1, a simple approach is as follows: P_0 first samples a bit s and then computes $t'_i = s \oplus t_i$ for $i \in [N]$ to generate $\vec{T}'$, and then P_1 privately fetch t'_b instead of t_b. It is easy to see that $s \oplus t'_b = 1$ if and only if $a = b$. However, there is a drawback to the above approach that typically requires an instance of $\mathcal{F}_{(1,N)\text{-OT}}$ to fetch t'_b. The $(1,N)$-OT protocol has the huge communication complexity of $O(2^n)$ and requires two rounds of online communication. In the following, we will show how to overcome this drawback.

One-round equality testing. Our goal is to design an equality testing protocol that achieves one-round communication and $O(n)$ communication complexity in the online phase. At a high level, our protocol works as follows. In the offline phase, P_0 and P_1 respectively pick offsets $\varepsilon_0 \in [N]$ and $\varepsilon_1 \in [N]$, and then they generate a shared binary vector $\vec{T} := (t_0,\ldots,t_{N-1})$, where only $t_{\varepsilon_0+\varepsilon_1} = 1$. In the online phase, P_0 and P_1 open the value $w = \varepsilon_0+\varepsilon_1+a-b$, and then output $[t_w]_0$ and $[t_w]_1$ locally as the result of equality testing. Specifically, P_0 computes $w_0 = \varepsilon_0 + a$ and sends it to P_1. At the same round, P_1

computes $w_1 = \varepsilon_1 - b$ and sends it to P_0. Subsequently, P_0 and P_1 can recover w locally. Therefore, in the online phase, the communication cost is $2n$ bits, and only one round is required.

Recall that in the offline phase, P_0 picks an offset ε_0 and generate a binary vector $\vec{T}'$ with only $t'_{\varepsilon_0} = 1$. Then, P_0 and P_1 perform a right circular shift on $\vec{T}'$ by an offset of ε_1. Finally, P_0 obtains $\vec{T}_0$ and P_1 obtains ε_1 and $\vec{T}_1$, where $\vec{T}_0$ and $\vec{T}_1$ are the shares of $\vec{T} := \mathsf{shift}(\vec{T}', \varepsilon_1)$, such that $[t_i]_0 \oplus [t_i]_1 = t_i$. We construct the offline phase based on a new primitive – Vector Oblivious Shift Evaluation (VOSE).

Protocol $\Pi^N_{\mathsf{vose}}(\vec{T}')$

Input : P_0 inputs a binary vector $\vec{T}' \in \mathbb{Z}_2^N$.
Output : P_0 receives a share vector $\vec{T}_0$; P_1 receives a offset $\varepsilon_1 \in [N]$ and $\vec{T}_1$, where $\vec{T}_0 \oplus \vec{T}_1 = \mathsf{shift}(\vec{T}', \varepsilon_1)$.

Protocol:

1. P_0 and P_1 invoke $\mathcal{F}_{(N-1,N)\text{-}\mathsf{ROT}}$:
   - P_0 receives $\{m_i | i \in [N], m_i \in \mathbb{Z}_2^N\}$.
   - P_1 receives ε_1 and $\{m_i | i \in [N] \backslash \{\varepsilon_1\}, m_i \in \mathbb{Z}_2^N\}$.
2. P_0 and P_1 generate the binary matrix $\mathbf{M} \in \{0,1\}^{N \times N}$ by using m_i as the binary column vectors for $i \in [N]$, locally. (Note that P_1 does not have the $\varepsilon_1{}^{\text{th}}$ column of $\mathbf{M}$.)
3. For $i \in [N]$, P_0 and P_1 perform a right circular shift on the i^{th} row of their matrices by an offset of i locally.
4. P_0 computes $v_i = \bigoplus_{j=0}^{N-1} m_{(i,j)}$ and $u_i = \bigoplus_{j=0}^{N-1} m_{(j,i)}$ for $i \in [N]$, and denotes $\vec{V} := \{v_0, \ldots, v_{N-1}\}$ and $\vec{U} := \{u_0, \ldots, u_{N-1}\}$.
5. P_1 computes $w_i = \bigoplus_{j=0}^{\varepsilon_1+i-1} m_{(i,j)} \oplus \bigoplus_{j=\varepsilon_1+i+1}^{n-1} m_{(i,j)} \oplus \bigoplus_{j=0}^{i-1} m_{(j,\varepsilon_1+i)} \oplus \bigoplus_{j=i+1}^{n-1} m_{(j,\varepsilon_1+i)}$, which equals to $w_i = v_i \oplus u_{\varepsilon_1+i}$, and denotes $\vec{W} := \{w_0, \ldots, w_{N-1}\}$.
6. P_0 sends $\vec{S}' = \vec{T}' \oplus \vec{U}$ to P_1 and sets $\vec{T}_0 := \vec{V}$.
7. P_1 computes $\vec{T}_1 := \mathsf{shift}(S', \varepsilon_1) \oplus \vec{W}$.

Figure 3: The Vector Oblivious Shift Evaluation Protocol.

<u>*Random Vector Oblivious Shift Evaluation (RVOSE).*</u> Before introducing the standard VOSE, we first describe its randomized version (Π_{RVOSE}). In this protocol, P_0 receives two random binary vectors $\vec{U}$ and $\vec{V}$, and P_1 receives the offset ε_1 and a vector $\vec{W}$, such that $\vec{W} = \mathsf{shift}(\vec{U}, \varepsilon_1) \oplus \vec{V}$. The RVOSE can be built from $\mathcal{F}_{(N-1,N)\text{-}\mathsf{ROT}}$. As illustrated Figure 5, our protocol works as follows.

- P_0 and P_1 invoke $\mathcal{F}_{(N-1,N)\text{-}\mathsf{ROT}}$. After this, P_0 receives N messages $\{m_0, \ldots, m_{N-1}\}$ and $m_i \in \{0,1\}^N$. P_1 receives ε_1 and all messages except for m_{ε_1}. We view each message as a N-dimension binary vector and denote the binary matrix consisting of N-column vectors as $\mathbf{M}$. Therefore, P_0 obtains $\mathbf{M}$. P_1 obtains $(N-1)$ columns of $\mathbf{M}$ except for the $\varepsilon_1{}^{\text{th}}$ column.

Protocol $\Pi^N_{\mathsf{eq}_1}(a, b)$

The parameter N is defined as $N = 2^n$.
Input : P_0 inputs $a \in \{0,1\}^n$ and P_1 inputs $b \in \{0,1\}^n$.
Output : P_0 receives $[e]_0^2$ and P_1 receives $[e]_1^2$, where $[e]_0^2 \oplus [e]_1^2 = \mathbf{1}\{a = b\}$.

Offline:

1. For $i \in \{0,1\}$, P_i picks $\varepsilon_i \leftarrow [N]$.
2. P_0 generates a binary vector $\vec{T}' \in \mathbb{Z}_2^N$, where $t'_{\varepsilon_0} = 1$ and $t'_i = 0$ for $i \in [N] \backslash \{\varepsilon_0\}$.
3. P_0 and P_1 invoke $\{\vec{T}_0, \vec{T}_1\} \leftarrow \Pi^N_{\mathsf{vose}}(\vec{T}')$.

Online:

1. P_0 computes $w_0 = a + \varepsilon_0$ and sends it to P_1, while P_1 computes $w_1 = \varepsilon_1 - b$ and sends it to P_0.
2. P_0 and P_1 computes $w = w_0 + w_1$, locally.
3. For $i \in \{0,1\}$, P_i sets $[e]_i^2 = [t_w]_i$.

Figure 4: One-Round Equality Testing.

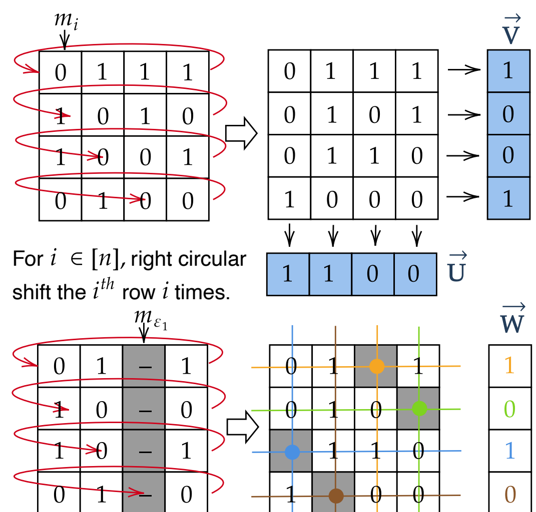


Figure 5: The Overview of the Random VOSE.

- For $i \in [N]$, P_0 and P_1 perform a right circular shift on the i^{th} row of their matrices by an offset of i. The new matrix of P_0 is denoted as $\mathbf{M}'$.
- For $i \in [N]$, P_0 computes $v_i = \bigoplus_{j=0}^{n-1} m_{(i,j)}$ and $u_i =$

$\bigoplus_{j=0}^{n-1} m_{(j,i)}$ to generate $\vec{V}$ and $\vec{U}$. Obviously, v_i is the XOR value of the N bits in the i^{th} row of $\mathbf{M}'$ and u_i is the value of the i^{th} column.

- For $i \in [N]$, P_1 computes w_i as $w_i = \bigoplus_{j=0}^{\varepsilon_1+i-1} m_{(i,j)} \oplus \bigoplus_{j=\varepsilon_1+i+1}^{n-1} m_{(i,j)} \oplus \bigoplus_{j=0}^{i-1} m_{(j,\varepsilon_1+i)} \oplus \bigoplus_{j=i+1}^{n-1} m_{(j,\varepsilon_1+i)}$, which equals to $w_i = v_i \oplus u_{\varepsilon_1+i}$. The resulting vector $\vec{W}$ satisfies $\vec{W} = \mathsf{shift}(\vec{U}, \varepsilon_1) \oplus \vec{V}$.

In conclusion, P_1 obtains $w_i = v_i \oplus u_{\varepsilon_1+i}$ for $i \in [N]$, while P_0 obtains u_i and v_i. Therefore, the vectors $\vec{W}, \vec{U}$ and $\vec{V}$ satisfy $\vec{W} = \mathsf{shift}(\vec{U}, \varepsilon_1) \oplus \vec{V}$.

Vector Oblivious Shift Evaluation (VOSE). Based on the RVOSE, we construct the VOSE in the following three steps. The protocol is shown in Figure 3.

- P_0 and P_1 invoke the Π_{RVOSE}. After this, P_0 receives $\vec{U}$ and $\vec{V}$, and P_1 receives the offset ε_1 and a vector $\vec{W}$, such that $\vec{W} = \mathsf{shift}(\vec{U}, \varepsilon_1) \oplus \vec{V}$.
- P_0 sends $\vec{S'} = \vec{T'} \oplus \vec{U}$ to P_1 and sets $\vec{T_0} = \vec{V}$.
- P_1 computes $\vec{T_1} = \mathsf{shift}(S', \varepsilon_1) \oplus \vec{W}$.

Correctness. $\vec{T_1} = \mathsf{shift}(\vec{T'}, \varepsilon_1) \oplus \mathsf{shift}(\vec{U}, \varepsilon_1) \oplus \mathsf{shift}(\vec{U}, \varepsilon_1) \oplus \vec{V} = \mathsf{shift}(\vec{T'}, \varepsilon_1) \oplus \vec{V} = \vec{T} \oplus \vec{V}$ and $\vec{T_0} = \vec{V}$. Fianlly, we have $\vec{T_0} \oplus \vec{T_1} = \vec{T}$.

Efficiency. In the offline phase, P_0 and P_1 invoke one time of $(N-1, N)$-ROT and P_0 send $\vec{S'} \in \mathbb{Z}_2^N$. The $(N-1, N)$-ROT [16] can be implemented with $n\lambda$ bits. Therefore, the communication cost in the offline phase is $n\lambda + N$. In the online phase, P_0 and P_1 send w_0 and w_1 to each other simultaneously, resulting in a communication cost of $2n$ bits.

3.2 Two-round equality testing

In the one-round equality testing protocol, the communication complexity in the offline phase is $O(2^n)$, which is impractical in real-world applications. We introduce a dimension reduction protocol that can reduce the communication complexity, i.e. from $O(2^n)$ to $O(n \log n)$. The overview of the protocol is shown in Figure 2.

Dimension reduction. The dimension reduction protocol is designed to reduce the integers $(a \in \{0,1\}^n, b \in \{0,1\}^n)$ to $(a' \in \{0,1\}^{\log n}, b' \in \{0,1\}^{\log n})$ such that $a' = b'$ if and only if $a = b$. The intuition of generating a' and b' is that, $d = \sum_{i=0}^{n-1}(a_i \oplus b_i) = 0$ if and only if $a = b$. Note that the maximum value of d is n. Thus, the arithmetic sharing of d, denoted as $[d]_0$ and $[d]_1$ where $d = [d]_0 + [d]_1$, can be used to represent a' and b' as $a' = [d]_0$ and $b' = -[d]_1$. For correctness, we have $a' - b' = [d]_0 + [d]_1 = d$.

To generate $[d]$, our approach is to convert the boolean sharing of a_i and b_i into arithmetic sharing s_i and t_i, such that $s_i + t_i = a_i \oplus b_i$. Consequently, P_0 and P_1 obtain the sharing of d by computing $[d]_0 = \sum_{i=1}^{n} s_i$ and $[d]_1 = \sum_{i=1}^{n} t_i$, respectively. We refer to the above conversion process as sharing conversion. Formally, in an instance of sharing conversion, P_0 and P_1 input the boolean sharing $[u]_0^2$ and $[u]_1^2$. At the end of the protocol, they receive the arithmetic sharing $[v]_0^p$ and $[v]_1^p$, satisfying $[v]_0^p + [v]_1^p = [u]_0^2 \oplus [u]_1^2$. Here, p is a prime larger than n. By the Bertrand's postulate [41], there always exists at least one prime number p with $n < p < 2n-2$. Thus, the next prime larger than n has at most $\log n + 1$ bits.

Protocol $\Pi_{\mathsf{convert}}^{2\to p}([u]_0^2, [u]_1^2)$

Input : P_0 inputs $[u]_0^2$; P_1 inputs $[u]_1^2$.
Output : P_0 receives $[v]_0^p$ and P_1 receives $[v]_1^p$, where $[v]_0^p + [v]_1^p = [u]_0^2 \oplus [u]_1^2$.

Offline:

1. P_0 samples $[r]_0^2$, and P_1 samples $[r]_1^2$.
2. P_0 and P_1 invoke $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$:
   - P_0 samples $[s]_0^p$.
   - P_0 as the sender inputs $m_0 = [s]_0^p - [r]_0^2$ and $m_1 = [s]_0^p - (1 - [r]_0^2)$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$;
   - P_1 as the receiver inputs $[r]_1^2$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$, and then receives $[s]_1^p := m_{[r]_1^2}$.
3. P_0 sets $[t]_0^p = [s]_0^p$ and P_1 set $[t]_1^p = -[s]_1^p$.

Online:

1. For $i \in \{0,1\}$, P_i computes $[w]_i^2 = [u]_i^2 \oplus [r]_i^2$ and sends $[w]_i^2$ to P_{1-i}.
2. P_0 and P_1 computes $w = [w]_0^2 \oplus [w]_1^2$, locally.
3. P_0 computes $[v]_0^p = w + [t]_0^p - 2w[t]_0^p$, and P_1 computes $[v]_1^p = [t]_1^p - 2w[t]_1^p$.

Figure 6: The Sharing Conversion Protocol.

Note that the sharing conversion can be easily constructed based on the $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$. In particular, P_0 samples $[s]_0^p$, and inputs $m_0 = [s]_0^p - [u]_0^2$ and $m_1 = [s]_0^p - (1 - [u]_0^2)$. P_1 inputs the selection bit $[u]_1^2$ and receives z, where $z = [s]_0^p - ([u]_0^2 \oplus [u]_1^2)$. Then, P_0 sets $[v]_0^p = [s]_0^p$ and P_1 sets $[v]_0^p = -z$. For correctness, we have $[v]_0^p + [v]_1^p = [s]_0^p - z = [s]_0^p - [s]_0^p - ([u]_0^2 \oplus [u]_1^2) = [u]_0^2 \oplus [u]_1^2$ as required. However, all the workload is currently performed in the online phase, resulting in the communication complexity is $O(n^2)$ and requiring two communication rounds.

Optimization of sharing conversion. We attempt to shift a significant portion of expensive operations to the offline phase, resulting in only a small amount of communication in the online phase. The protocol is described in Figure 6. In the offline phase, P_0 and P_1 generate a random sharing conversion pairs, i.e. P_0 receives $([r]_0^2, [t]_0^p)$ and P_1 receives $([r]_1^2, [t]_1^p)$, such that $[t]_0^p + [t]_1^p = [r]_0^2 \oplus [r]_1^2$. In the online phase, P_0 com-

Protocol $\Pi^n_{\mathsf{eq}_2}(a,b)$

Input : P_0 inputs $a \in \{0,1\}^n$ and P_1 inputs $b \in \{0,1\}^n$.
Output : P_0 receives $[e]^2_0$ and P_1 receives $[e]^2_1$, where $[e]^2_0 \oplus [e]^2_1 = \mathbf{1}\{a=b\}$.

Protocol:

1. For $i \in [n]$, P_0 and P_1 invoke $\{s_i, t_i \in \mathbb{Z}_p\} \leftarrow \Pi^{2\to p}_{\mathsf{convert}}(a_i, b_i)$, where $s_i + t_i = a_i \oplus b_i$.
2. P_0 computes $[d]_0 = \sum_{i=0}^{n-1} s_i$, and P_1 computes $[d]_1 = \sum_{i=0}^{n-1} t_i$ locally.
3. P_0 and P_1 invoke $([e]^2_0, [e]^2_0) \leftarrow \Pi^p_{\mathsf{eq}_1}([d]_0, -[d]_1)$.

Figure 7: Two-Round Equality Testing.

putes $[w]^2_0 = [a]^2_0 \oplus [r]^2_0$ and sends it to P_1; Meanwhile, P_1 computes $[w]^2_1 = [a]^2_1 \oplus [r]^2_1$ and sends it to P_0. Subsequently, P_0 and P_1 open the value $w = [w]^2_0 \oplus [w]^2_1$. Finally, P_0 sets $[v]^p_0 = w + [t]^p_0 - 2w[t]^p_0$ and P_1 sets $[v]^p_1 = [t]^p_1 - 2w[t]^p_1$ locally. Therefore, we have $[v]^p_0 + [v]^p_1 = [a]^2_0 \oplus [a]^2_1$.

Protocol description. As shown in Figure 7, our complete protocol works as follows.

- At step 1, P_0 and P_1 invoke n times of $\Pi^{2\to p}_{\mathsf{convert}}$ for a_i and b_i simultaneously. Then, they receive s_i and t_i for $i \in [n]$, such that $s_i + t_i = a_i \oplus b_i$.
- At step 2, P_0 computes $[d]_0 = \sum_{i=0}^{n-1} s_i$ and P_1 computes $[d]_1 = \sum_{i=0}^{n-1} t_i$, where it holds that $d = \sum_{i=0}^{n-1} a_i \oplus b_i$.
- At step 3, P_0 and P_1 invoke $\Pi^p_{\mathsf{eq}_1}([d]_0, -[d]_1)$ and receive $[e]_0$ and $[e]_1$. Then, they output $[e]_0$ and $[e]_1$ as the shared result of $\mathbf{1}\{a=b\}$.

Efficiency. In the offline phase, P_0 and P_1 invoke n times of $(1,2)$-OT and one time of $(2n-1,2n)$-ROT. In addition, P_0 send $\vec{S'} \in \mathbb{Z}_2^{2n}$. Note that $2^{\log p} = 2^{\log n+1} = 2n$. The $(1,2)$-OT [11, 32] can be implemented with an amortized communication cost of $n\log(2n)$ bits. Therefore, the corresponding communication cost in the offline phase is $n\log(2n) + \lambda\log(2n) + 2n = n(\log n+1) + \lambda(\log n+1) + 2n$ bits. In the online phase, P_0 and P_1 send n bits to each other in the sharing conversion $\Pi^{2\to p}_{\mathsf{convert}}$, and send p with $\log n+1$ bits to each other in the $\Pi^p_{\mathsf{eq}_1}$. Therefore, the rounds are 2 and the communication cost is $2n + 2\log n + 2$ bits.

Security. We define the functionality $\mathcal{F}_{\mathsf{eq}}$ for the equality testing as an instance of $\mathcal{F}_{\mathsf{2PC}}$. In this setup, $\mathcal{F}_{\mathsf{eq}}$ receives a from P_0 and b from P_1, computes $[e]^2_0 \oplus [e]^2_1 = \mathbf{1}\{a=b\}$, and sends $[e]^2_0$ to P_0 and $[e]^2_1$ to P_1. Note that to ensure security, the offline computations are designed to be single-use only. Next, we prove our protocol Π_{eq_2} UC-realizes functionality $\mathcal{F}_{\mathsf{eq}}$.

Theorem 1. *The protocol Π_{eq_2} as shown in Fig. 7 UC realizes $\mathcal{F}_{\mathsf{eq}}$ in the $(\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}, \mathcal{F}_{(n-1,n)\text{-}\mathsf{OT}})$-hybrid model against semi-honest probabilistic polynomial time (PPT) adversaries with statical corruption.*

Proof. cf. Appendix. C.1 for detail. ☐

4 Secure Comparison

In this section, we propose a novel secure comparison protocol where P_0 inputs $a \in \{0,1\}^n$ and P_1 inputs $b \in \{0,1\}^n$, receiving the shared result $\mathbf{1}\{a > b\}$. We first give an overview of our protocol using a new primitive – oblivious selective zero check (OZC) as a building block, which will be explained afterward. Our protocol can also build the comparison protocol over shared value.

Comparison over share. The comparison between shared value $[a]$ and $[b]$ can be reduced to the comparison between the private input a and b, by sacrificing 1-bit storage. In particular, let $[a]$ and $[b]$ denote the secret share over 2^n, and we take $a \in \mathbb{Z}_{2^{n-1}}$ and $b \in \mathbb{Z}_{2^{n-1}}$ which sacrifice the highest 1-bit storage. It holds that $\mathbf{1}\{a<b\} = \mathsf{sign}(a-b)$, where sign denote the sign-bit function. P_0 and P_1 first locally calculate $[c] = [a] - [b]$. To extract the sign-bit of shared value $[c]$, we observe that $\mathsf{sign}(c) = \mathsf{sign}([c]_0) \oplus \mathsf{sign}([c]_1) \oplus \mathbf{1}\{([c]_0 \mod 2^{n-1}) \le 2^{n-1} - ([c]_1 \mod 2^{n-1})\}$. It works as follows: expending $\mathsf{sign}(c) = \mathsf{sign}([c]_0 + [c]_1)$ as a circuit, the sign-bit of c equals the XOR result of $\mathsf{sign}([c]_0)$, $\mathsf{sign}([c]_1)$ and the carry-bit from adding the lower bits (besides of sign-bit) of $[c]_0$ and $[c]_1$. The carry-bit can be represented as $\mathbf{1}\{([c]_0 \mod 2^{n-1}) \le 2^{n-1} - ([c]_1 \mod 2^{n-1})\}$. $\mathsf{sign}([c]_0)$, $\mathsf{sign}([c]_1)$ can be locally evaluate, and $\mathbf{1}\{([c]_0 \mod 2^{n-1}) < 2^{n-1} - ([c]_1 \mod 2^{n-1})\}$ corresponds to comparison over the private input, in which P_0 holds $[c]_0 \mod 2^{n-1}$ and P_1 holds $2^{n-1} - [c]_1 \mod 2^{n-1}$.

4.1 Protocol Overview

For the integers a held by P_0 and b held by P_1, the result of comparison $\mathbf{1}\{a > b\}$ can be obtained by bitwise comparing a and b from the big-endian. Formally, it is denoted by $\mathbf{1}\{a > b\} = a_\rho$, where the position ρ correspond to the first different bit between a and b. Observe that in the case $a = b$ of which there are no different bits between a and b. To ensure $a \neq b$, we append 1 to the end of b and 0 to the end of a (analogously, we append 1 to a and 0 to b for $\mathbf{1}\{a \ge b\}$). Fig. 10 illustrates the overview of our secure comparison protocol. In the first step, we locate the position ρ. In the second step, we design a protocol to make two parties securely obtain the corresponding bit a_ρ which implies the comparison result.

First different bit detection. Lu *et.al* [40] introduce a transformation $\phi : (\mathbb{Z}_2)^n \mapsto (\mathbb{Z}_p)^n$ which can transfer any non-all-zero binary list $\{m_i\}_{i\in[n]}$ to an arithmetic list $\{s_i\}_{i\in[n]} \in (\mathbb{Z}_p)^n$ while $\{s_i\}_{i\in[n]} \in (\mathbb{Z}_p)^n$ holds that:

Functionality $\mathcal{F}_{\mathsf{ozc}}^{k,n,p}$

$\mathcal{F}_{\mathsf{ozc}}$ interacts with the parties $\mathcal{P}$ and the adversary $\mathcal{S}$.

Input:

- Upon receiving $(\mathsf{Input}, \mathsf{sid}, I, X)$ from $P_0 \in \mathcal{P}$, record (I, X) and send $(\mathsf{Input}, \mathsf{sid}, P_0)$ to $\mathcal{S}$, where
  - $X := \{x_0, \cdots, x_{n-1}\} \in (\mathbb{Z}_p)^n$;
  - $I \in (\mathbb{Z}_n)^k$;
- Upon receiving $(\mathsf{Input}, \mathsf{sid}, Y)$ from $P_1 \in \mathcal{P}$, record Y and send $(\mathsf{Input}, \mathsf{sid}, P_1)$ to $\mathcal{S}$, where .
  - $Y = \{y_0, \cdots, y_{n-1}\} \in (\mathbb{Z}_p)^n$;

Execution:

- If I, X and Y are recorded, $\mathcal{F}_{\mathsf{ozc}}$ does:
  - set $z = 1$ if $\exists i \in I, x_i + y_i = 0$.
  - set $z = 0$ otherwise.
- Send $(\mathsf{Output}, \mathsf{sid}, z)$ to P_1.

Figure 8: The Ideal Functionality $\mathcal{F}_{\mathsf{ozc}}$.

- contains a unique zero-value in the position ρ, and ρ corresponds to the first non-zero bit of binary list $\{m_i\}_{i\in[n]}$, namely, $m_i = 0$ for $i < \rho$, meanwhile, $m_\rho = 1$;
- contains positive values in any other positions.

Utilizing ϕ, we view m_i as the i^{th} bitwise-XOR of a and b, namely, $m_i = a_i \oplus b_i$, setting list $\{m_i\}$ as the input of ϕ. As mentioned, the position of zero value s_ρ corresponds to the first different bit between a and b.

Transformation ϕ. Let $\{s'_i\}_{i\in[n]}$ be the prefix sum of m_i. Specifically, $s'_i := \sum_{j=0}^{j=i} m_j$ for $i \in [n]$. We define $s_i = \phi(m_i) := s'_i - 2m_i + 1$. Obviously, when $i < \rho$, it holds that $m_i = s'_i = 0$, therefore, we have $s_i = 1$; when $i = \rho$, it holds that $s'_i = m_i = 1$, therefore, $s_i = 0$; when $i > \rho$, it holds that $s'_i \geq m_i + 1$, therefore, $s_i \geq 2 - m_i \geq 1$. In general, $s_i = 0$ if and only if $i = \rho$. Given a toy example, $a = 10010$ and $b = 10101$, we have $m = a \oplus b = 00111$, and then $s' = 00123$ and $s = 11012$.

In addition, we observe that $s_i \leq n$ (The maximum s_i takes n when $s'_{n-1} = n-1$ and $m_i = 0$). Above ϕ only contains linear operations that can be easily performed on the MPC setting. However, considering that wrapping around the modular will cause an extra 0, ϕ should be performed on $\mathbb{Z}_p$ where $p > n$ (such that $s_i \leq n$ will never wrap around), w.r.t. $[m_i]^p$ instead of $[m_i]^2$. As Bertrand's postulate [41], at least one prime number p lays on $[n, 2n-2]$ and its size can be taken as $\lceil \log n \rceil + 1 > \lceil \log(2n-2) \rceil$. For the share conversion part of $[m_i]^2 \in \mathbb{Z}_2$ to $[m_i]^p \in \mathbb{Z}_p$, we employ protocol $\Pi_{\mathsf{convert}}^{2\to p}$ in § 3.

Now we have shared list $\{[s_i]^p\}_{i\in[n]}$, where the position ρ of its zero element corresponds to the comparison result of a and b, that is, $a_\rho = \mathbf{1}\{a > b\}$. The second challenge is how P_0 and P_1 can obliviously obtain $[a_\rho]$ from $\{[s_i]^p\}_{i\in[n]}$ and a. To address this challenge, we introduce a new primitive – Oblivious Selective Zero Check. (OZC).

Oblivious Selective Zero Check. The OZC scheme checks if a shared list contains zero on a subsequence. We formalize its functionality in Fig. 8. In particular, an OZC functionality $\mathcal{F}_{\mathsf{ozc}}^{k,n,p}$ allows P_0 input k-dimension selective index set $I := \{\zeta_0, \ldots, \zeta_{k-1}\}$, P_0 and P_1 input n-dimension shared list $\{[x_i]\}_{i\in[n]}$. For $x_i = [x_i]_0 + [x_i]_1$, it checks if $\{x_{\zeta_i}\}_{i\in[k]}$ contains zero and sends the check result to P_1 (cf. Sec. 4.2).

Before going into the construction of OZC, we present a high-level overview of how we realize the secure comparison protocol on top of OZC and ϕ. Without considering security, We let P_0 toss a coin $\Delta \in \{0,1\}$ and input all the position $\{\zeta_i\}_{i\in[k]}$, where $a_{\zeta_i} = \Delta$, as the indices of $\mathcal{F}_{\mathsf{ozc}}$ (where a is input of P_0 and k is the number of bits in a equal to Δ). P_0 and P_1 input aforemationed $\{[s_i]^p\}_{i\in[n]}$, the result of ϕ, as the shared list of $\mathcal{F}_{\mathsf{ozc}}$. The result $z \in \{0,1\}$ which is given to P_1 means that:

- For the case $z = 0$, it indicates that all the bits of $a_{\zeta_i} = \Delta$ do not lay on the position ρ for $s_\rho = 0$, which implies $a_\rho = \Delta \oplus 1$.
- For the case $z = 1$, the positions $\{\zeta_i\}_{i\in[k]}$, in which $a_{\zeta_i} = \Delta$, contain ρ. Such case indicates $a_\rho = \Delta$.

Obviously, it holds that $a_\rho = \Delta \oplus z \oplus 1$. We let P_0 output the result $[c]_0 = \Delta$ and P_1 output $[c]_1 = z \oplus 1$.

Appending dummy queries. The number of queries k will leak the hamming weight of a to P_1. To avoid this leakage, we introduce dummy queries which pad the overall queries to the maximum possible number of queries. Firstly, we let P_0 and P_1 generate non-zero share $[s_n]^p$. We let P_0 perform extra $n-k$ queries using index n. Namely, for $i \in \{k, \ldots, n-1\}$, P_0 sets $\zeta_i = n$ and all parties invoke $\mathcal{F}_{\mathsf{ozc}}$ with n dimention indices and $(n+1)$ dimension shared list $\{[s_i]^p\}_{i\in[n+1]}$. Consequently, the overall queries are n.

Protocol description. As depicted in Figure 9, our full protocol works as follows.

- At step 1, P_0 and P_1 invoke $\Pi_{\mathsf{convert}}^{p}(a_i, b_i)$ for each bit a_i and b_i, receiving $[m_i]_0$ and $[m_i]_1$ respectively, such that $[m_i]_0 + [m_i]_1 = a_i \oplus b_i$.
- At step 2, P_0 and P_1 append 0 to a and 1 to b for dealing with $a = b$.
- At step 3, P_0 and P_1 compute $[s_i]_0 = \sum_{j=0}^{i} x_j - 2x_i + 1$ and $[s_i]_1 = \sum_{j=0}^{i} y_j - 2y_i + 1$, respectively. It holds that $s_\rho = 0$, where ρ denotes the position of the first differing bit between a and b.

Protocol $\Pi^n_{\mathsf{cmp}}(a,b)$

Input : P_0 inputs $a \in \mathbb{Z}_{2^n}$; P_1 inputs $b \in \mathbb{Z}_{2^n}$.
Output : P_0 receives $[c]^2{}_0 \in \mathbb{Z}_2$; P_1 receives $[c]^2_1 \in \mathbb{Z}_2$; it holds that $[c]^2_0 \oplus [c]^2_1 = \mathbf{1}\{a < b\}$.

Protocol:

- Let $p \in [n, 2n-2]$ be a prime, for $i \in [n]$, P_0 and P_1 invoke $[m_i] \leftarrow \Pi^p_{\mathsf{convert}}(a_i, b_i)$.
- P_0 sets $a_n = [m_n]_0 = 0$; P_1 sets $b_n = [m_n]_1 = 1$;
- For $i \in [n+1]$, P_0 computes $[s_i]_0 = \sum_{j=0}^{i}[m_j]_0 - 2\cdot[m_i]_0 + 1$, and P_1 computes $[s_i]_1 = \sum_{j=0}^{i}[m_j]_1 - 2\cdot[m_i]_1 + 1$;
- P_0 and P_1 sets $[s_{n+1}]_0 = [s_{n+1}]_1 = 1$;
- P_0 picks $\Delta \leftarrow \{0,1\}$;
- P_0 sets $I := \{\zeta_j\}_{j\in\mathbb{Z}_k} = \{i | a_i = \Delta, i \in \mathbb{Z}_{n+1}\}$, where we assume the size of I is k;
- For $j \in \{k, \ldots, n\}$, P_0 appends $\zeta_j = n+1$ to get $n+1$-dimension vector I';
- P_0 inputs index list I and $\{[s_i]_0\}_{i\in[n+2]}$ to $\mathcal{F}^{n+1,n+2,p}_{\mathsf{ozc}}$, P_1 inputs $\{[s_i]_1\}_{i\in[n+2]}$ to $\mathcal{F}^{n+1,n+2,p}_{\mathsf{ozc}}$ and receives $z \in \{0,1\}$;
- P_1 sets $[c]^2_1 = z \oplus 1$.
- P_0 set $[c]^2_0 = \Delta$.

Figure 9: The Comparison Protocol

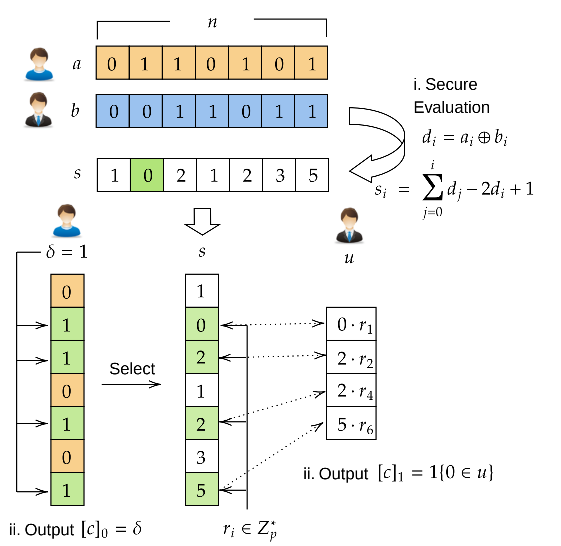


Figure 10: The Overview of Secure Comparison

- At step 4, P_0 and P_1 sets $[s_{n+1}]_0 = [s_{n+1}]_1 = 1$ for dummy queries.
- At steps 5-6, P_0 picks random Δ, records all indices i where $a_i = \Delta$, and denotes the set of these indices as I. We assume the size of the set I is k, namely, $I = \{\zeta_j\}_{j\in\mathbb{Z}_k}$.
- At step 7, to prevent the leakage of the hamming weight of a, P_0 pads the size of I to $n+1$. Therefore, P_0 appends $\zeta_j = n+1$ for $j \in n+1$.
- At step 8, P_0 and P_1 invoke $\mathcal{F}^{n+1,n+2,p}_{\mathsf{ozc}}$. Specifically, P_0 inputs the index list $I = \{\zeta_j\}_{j\in[n+1]}$ and the shared list $\{[s_i]_0\}_{i\in[n+1]}$, and P_1 inputs the shared list $\{[s_i]_0\}_{i\in[n+2]}$. At the end of protocol, P_1 receives $z = \mathbf{1}\left\{0 \in \{s_{\zeta_0}, \ldots, s_{\zeta_{k-1}}\}\right\}$.
- At steps 9-10, P_1 outputs $[c]_1 = z \oplus 1$ and P_0 outputs $[c]_0 = \Delta$.

Efficiency. Our secure comparison protocol Π^n_{cmp} requires to perform n times invoking of $\Pi^{2\to p}_{\mathsf{convert}}$ and one times of $\mathcal{F}_{\mathsf{ozc}}[n+1, n+2, p]$. The communication cost of $\mathcal{F}_{\mathsf{ozc}}[n+1, n+2, p]$, as we instantiate its protocol in the next section, requires 2-round $(2n+3)\lceil\log p\rceil = (2n+3)(\log n+1)$ bits communication in the online phase, and $n\log^2 n + 2n\log n + 2n\lambda\log 2n$ bits communication in the offline phase. As mentioned before, n times $\Pi^{2\to p}_{\mathsf{convert}}$ requires one-round $2n$ bits communication in the online phase and $2\lambda(\log n+1)$ bits communication in the offline phase. In summary, our secure comparison protocol Π^n_{cmp} requires 3-round communication of $4n + 3 + (2n+3)\log n$ bits in the online phase and $\lambda(\log n+1) + n\log^2 n + 2n\log n + 2n\lambda\log 2n \approx n\log^2 n + 2n\lambda\log 2n$ bits communication in the offline phase.
Security. We define the functionality $\mathcal{F}_{\mathsf{cmp}}$ depicted in Fig. 11. It is an instance of $\mathcal{F}_{\mathsf{2PC}}$ where $\mathcal{F}_{\mathsf{cmp}}$ receives $(\mathsf{Input}, \mathsf{sid}, a)$ from P_0 and $(\mathsf{Input}, \mathsf{sid}, b)$ from P_1, picks random value $[c]^2_0 \leftarrow \mathbb{Z}_2$, if P_0 is corrupted, receives $(\mathsf{Modify}, \mathsf{sid}, [c]^2_0)$ from $\mathcal{A}$, calculates $[c]^2_1 = \mathbf{1}\{a > b\} \oplus [c]^2_0$ and sends $(\mathsf{Output}, \mathsf{sid}, [c]^2_0)$ to P_0 and $(\mathsf{Output}, \mathsf{sid}, [c]^2_1)$ to P_1. Next, we prove our protocol Π_{cmp} realizes functionality $\mathcal{F}_{\mathsf{cmp}}$. Similar to equality testing, the offline computations are strictly one-time use.

Theorem 2. *The protocol Π_{cmp} as depicted in Fig. 9 UC realizes $\mathcal{F}_{\mathsf{cmp}}$ in the $(\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}, \mathcal{F}_{\mathsf{ozc}})$-hybrid model against semi-honest PPT adversaries with statical curroption.*

Proof. cf. Appendix. C.2 for detail. □

4.2 Construction of $\mathcal{F}_{\mathsf{ozc}}$

We first provide a basic construction of the OZC protocol, which requires heavy communication in the online phase. After that, we optimize the communication of the online phase by introducing the permutation tuples, which can be generated in the offline phase.

Functionality $\mathcal{F}^n_{\mathsf{cmp}}$

$\mathcal{F}^n_{\mathsf{cmp}}$ interacts with P_0, P_1 and the adversary $\mathcal{S}$. Let cmp denote the comparison function.

Input:

- Upon receiving $(\mathsf{Input}, \mathsf{sid}, a)$ from P_0, record a and send $(\mathsf{Input}, \mathsf{sid}, P_0)$ to $\mathcal{S}$, where $a \in \{0,1\}^n$.
- Upon receiving $(\mathsf{Input}, \mathsf{sid}, b)$ from P_1, record b and send $(\mathsf{Input}, \mathsf{sid}, P_1)$ to $\mathcal{S}$, where $b \in \{0,1\}^n$.

Execution:

- If both a, b are recorded, pick $[c]^2_0 \leftarrow \mathbb{Z}_2$;
- If P_0 is corrupted, receive $(\mathsf{Modify}, \mathsf{sid}, [c]^2_0)$ from $\mathcal{S}$;
- Calculate $[c]^2_1 = \mathbf{1}\{a > b\} \oplus [c]^2_0$;
- Send $(\mathsf{Output}, \mathsf{sid}, [c]^2_0)$ to P_0 and $(\mathsf{Output}, \mathsf{sid}, [c]^2_1)$ to P_1.

Figure 11: The Ideal Functionality $\mathcal{F}_{\mathsf{cmp}}$.

The Basic Approach. Recall that the functionality $\mathcal{F}_{\mathsf{ozc}}$ receives a list $X := \{x_0, \ldots, x_{n-1}\}$ and a index list $I := \{\zeta_0, \ldots, \zeta_{k-1}\}$ from P_0, while receives $Y := \{y_0, \ldots, y_{n-1}\}$ from P_1. $\mathcal{F}_{\mathsf{ozc}}$ then sets a bit $z = 1$ if there exists $\zeta_i \in I$ such that $x_{\zeta_i} + y_{\zeta_i} = 0$ and $z = 0$ otherwise. After that, $\mathcal{F}_{\mathsf{ozc}}$ sends z to P_1. Without considering security, we let P_0 directly fetch y_{ζ_i} from P_1 using $(1,n)$-OT, and check if $x_{\zeta_i} + y_{\zeta_i}$ equals to 0 for all $\zeta_i \in I$. However, some challenges remain:

- The zero checking task should be performed by P_1 rather than P_0 (according to $\mathcal{F}_{\mathsf{ozc}}$);
- The plaintext value of $x_{\zeta_i} + y_{\zeta_i}$ should not be leaked to any party.

To address the first challenge, we let P_0 and P_1 generate the share of $d_i = x_{\zeta_i} + y_{\zeta_i}$ rather than plaintext d_i held by P_0, and open d_i to P_1. In particular, for each index ζ_i held by P_0, P_1 inputs $\{y_0 + r_i, \ldots, y_{n-1} + r_i\}$ to $(1,n)$-OT instead of Y, where r_i is a fresh random coin. P_0 sets $[d_i]_0 = x_{\zeta_i} + y_{\zeta_i} + r_i$ and P_1 sets $[d_i]_1 = -r_i$. Depending on who receives the output of the zero-checking task, we open d_i to the corresponding party.

For the second challenge, to avoid of leaking $x_{\zeta_i} + y_{\zeta_i}$ to P_1, we introduce a non-zero scaler β_i for each $x_{\zeta_i} + y_{\zeta_i}$, $\zeta_i \in I$. In short, we turn to use $d_i = \beta_i \cdot (x_{\zeta_i} + y_{\zeta_i})$ rather than $d_i = x_{\zeta_i} + y_{\zeta_i}$. It is easy to see that if $x_{\zeta_i} + y_{\zeta_i} \neq 0$ than d_i is a random value; while, if $x_{\zeta_i} + y_{\zeta_i} = 0$ then $d_i = 0$. To avoid potential errors ($d_i = 0$ yet $x_{\zeta_i} + y_{\zeta_i} \neq 0$) caused by wrapping around, we take p as prime and $\beta_i \in \mathbb{Z}^*_p$. The share $[d_i]$ can be calculated by $[d_i] = \beta_i \cdot (x_{\zeta_i} + y_{\zeta_i} + r_i) - [\beta_i \cdot r_i]$. In the actual protocol, we let P_0 calculate $x_{\zeta_i} + y_{\zeta_i} + r_i$ as before and pick $\beta_i \leftarrow \mathbb{Z}^*_p$. For the share $[\beta_i \cdot r_i]$, we adopt the Oblivious Linear Evaluation (OLE) $\mathcal{F}_{\mathsf{ole}}$ in which P_0 inputs β_i, P_1 inputs r_i, and each parties receives the corresponding share of $[\beta_i \cdot r_i]$ as outputs. Subsequently, P_0 sets $[d_i]_0 = \beta_i \cdot (x_{\zeta_i} + y_{\zeta_i} + r_i) - [\beta_i \cdot r_i]_0$, while P_1 sets $[d_i]_1 = -[\beta_i \cdot r_i]_1$. After opening each d_i for $i \in [k]$ to P_1, P_1 can detect whether there exists $x_{\zeta_i} + y_{\zeta_i} = 0$ through zero checking for each d_i. The formal description of our basic approach is shown in Fig. 16 (cf. Appendix. A.2).

Protocol $\Pi^{k,n,p}_{\mathsf{ozc}}(I, X, Y)$

Input : Index list $I := \{\zeta_i\}_{i \in [k]}$ input by P_0 which contains $k - t$ non-repeating items, and last t indices equal to n; list $X := \{x_i\}_{i \in [n]}$ input by P_0; list $Y := \{y_i\}_{i \in [n]}$ input by P_1;
Output : P_1 receives $z = 1$ if exists $\zeta_i \in I$ such that $x_{\zeta_i} + y_{\zeta_i} = 0$, otherwise, P_1 receives $z = 0$.

Offline:

- P_0 and P_1 invoke:
  - $(\beta_i, r_i, u_i, v_i) \leftarrow \mathcal{F}^p_{\mathsf{ole}}$, for $i \in [n]$.
  - $(\{\beta_j\}_{j \in [k-1]}, r, \{u_j\}_{j \in [k-1]}, \{v_j\}_{j \in [k-1]}) \leftarrow \mathcal{F}^{k-1,p}_{\mathsf{vole}}$
- P_1 concatenates $\{\beta_j\}_{j \in [k-1]}, r, \{u_j\}_{j \in [k-1]}, \{v_j\}_{j \in [k-1]}$ with β_i, r_i, u_i, v_i where copy $k-2$ copies of r as alignment;
- P_0 picks random permutation $\pi : S_{n+k-1} \mapsto S_{n+k-1}$;
- P_0 and P_1 invoke $\mathcal{F}^{n+k-1,p}_{\mathsf{permute}}$:
  - P_0 inputs the permutation π, and P_1 inputs the list $\{v_i\}_{i \in [n+k-1]}$.
  - P_0 receives the sharing list $\{[v_{\pi(i)}]_0\}_{i \in [n+k-1]}$ and P_1 receives $\{[v_{\pi(i)}]_1\}_{i \in [n+k-1]}$, respectively.
- P_0 sets $[w_i]_0 = [v_i]_0 + u_{\pi(i)}$; P_1 sets $[w_i]_1 = [v_i]_1$

Online:

- P_1 sets $y'_i = y_i + r_i$ for $i \in [n]$ and sends the set $Y' = \{y'_0, \cdots, y'_{n-1}\}$ to P_0;
- P_0 sets
  - $[d_i]_0 = \beta_{\zeta_i} \cdot (x_{\zeta_i} + y'_{\zeta_i}) - [w_{\pi^-(\zeta_i)}]_0$ for $i \in [k-t]$;
  - $s_i = \pi^-(\zeta_i)$ for $i \in [k-t]$;
  - $[d_i]_0 = \beta_{n+i-k} \cdot (x_n + y'_n) - [w_{\pi^-(n+i-k)}]_0$ for $i \in [k-t, k]$;
  - $s_i = \pi^-(n+i-k)$ for $i \in [k-t, k]$;
- P_0 sends $\{[d_i]_0\}_{i \in [k]}$ and $\{s_i\}_{i \in [k]}$ to P_1.
- P_1 calculates $d_i = [d_i]_0 - [w_{s_i}]_1$ for $i \in [k-t]$.
- P_1 outputs $z = \mathbf{1}\{0 \in \{d_0, \cdots, d_{k-1}\}\}$.

Figure 12: The Oblivious Selective Multiplication Protocol

Online Phase Communication Optimization. For k indices, the above basic approach requires invoking k times 1-out-of-n OT in the online phase, which causes a huge communication cost. We optimize the online phase communication through the oblivious permutation. Revisit the above basic approach, we observe that if the mask value r is independent and different for each item rather than single r, namely, P_1 input $\{y_0+r_0,\dots,y_{n-1}+r_{n-1}\}$, such a vector can be sent to P_0 directly. Using $I:=\{\zeta_0,\dots,\zeta_{k-1}\}$, P_0 can calculate $\beta_{\zeta_i}(x_{\zeta_i}+y_{\zeta_i}+r_{\zeta_i})$, where β_{ζ_i} is a fresh random coin. Notice that in the basic approach, each item corresponds to the same r_i, hence P_1 only requires input r_i to evaluate deterministic OLE $\beta_i\cdot r_i$; Nevertheless, in the optimized approach, ζ_i of $\beta_{\zeta_i}\cdot r_{\zeta_i}$ is unknown to P_1, resulting in the challenge in eliminating $\beta_{\zeta_i}\cdot r_{\zeta_i}$.

We tackle this challenge by introducing permutation tuples. More sepcifically, we can generate a list of $[\beta_i\cdot r_i]$ where P_0 holds β_i, P_1 holds r_i, and the shared product $[\beta_i\cdot r_i]$ is randomly permuted with π, namely $[w_{\pi(i)}]=[\beta_i\cdot r_i]$. Letting P_0 be aware of π, P_0 can notify P_1 the shared product $[w_{\pi(\zeta_i)}]=[\beta_{\zeta_i}\cdot r_{\zeta_i}]$ with the corresponding permuted index $\pi(\zeta_i)$ without revealing ζ_i. Utilizing $\pi(\zeta_i)$, both parties evaluate $\beta_{\zeta_i}(x_{\zeta_i}+y_{\zeta_i}+r_{\zeta_i})-[\beta_{\zeta_i}\cdot r_{\zeta_i}]$ and reveal it to P_1 for zero checking.

Formally, we define the permutation tuple as $(\{\beta_i,r_i,[w_i]_0,[w_i]_1\}_{i\in[n]},\pi)$. In detail,

- π is a random permutation held by P_0 (we use $\pi(i)$ to denote the permuted result of i);
- $\beta_i\cdot r_i=[w_{\pi(i)}]_0+[w_{\pi(i)}]_1$ are the permuted OLE tuples, where P_0 holds $(\{\beta_i,[w_i]_0\}_{i\in[n]})$ and P_1 holds $(\{r_i,[w_i]_1\}_{i\in[n]})$.

Considering $d_i=\beta_{\zeta_i}(x_{\zeta_i}+y_{\zeta_i}+r_{\zeta_i})-\beta_{\zeta_i}\cdot r_{\zeta_i}$, we replace $\beta_{\zeta_i}\cdot r_{\zeta_i}$ with $[w_{\pi(\zeta_i)}]_0+[w_{\pi(\zeta_i)}]_1$. Namely, $d_i=\beta_{\zeta_i}(x_{\zeta_i}+y_{\zeta_i}+r_{\zeta_i})-[w_{\pi(\zeta_i)}]_0-[w_{\pi(\zeta_i)}]_1$. Since P_0 is aware of ζ_i, we let P_0 calculate $[d_i]_0=\beta_{\zeta_i}(x_{\zeta_i}+y_{\zeta_i}+r_{\zeta_i})-[w_{\pi(\zeta_i)}]_0$ and send both $[d_i]_0$ and $\pi(\zeta_i)$ to P_1. $\pi(\zeta_i)$ can be revealed to P_1 directly without leaking ζ_i, due to the fact that the random permutation π is unknown to P_1. Subsequently, P_1 selects $[d_i]_1=-[w_{\pi(\zeta_i)}]_1$ and reconstructs $d_i=[d_i]_0+[d_i]_1$.

Dealing with dummy queries. The foregoing version of the protocol cannot deal with the duplicated indices. Because the same index ζ_i will obtain the same permuted index $\pi(\zeta_i)$ which can not be directly revealed to P_1, leading to an incompatible with the original dummy queries approach. Considering that the dummy queries return the dummy item appended in the tail, we generate $k-1$ duplications (k is the max number of queries) of the mask r corresponding to the dummy item and produce the permutation tuple using the duplications. Since different β_{ζ_i} correlates to the same r, we utilize the VOLE permutation tuple $(\{\beta_i,[w_i]_0,[w_i]_1\}_{i\in[k-1]},r)$ for dummy queries. In detail,

- $\beta_i\cdot r=[w_{\pi(i)}]_0+[w_{\pi(i)}]_1$ are the permuted VOLE tuples;
- P_0 holds $(\beta_i,[w_i]_0)$ and P_1 holds $(r,[w_i]_1)$.

The VOLE tuple is concatenated with the original OLE tuples and the $\pi:\mathbb{Z}_p^{n+k-1}\mapsto\mathbb{Z}_p^{n+k-1}$ is performed on the concatenated tuples; namely, $(\{\beta_i,r_i,[w_i]_0,[w_i]_1\}_{i\in[n+k-1]},\pi)$ where $r_n=r_{n+1}\dots=r_{n+k}$ corresponds to the r of VOLE tuple. In particular, assume the last t items of I are duplicated indices (as the dummy queries) and their values equal to r_n, i.e. $\zeta_i=n$ for $i\in[k-t,k]$. P_0 sets $[d_i]_0=\beta_{n+i-k+t}\cdot(x_n+y_n+r_n)-[w_{\pi(n+i-k+t)}]_0$ and sends $[d_i]_0$ and $\pi(n+i-k+t)$ to P_1. Clearly, it holds that $w_{\pi(n+i-k+t)}=\beta_{n+i-k+t}\cdot r_{n+i-k+t}=\beta_{n+i-k+t}\cdot r_n$ so that the reveal value d_i equals to $\beta_{n+i-k+t}\cdot(x_n+y_n)$, since $r_n=r_{n+1}\dots=r_{n+k}$. Note that though dummy queries utilize the same item $x_n+y_n+r_n$, and different $\beta_{n+i-k+t}$ can ensure each reveal value $d_i=\beta_{n+i-k+t}\cdot(x_n+y_n)$ is individual.

Offline tuples generation. We generate the offline truples with three primitives: $\mathcal{F}_{\mathsf{ole}},\mathcal{F}_{\mathsf{vole}},\mathcal{F}_{\mathsf{permute}}$. We let $\mathcal{F}_{\mathsf{ole}}$ and $\mathcal{F}_{\mathsf{vole}}$ generate the OLE tuples and VOLE tuples for dummy queries, denote them as $\{\beta_i,r_i,u_i,v_i\}$ where $\beta_i\cdot r_i=u_i+v_i$. We let P_0 input random permutation π and P_1 input list $\{v_i\}_{i\in[n]}$ to functionality $\mathcal{F}_{\mathsf{permute}}$. After that P_0 and P_1 receive $[v_{\pi(i)}]$ and calculate $[w_i]=[v_{\pi(i)}]+u_{\pi(i)}$. Now we have $\beta_i\cdot r_i=[w_{\pi^-(i)}]_0+[w_{\pi^-(i)}]_1$ for $i\in[n]$, while π^- denote the inverse of π. In our benchmark, we use the SOTA protocol to realize $\mathcal{F}_{\mathsf{ole}}$ [36],$\mathcal{F}_{\mathsf{vole}}$ [48],$\mathcal{F}_{\mathsf{permute}}$ [16]. Our complete protocol design is illustrated in Figure. 12.

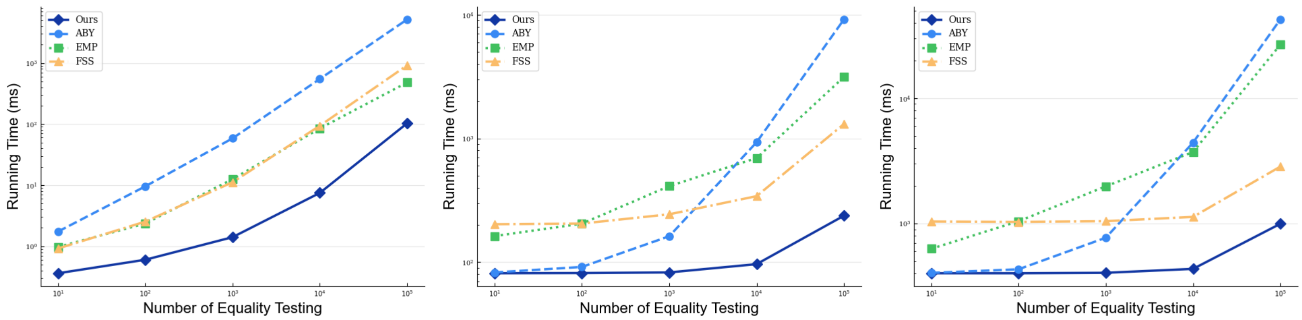
Efficiency. Our oblivious selective zero checking protocol $\Pi_{\mathsf{ozc}}^{k,n,p}$ requires 2-round communication of $(k+n)\cdot\lceil\log p\rceil$ bits in the online phase. In the offline phase, it requires n times invoking of Π_{ole}^{p} (the instance of $\mathcal{F}_{\mathsf{ole}}^{p}$ [36], n times invoking require $n\log^2 p$ bits communicaiton), one time invoking of $\Pi_{\mathsf{vole}}^{k-1,p}$ (the instance of $\mathcal{F}_{\mathsf{vole}}^{k-1,p}$ [48], it requires $2(k-1)\log p$ bits communicaiton) and one time invoking of $\Pi_{\mathsf{permute}}^{n+k-1,p}$ (the instance of $\mathcal{F}_{\mathsf{permute}}^{n+k-1,p}$ [16] requires $(k+n-1)\lambda\log(k+n-1)$ bits communication). In summary, our oblivious selective zero checking protocol $\Pi_{\mathsf{ozc}}^{k,n,p}$ requires $n\log^2 p+2(k-1)\log p+(k+n-1)\lambda\log(k+n-1)$ bits communication in the offline phase. For the invoking of $\Pi_{\mathsf{ozc}}^{n+1,n+2,p}$ in aforementioned comparison, its offline communicaiton cost approximate to $n\log^2 n+2n\log n+2n\lambda\log 2n$.

Theorem 3. *The protocol Π_{ozc} as depicted in Fig. 12 UC realizes $\mathcal{F}_{\mathsf{ozc}}$ in the $(\mathcal{F}_{\mathsf{ole}},\mathcal{F}_{\mathsf{vole}},\mathcal{F}_{\mathsf{permute}})$-hybrid model against semi-honest PPT adversaries who can statically corrupt up to one party.*

Proof. cf. Appendix. C.3 for detail. ☐

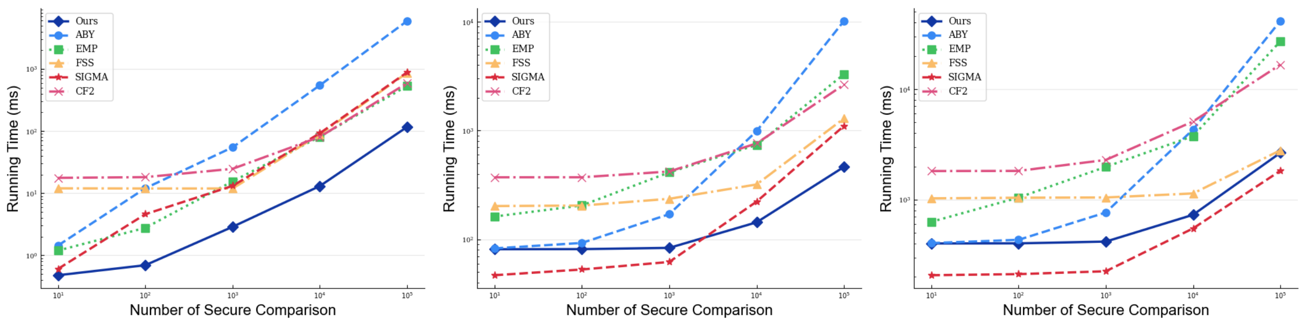
5 Performance Evalutaion

In this section, we respectively implement our equality test (Section 3) and secure comparison (Section 4), and com-



(a) LAN (b) MAN (c) WAN

Figure 13: The running time in the online phase of equality testing protocol Π_{eq_2} compare with ABY [20], GC scheme implemented in EMP-toolkits [52] and DPF [27] in LAN/MAN/WAN setting. All benchmarks take the data length $n = 64$.



(a) LAN (b) MAN (c) WAN

Figure 14: The running time in the online phase of Π_{cmp} compare with ABY [20], GC implemented in EMP [52], DCF [27] SIGMA [28] and CrypFlow2 [47] in LAN/MAN/WAN setting; take the data length $n = 64$; CF2 refers to CrypTFlow2.

pare their performance with the CrypTFlow2 [47], ABY [20], GC [3], FSS [27].

5.1 Experiment Setting

We implement our protocols in C++. For the $\mathcal{F}_{\mathsf{OT}}$, we utilize the OT library – libOTe [4]. For 2PC FSS, there are two primary approaches for offline implementations: (1) a variant of the secure DPF generation protocol proposed by Doerner and Shelat [21] (Figure 7), which was later extended to DCF by Elette Boyle et al. [12](Appendix A.1); and (2) generic two-party computation methods that implement the PRG using AES or specially designed "MPC-friendly" ciphers. We follow the first approach, which migrates the PRG to a local computation scheme. We update our code on the anonymous GitHub repository [1]. A key advantage of this method is that it allows MPC to perform only linear computations, without requiring the execution of the PRG within the MPC framework. For the garbled circuit, we utilize EMP-toolkits [3], which is integrated half-gate [55]. The source code of our protocol can be obtained from the anonymous GitHub repository [5]. For ABY and CrypTFlow, we utilize their open-source code [2]. Our experiments are performed in a local area network, using traffic control in Linux to simulate three network settings: (1) local-area settings (LAN): 20Gbps bandwidth with 0.01 ms round-trip latency (RTT). (2) metropolitan-area setting (MAN): 400 Mbps bandwidth with 20 ms round-trip. (3) wide-area setting (WAN): 10Mbps bandwidth with 100 ms round-trip. Our benchmark setting is deployed on the server running Ubuntu 18.04.2 LTS with Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz, 48 CPUs, 128 GB Memory. In our benchmark, we set the security parameter $\lambda = 128$. We select the most commonly used constant-round protocols based on different techniques as our baseline. ABY [20] is based on binary circuits; emp-tool [52] is based on the garbled circuit; SIGMA [28] and DCF [10] are based on FSS and CrypTFlow2 [47] is the typical secret-share-based constant round secure comparison. Since SIGMA's offline phase requires the participation of an additional server, we only compare our online phase with SIGMA's.

5.2 Experiment Evaluation

In this section, we evaluate the performance of our equality test and secure comparison.

Equality testing. The equality testing running time of the online phase (for $n = 64$) is depicted in Fig. 13. In Table 2, we present the runtime under a LAN setting for different bit

Table 2: Running time of our protocols compared to baselines (given in ms) in the LAN setting.

Batch Size		Element Size	8			16			32			64		
			Online	**Offline**	**Total**	**Online**	**Offline**	**Total**	**Online**	**Offline**	**Total**	**Online**	**Offline**	**Total**
100	Equality Testing	ABY [20]	1.78	3.08	4.86	2.87	4.20	7.07	4.58	9.91	14.50	10.12	16.82	26.94
		EMP [52]	0.85	0	0.85	1	0	1	1.38	0	1.38	2.21	0	2.21
		FSS [10]	0.70	1028.96	1029.66	0.70	3095.14	3095.84	0.84	-	-	1.49	-	-
		Ours	0.35	3.85	4.20	0.43	6.90	7.33	0.48	11.11	11.59	0.51	17.50	18.01
	Secure Comparison	ABY [20]	2.14	3.21	5.35	3.55	5.67	9.22	6.31	10.34	16.65	9.23	16.52	25.75
		EMP [52]	1.02	0	1.02	1.17	0	1.17	1.58	0	1.58	2.39	0	2.39
		FSS [10]	0.57	1346.74	1347.31	0.84	4593.09	4593.93	1.14	-	-	1.32	-	-
		SIGMA [28]	1.92	-	-	2.55	-	-	4.41	-	-	4.59	-	-
		Ours	0.51	35.84	36.35	0.58	53.51	54.09	0.56	108.26	108.82	0.69	338.19	338.88
1000	Equality Testing	ABY [20]	12.99	19.98	32.97	18.56	35.03	53.39	25.80	64.03	89.83	56.62	109.21	165.83
		EMP [52]	3.23	0	3.23	5.22	0	5.22	12.10	0	12.10	15.56	0	15.56
		FSS [10]	1.81	10344.30	10346.11	3.20	31406.90	31410.10	5.36	-	-	10.14	-	-
		Ours	0.49	22.11	22.60	0.59	35.52	36.11	0.83	65.91	66.74	1.21	138.28	139.49
	Secure Comparison	ABY [20]	12.11	21.33	33.44	20.66	38.77	59.43	32.29	60.68	92.97	62.78	112.51	175.29
		EMP [52]	4.01	0	4.01	6.40	0	6.40	10.24	0	10.24	18.18	0	18.18
		FSS [10]	2.85	13495.70	13498.55	3.02	46101.70	46104.72	5.20	-	-	10.01	-	-
		SIGMA [28]	6.84	-	-	9.46	-	-	11.53	-	-	13.11	-	-
		Ours	0.71	116.09	116.80	0.87	266.45	267.32	1.66	701.29	702.95	2.88	2255.94	2258.82
10000	Equality Testing	ABY [20]	74.96	135.39	210.35	173.26	249.03	422.29	295.78	399.57	695.35	601.67	667.26	1265.93
		EMP [52]	21.92	0	21.92	34.11	0	34.11	67.09	0	67.09	112.38	0	112.38
		FSS [10]	14.33	99811.90	99826.23	27.22	316994	317021.22	61.55	-	-	94.26	-	-
		Ours	1.63	151.93	153.56	2.23	279.61	281.84	3.76	511.50	515.26	11.12	942.93	954.05
	Secure Comparison	ABY [20]	88.74	148.40	237.14	181.22	255.58	436.80	289.43	360.90	650.33	624.11	722.94	1347.05
		EMP [52]	29.98	0	29.98	45.01	0	45.01	73.10	0	73.10	121.83	0	121.83
		FSS [10]	11.86	132826	132837.86	22.80	463840	463862.80	45.53	-	-	112.77	-	-
		SIGMA [28]	19.79	-	-	29.50	-	-	81.71	-	-	92.54	-	-
		Ours	3.34	678.52	681.86	6.79	1768.91	1775.70	11.28	5812.32	5823.60	13.06	20969.40	20982.46

lengths and batch sizes. In Appendix B, we provide the corresponding runtime under other network settings. Compared with other equality testing implementations, our protocol realizes multiple performance improvements for the online phase. The communication cost of our protocol is close to FSS [27], while the computation cost of our protocol is more subtle than FSS, leading to a significant performance superiority in LAN and MAN settings. In general, considering appropriate data size, the online phase running time of our equality-testing is (i) over 2× of the garbled circuit, over 7× of the FSS, and over 40× of ABY in the LAN setting; (ii) over 9× of the FSS, over 15× of garble circuit and over 50× of ABY in both MAN and WAN settings. Fig. 17(a) (also, Table 2) depicts the offline running time compared to FSS (with the correlated keys generation) and ABY. Since the offline phase of FSS is almost unable to halt for bit sizes above 32, we compared the performance for bit sizes below 16. Our protocol's offline phase is several orders of magnitude faster than FSS. Although the offline phase of our protocol is slower than other protocols apart from FSS, it achieves significant gains in the online phase.

Secure comparison. Fig. 14 depicts the online phase running time of secure comparison compared to ABY [20], GC [52], DCF [27], SIGMA [28] and CrypTflow2 [47]. Table 2 also presents the runtime for different bit lengths and batch sizes. In most cases, our protocol outperforms other protocols in the online phase. In particular, the efficiency of our protocol is (i) over 3× of the FSS/CrypTflow2/GC, and over 20× of the ABY in the LAN setting; (ii) over 3× of the FSS, over 6× of GC/CrypTflow2 and over 15× of ABY in WAN settings. When the network is worse and the data volume is large enough, our protocol efficiency will be slightly lower than FSS (WAN setting and $> 10^5$ number of comparisons). Fig. 17(b) depicts the offline running time. The offline phase performance of our protocol is 1000× of FSS. As a trade-off, our offline phase is slower than ABY. The communication cost in the online phase of our protocol is reduced by more than 2× compared to the ABY and 10× compared to the EMP. While SIGMA has fewer online communication rounds and a lower communication volume compared to our protocol, its computational workload is significantly higher. Under favorable network conditions—such as LAN or MAN—and with larger data sizes, our protocol outperforms SIGMA. However, in worse network conditions, SIGMA demonstrates better performance. Nevertheless, SIGMA's offline phase requires three parties, meaning it is not a true 2-PC solution.

For more benchmarks, we refer readers to Appendix. B.

6 Conclusion

We propose constant-round equality testing and secure comparison protocols, where each of our protocols enjoys a low communication round and volume in the online phase. Our benchmarks show that the performance of our protocols is several times better than that of SOTA, both in the equality testing and secure comparison.

Acknowledges

This work was supported in part by the National Natural Science Foundation of China (62072401, 62232002, U21A20464, U23A20306, U23A20307, 62261160651, U2436206,62406239), in part by the National Key Research and Development Program of China (2023YFE0111100), in part by Input Output (iohk.io), in part by Ant Group, and in part by Cybersecurity College Student Innovation Funding, in part by China Postdoctoral Science Foundation (No.

2023M742739), in part by the Fundamental Research Funds for the Central Universities (Program No.QTZX24081).

Ethical Implications

The proposed equality testing and secure comparison protocol adheres to ethical standards by ensuring the privacy of input values, as no sensitive data is disclosed during the process. While the protocol enhances privacy and security in data comparisons, we acknowledge the potential for misuse in unethical contexts and recommend its use in secure, regulated environments. Overall, this work contributes to improving privacy-preserving cryptographic protocols, supporting ethical data handling and secure communications.

Compliance with Open Since Policy

According to the open science policy, all experiments were conducted using synthetic, randomly generated data, avoiding the use of any personal or private information. In addition, we will make the source code for our equality testing and secure protocols publicly available following the paper's acceptance and before the camera-ready submission deadline, promoting transparency and supporting further research in this field.

References

[1] Correlated FSS keys generation. `https://github.com/Esion-lin/oram_pro`.

[2] Cryptflow2-code. `https://github.com/mpc-msri/EzPC`.

[3] Emp-toolkit. `https://github.com/emp-toolkit`.

[4] libote. `https://github.com/osu-crypto/libOTe`.

[5] Our code. `https://zenodo.org/records/14580231`.

[6] Carsten Baum, Daniel Escudero, Alberto Pedrouzo-Ulloa, Peter Scholl, and Juan Ramón Troncoso-Pastoriza. Efficient protocols for oblivious linear function evaluation from ring-lwe. *Journal of Computer Security*, 30(1):39–78, 2022.

[7] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

[8] Donald Beaver. Precomputing oblivious transfer. In *Annual International Cryptology Conference*, pages 97–109. Springer, 1995.

[9] Ian F Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 515–529. Springer, 2004.

[10] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *EUROCRYPT*, 2021.

[11] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent ot extension and more. In *CRYPTO*, 2019.

[12] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *CCS*, 2016.

[13] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. Flash: Fast and robust framework for privacy-preserving machine learning. In *PoPETs*, 2020.

[14] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[15] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-psi with linear complexity via relaxed batch opprf. *Proceedings on Privacy Enhancing Technologies*, 2022.

[16] Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-shared shuffle. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part III 26*, pages 342–372. Springer, 2020.

[17] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. Astra: High throughput 3pc over rings with application to secure prediction. In *CCSW*, 2019.

[18] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*, pages 221–256. Springer, 2020.

[19] Geoffroy Couteau. New protocols for secure equality test and comparison. In *International Conference on Applied Cryptography and Network Security*, pages 303–320. Springer, 2018.

[20] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[21] Jack Doerner and Abhi Shelat. Scaling oram for secure computation. In *CCS*, 2017.

[22] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. Two-round oblivious transfer from cdh or lpn. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 768–797. Springer, 2020.

[23] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[24] Philippe Fournier-Viger, Yanjun Yang, Peng Yang, Jerry Chun-Wei Lin, and Unil Yun. Tke: Mining top-k frequent episodes. In *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices: 33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020, Proceedings 33*, pages 832–845. Springer, 2020.

[25] Juan Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *Public Key Cryptography–PKC 2007: 10th International Conference on Practice and Theory in Public-Key Cryptography Beijing, China, April 16-20, 2007. Proceedings 10*, pages 330–342. Springer, 2007.

[26] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

[27] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in cot and dpf. In *EUROCRYPT*, 2023.

[28] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. Sigma: Secure gpt inference with function secret sharing. *Cryptology ePrint Archive*, 2023.

[29] Manoj Kumar Gupta and Pravin Chandra. A comprehensive survey of data mining. *International Journal of Information Technology*, 12(4):1243–1257, 2020.

[30] Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Wen jie Lu, Cheng Hong, and Kui Ren. Ciphergpt: Secure two-party gpt inference. Cryptology ePrint Archive, Paper 2023/1147, 2023.

[31] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 809–826, 2022.

[32] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference*, pages 145–161. Springer, 2003.

[33] Markus Jakobsson and Moti Yung. Proving without knowing: On oblivious, agnostic and blindfolded provers. In *Annual International Cryptology Conference*, pages 186–200. Springer, 1996.

[34] Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. Orca: Fss-based secure training and inference with gpus, 2024.

[35] Bo Jiang, Jian Du, and Qiang Yan. Anonpsi: An anonymity assessment framework for psi. *arXiv preprint arXiv:2311.18118*, 2023.

[36] Florian Kerschbaum, Erik-Oliver Blass, and Rasoul Akhavan Mahdavi. Faster secure comparisons with offline phase for efficient private set intersection. In *NDSS*, 2023.

[37] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology and Network Security: 8th International Conference, CANS 2009, Kanazawa, Japan, December 12-14, 2009. Proceedings 8*, pages 1–20. Springer, 2009.

[38] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35*, pages 486–498. Springer, 2008.

[39] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *CCS*, 2017.

[40] Tianpei Lu, Bingsheng Zhang, Lichun Li, and Kui Ren. Aegis: A lightning fast privacy-preserving machine learning platform against malicious adversaries. Cryptology ePrint Archive, Paper 2023/1890, 2023.

[41] Pieter Moree. Bertrand's postulate for primes in arithmetical progressions. *Computers & Mathematics with Applications*, 26(5):35–43, 1993.

[42] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA*, volume 1, pages 448–457, 2001.

[43] Dimitrios Papakyriakou and Ioannis S Barbounakis. Data mining methods: a review. *International Journal of Computer Application*, 183(48):5–19, 2022.

[44] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. Aby2. 0: Improved mixed-protocol secure two-party computation. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2165–2182, 2021.

[45] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based psi with linear communication. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*, pages 122–153. Springer, 2019.

[46] Srinivasan Raghuraman and Peter Rindal. Blazing fast psi from improved okvs and subfield vole. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2505–2517, New York, NY, USA, 2022. Association for Computing Machinery.

[47] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 325–342, 2020.

[48] Peter Rindal and Phillipp Schoppmann. Vole-psi: Fast oprf and circuit-psi from vector-ole. In *Advances in Cryptology – EUROCRYPT 2021*, pages 901–930, Cham, 2021. Springer International Publishing.

[49] Mike Rosulek and Lawrence Roy. Three halves make a whole? beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO*, 2021.

[50] Abir Smiti. A critical overview of outlier detection methods. *Computer Science Review*, 38:100306, 2020.

[51] Wen-Guey Tzeng. Efficient 1-out-n oblivious transfer schemes. In *Public Key Cryptography: 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002 Paris, France, February 12–14, 2002 Proceedings 5*, pages 159–171. Springer, 2002.

[52] Xiao Wang, Alex J Malozemoff, and Jonathan Katz. Emp-toolkit: Efficient multiparty computation toolkit, 2016.

[53] Andrew Chi-Chih Yao. How to generate and exchange secrets extended abstract. In *27th FOCS*, pages 162–167, 1986.

[54] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *EUROCRYPT*, 2015.

[55] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II 34*, pages 220–250. Springer, 2015.

[56] Lijing Zhou, Ziyu Wang, Hongrui Cui, Qingrui Song, and Yu Yu. Bicoptor: Two-round secure three-party non-linear computation without preprocessing for privacy-preserving machine learning. In *S&P*, 2023.

A Other building block

This section gives other building blocks such as the OLE and the oblivious selective zero check.

A.1 OLE protocol

In the OLE, both parties have no input initially, and then P_0 receives (a,c) and P_1 receives (b,d) such that $ab = c+d$. The OLE can be implemented by invoking p times $\mathcal{F}_{\binom{1}{2}\text{-OT}}$. Specifically, P_1 picks $a \in \mathbb{Z}_p$ and $\{r_j\}_{j\in\mathbb{Z}_p}$, while P_1 picks $b \in \mathbb{Z}_p^*$. Subsequently, For each invoking of $\mathcal{F}_{(1,2)\text{-OT}}$, P_0 sets $m_0 = -r_j$ and $m_1 = a \cdot 2^j - r_j$, and as the sender inputs (m_0, m_1) to $\mathcal{F}_{(1,2)\text{-OT}}$; P_1 as the receiver inputs the chooes bit b_j and receives output z_j. Finally, P_0 computes $c = \sum_{j=1}^{p} r_j$, and P_1 computes $d = \sum_{j=1}^{p} z_j$. Clearly, $c+d = \sum_{j=1}^{p} r_j + \sum_{j=1}^{p} z_j = \sum_{j=1}^{p} a \cdot 2^{b_j} = ab$.

Protocol Π_{ole}^{p}

Input : P_0 and P_1 have no input.
Output : P_0 receives $a \in \mathbb{Z}_p$ and $c \in \mathbb{Z}_p$, while P_1 receives $b \in \mathbb{Z}_p$ and $d \in \mathbb{Z}_p$, where $a \cdot b = c+d$.

Protocol:

- P_0 samples $a \in \mathbb{Z}_p$ and $\{r_j\}_{j\in\mathbb{Z}_p}$.
- P_1 samples $b \in \mathbb{Z}_p^*$.

- For $j \in \mathbb{Z}_p$, P_0 and P_1 invoke $\mathcal{F}_{(1,2)\text{-OT}}$:
  - P_0 inputs $m_0 = -r_j$ and $m_1 = a \cdot 2^j - r_j$.
  - P_1 inputs the chooes bit b_j and receives output z_j.
- P_0 computes $c = \sum_{j=1}^{p} r_j$, P_1 computes $d = \sum_{j=1}^{p} z_j$.

Figure 15: The Oblivious Linear Evaluation Triple Generation Protocol

A.2 Oblivious Selective Zero Check with OLE

We describe the implementation of the oblivious short-list zero check with OLE in Figure 16.

Protocol $\Pi_{\mathsf{ozc}}^{k,n,p}(I,X,Y)$

Input : Index list $I := \{\zeta_i\}_{i\in[k]}$ input by P_0 which contains $k-t$ non-repeating items, and last t indices equal to n; list $X := \{x_i\}_{i\in[n]}$ input by P_0; list $Y := \{y_i\}_{i\in[n]}$ input by P_1;
Output : P_1 receives $z_i = (x_{\zeta_i} + y_{\zeta_i}) \cdot \beta_{\zeta_i}$ for the random value β_{ζ_i} which is unknown to P_1.

Offline:

- P_0 and P_1 invoke n times $\{\beta_i, r_i, [t_i]_0^p, [t_i]_1^p\} \leftarrow \Pi_{\mathsf{ole}}$, where P_0 holds $\{\beta_i, [t_i]_0^p\}$, P_1 holds $\{r_i, [t_i]_1^p\}$.

Execution:

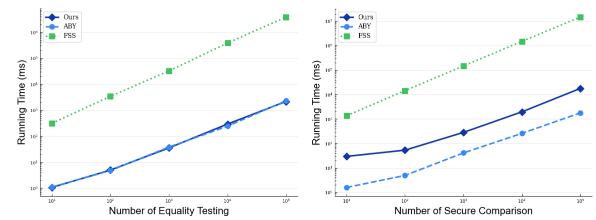
- For $i \in [k]$:
  - P_1 set $y'_j = y_j + r_i$ for $j \in [n]$;
  - P_0 and P_1 invoke $\mathcal{F}_{(1,n)\text{-}\mathsf{OT}}$:
    - P_1 as a sender inputs a set $Y' = \{y'_0, \cdots, y'_{n-1}\}$;
    - P_0 as a receiver inputs select index ζ_i and receives y'_{ζ_i};
  - P_0 calculates $[z_i]_0 = \beta_i \cdot (x_{\zeta_i} + y'_{\zeta_i}) - [t_i]_0^p$;
  - P_1 sets $[z_i]_1 = -[t_i]_1^p$;
  - P_0 and P_1 reveal z_{ζ_i} to P_1;
  - P_1 outputs $c = \mathbf{1}\{0 \in \{z_0, \ldots, z_{k-1}\}\}$.

Figure 16: The Oblivious Selective Zero Check with OLE Protocol.

B Other Benchmarks

In this section, we give more benchmarks. In Table 4, we provide detailed performance reports of the SOTA, including comparisons of different input lengths and batch sizes under various network settings.

Offline Trade-off. Due to the extremely slow offline phase of FSS, it is almost infeasible to halt at 32 bits, so we only evaluated its offline phase at 16 bits. Figure 17 shows the running time (for 16 bits) in the offline phase for the equality testing and comparison protocol compared with ABY [20] and DPF [27] in the LAN setting. The other detailed data for offline is shown in Table 2 and Table 4. The running time of our equality testing in the offline phase is entirely superior to the DPF [27], with performance nearly identical to ABY [20]. Similarly, our secure comparison protocol is also based entirely on the DPF [27]. Although it is slower than ABY [20], the offline performance loss is acceptable for the overall protocol as it achieves a $10\times$ improvement in running time over ABY during the online phase.



(a) Equality testing. (b) Secure comparison.

Figure 17: The running time of offline phase on equality testing protocol Π_{eq_2} and secure comparison protocol Π_{cmp} compare with ABY [20] and DPF [27] in LAN setting.

Benchmarks of Different Input Lengths. Overall, as the input lengths and batch size increase, the performance gains of our protocol improve. As shown in Table 4, the online phase of our protocol is $10\times$ faster than ABY. We observe the following conclusions: due to its fewer online communication rounds, SIGMA outperforms our protocol in certain data scenarios under WAN and MAN settings. Our equality testing protocol performs slightly better than ABY overall, and when the input length is small, our comparison protocol's total time is close to that of ABY. Regardless of the input lengths and batch size, the offline phase of our protocol is significantly faster than that of FSS.

Communication Cost. Table 3 compares the communication costs with batch size 10000 of our protocols to those of ABY [20], EMP [52], and DCF [10]. For equality testing, it is evident that our protocol significantly outperforms the others in the online phase across all input lengths and batch sizes, demonstrating lower communication costs. This advantage is especially noticeable as the batch size increases. For comparison, our protocol demonstrates superior performance in the online phase, particularly as the batch size increases. Despite the offline performance being slower than ABY [20], our approach achieves a better trade-off overall, especially in large batch sizes where communication efficiency is critical.

C Proof of Security

C.1 Proof of Theorem. 1

Proof. To prove Theorem 1, we construct a PPT simulator $\mathcal{S}$, such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{eq}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and the real world $\mathsf{Real}_{\Pi_{\mathsf{eq}_1},\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{(1,2)\text{-}\mathsf{OT}},\mathcal{F}_{(n-1,n)\text{-}\mathsf{OT}}}(1^\lambda)$. We consider the following cases:

Case 1: P_0 is corrupted. We construct the simulator $\mathcal{S}$ which internally runs $\mathcal{A}$, simulates $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$ and $\mathcal{F}_{(n-1,n)\text{-}\mathsf{OT}}$, forwards messages to/from $\mathcal{Z}$, and simulates the interface of the honest party P_1.

Table 3: Communication cost with batch size 10000 of our protocols compared to baselines in MB.

Protocol \ Element Size		8 Online	8 Offline	8 Total	16 Online	16 Offline	16 Total	32 Online	32 Offline	32 Total	64 Online	64 Offline	64 Total
Equality Testing	ABY [20]	3.67	3.36	7.03	7.34	7.02	14.36	14.68	14.34	29.03	29.39	29.00	58.38
	EMP [52]	3.52	0	3.52	7.28	0	7.28	14.82	0	14.82	29.89	0	29.89
	FSS [10]	0.15	46.14	46.22	0.31	87.19	87.27	0.61	-	-	1.22	-	-
	Ours	0.04	2.34	2.38	0.06	4.04	4.10	0.10	7.15	7.25	0.17	13.07	13.23
Secure Comparison	ABY [20]	3.67	3.66	7.34	7.34	7.33	14.67	14.69	14.65	29.34	29.37	29.3	58.67
	EMP [52]	3.52	0	3.52	7.29	0	7.29	14.82	0	14.82	29.89	0	29.89
	FSS [10]	0.15	66.15	66.22	0.31	132.29	132.37	0.61	-	-	1.22	-	-
	Ours	0.19	12.65	12.85	0.36	26.91	27.27	0.71	63.92	64.63	1.39	171.47	172.86

Table 4: Running time of our protocols compared to baselines (given in ms) in the WAN setting.

Batch Size \ Element Size			8 Online	8 Offline	8 Total	16 Online	16 Offline	16 Total	32 Online	32 Offline	32 Total	64 Online	64 Offline	64 Total
100	Equality Testing	ABY [20]	405.0	207.2	612.2	408.8	213.0	621.9	416.2	625.7	1041	599.4	1046	1646
		EMP [52]	630.4	0	630.4	633.1	0	633.1	835.5	0	835.5	1044	0	1044
		FSS [10]	827.3	7.205e5	7.214e5	827.7	1.363e6	1.364e6	828.7	-	-	830.2	-	-
		Ours	401.1	908.1	1309	401.1	912.4	1313	401.2	921.8	1323	401.2	1333	1735
	Secure Comparison	ABY [20]	405.3	207.8	613.2	409.4	213.9	623.3	417.4	625.6	1043.1	600.6	1046	1646
		EMP [52]	630.4	0	630.4	633.2	0	633.2	835.6	0	835.6	1044	0	1044
		FSS [10]	827.4	1.057e6	1.058e6	828.0	2.116e6	2.117e6	828.0	-	-	830.7	-	-
		SIGMA [28]	205.7	-	-	205.7	-	-	213.0	-	-	214.8	-	-
		Ours	401.2	2058	2459	401.3	2278	2679	401.6	2546	2948	402.3	2988	3390
1000	Equality Testing	ABY [20]	613.1	1253	1866	636.5	1492	2129	837.5	2132	2969	1140	3170	4311
		EMP [52]	1051	0	1051	1259	0	1259	1752	0	1752	2532	0	2532
		FSS [10]	829.2	7.015e6	7.016e	834.9	1.342e7	1.342e7	841.9	-	-	845.1	-	-
		Ours	401.3	1542	1943	401.9	1743	2145	402.9	2025	2428	404.7	2704	3109
	Secure Comparison	ABY [20]	441.0	1445	1886	642.1	1499	2141	839.7	2133	2973	1147	3198	4345
		EMP [52]	1052	0	1052	1260	0	1260	1753	0	1753	2532	0	2532
		FSS [10]	829.0	9.975e6	9.976e6	834.0	2.114e7	2.114e7	838.7	-	-	845.0	-	-
		SIGMA [28]	207.4	-	-	207.1	-	-	219.2	-	-	224.9	-	-
		Ours	403.0	3175	3578	405.0	3816	4221	409.1	4914	5323	417.2	8460	8877
10000	Equality Testing	ABY [20]	1495	3425	4920	1631	5160	6791	2273	7015	9289	3731	9402	1.313e4
		EMP [52]	2641	0	2641	3887	0	3887	4580	0	4580	6364	0	6364
		FSS [10]	850.8	7.009e7	7.009e7	858.9	1.337e8	1.337e8	881.708	-	-	925.564	-	-
		Ours	406.1	2997	3403	410.5	3729	4140	416.1	4425	4841	435.2	5693	6128
	Secure Comparison	ABY [20]	1368	3633	5001	1649	5212	6861	2210	7133	9343	4305	9478	1.378e4
		EMP [52]	2645	0	2645	3889	0	3889	4543	0	4543	6350	0	6350
		FSS [10]	846.2	1.019e8	1.019e8	857.5	2.109e8	2.109e8	878.6	-	-	921.5	-	-
		SIGMA [28]	226.9	-	-	241.1	-	-	284.0	-	-	549.0	-	-
		Ours	422.8	5521	5943	441.8	8984	9426	480.0	1.831e4	1.879e4	729.1	4.633e4	4.706e4

Upon receiving $(\mathsf{Input}, \mathsf{sid}, P_1)$ from $\mathcal{F}_{\mathsf{eq}}$, $\mathcal{S}$ does as follows.

- For the simulation of the i^{th} times of $\Pi^{2\to p}_{\mathsf{convert}}$, where $i \in [n]$,
  - When corrupted P_0 inputs $(m_{0,i}, m_{1,i})$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$, $\mathcal{S}$ records $(m_{0,i}, m_{1,i})$.
  - $\mathcal{S}$ computes $[s_i]^p_0$ and $[t_i]^p_0$ with $m_{0,i}, m_{1,i}$.
  - $\mathcal{S}$ picks $[w_i]^2_1 \in \{0,1\}$ and acts as P_1 to send it to P_0.
  - Upon receiving $[w_i]^2_0$ from P_0, $\mathcal{S}$ computes $w_i = [w_i]^2_0 \oplus [w_i]^2_1$ and $s'_i = w_i + [t_i]^p_0 - 2w_i[t_i]^p_0$.
- $\mathcal{S}$ computes $d_0 = \sum_{i=0}^{n-1} s'_i$.
- For the simulation of Π_{eq_1},
  - When P_1 invokes $\mathcal{F}_{(2n-1,2n)\text{-}\mathsf{ROT}}$, $\mathcal{S}$ selects $m_i \in \{0,1\}^{2n}$ for $i \in [2n-1]$ and computes $v_i = \bigoplus_{j=0}^{2n-2} m_{(i,j)}$. Upon receiving $(\mathsf{Output}, \mathsf{sid}, [e]_0)$ from $\mathcal{F}_{\mathsf{eq}}$, if each bit of v_i is equal, $\mathcal{S}$ samples m_{2n-1} such that each bit of it is not all equal to $v_i \oplus [e]_0 \oplus 1$. Then, $\mathcal{S}$ emulates $\mathcal{F}_{(2n-1,2n)\text{-}\mathsf{ROT}}$ and forwards m_i for $i \in [2n]$ to P_0. Note that $2^{\log p} = 2^{\log n+1} = 2n$.
  - $\mathcal{S}$ generates the binary matrix $\mathbf{M}$ by using the $\{m_i\}_{i\in[2n]}$ as the binary column vectors, and perform a right circular shift on the i^{th} row of $\mathbf{M}$ by an offset of i locally for $i \in [2n]$.
  - $\mathcal{S}$ computes $[t_i]_0 = \bigoplus_{j=0}^{2n-1} m_{(i,j)}$ to generate $\vec{T}_0$ and computes $u_i = \bigoplus_{j=0}^{2n-1} m_{(j,i)}$ to generate $\vec{U}$.
  - Upon receiving $\vec{S}'$ from P_0, $\mathcal{S}$ computes $\vec{T}' = \vec{S}' \oplus \vec{U}$ to extract ε_0, where only the $\varepsilon_0^{\text{th}}$ element of $\vec{T}'$ is equal to 1.
  - $\mathcal{S}$ picks a random index ρ satisfying $[t_\rho]_0 = [e]_0$.
  - $\mathcal{S}$ computes $w_1 = \rho - (\varepsilon_0 + d_0)$ and acts as P_1 to send it to P_0.

$\mathsf{Indistinguishability}$. We show that the incoming message and the output of P_0 in the ideal world are indistinguishable from the real world.

Claim 1. *The ideal world* $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{eq}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ *and the real world* $\mathsf{Real}^{\mathcal{F}_{(1,2)\text{-}\mathsf{OT}},\mathcal{F}_{(n-1,n)\text{-}\mathsf{OT}}}_{\Pi_{\mathsf{eq}_1},\mathcal{A},\mathcal{Z}}(1^\lambda)$ *are perfectly indistinguishable.*

Proof. There are three parts of incoming messages that are different between $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{eq}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and $\mathsf{Real}^{\mathcal{F}_{(1,2)\text{-}\mathsf{OT}},\mathcal{F}_{(n-1,n)\text{-}\mathsf{OT}}}_{\Pi_{\mathsf{eq}_1},\mathcal{A},\mathcal{Z}}(1^\lambda)$.

- In the ideal world, $[w_i]_1^2$ are calculated with dummy b' rather than real b.
- The output of $\mathcal{F}_{(2n-1,2n)\text{-ROT}}$ to P_0 is indistinguishabe between the ideal world and the real world.
- In the ideal world, $w_1 = \rho - (\varepsilon_0 + d_0)$ rather than $\varepsilon_1 + d_1$.

For the first part, due to randomly picked $[r_i]_1^2$, $[r_i]_1^2 \oplus b_i$ are uniformly random whether b_i is input by P_1 in the real world or picked random in the ideal world. For the third part, $w_1 = \rho - (\varepsilon_0 + d_0)$ in the ideal world, where ρ is a random index. In the real world, $w_1 = \varepsilon_1 + d_1$, where ε_1 is a random index. Therefore, w_1 is uniformly random in both the ideal world and the real world. □

Case 2: P_1 is corrupted. We construct the simulator $\mathcal{S}$ which internally runs $\mathcal{A}$, simulates $\mathcal{F}_{(1,2)\text{-OT}}$ and $\mathcal{F}_{(N-1,N)\text{-OT}}$, forwards messages to/from $\mathcal{Z}$, and simulates the interface of the honest party P_0.

Upon receiving $(\mathsf{Input}, \mathsf{sid}, P_0)$ from $\mathcal{F}_{\mathsf{eq}}$, $\mathcal{S}$ does as follows.

- For the simulation of the i^{th} times of $\Pi^{2\to p}_{\mathsf{convert}}$, where $i \in [n]$,
  - $\mathcal{S}$ picks random $[r_i]_0^2 \in \{0,1\}$ and $[s_i]_0^p \in \mathbb{Z}_p$, and emulates $\mathcal{F}_{(1,2)\text{-OT}}$ with the inputs $m_0 = [s_i]_0^p - [r_i]_0^2$ and $m_1 = [s_i]_0^p - (1 - [r_i]_0^2)$.
  - When a corrupted P_1 inputs $[r_i]_1{}^2$ to $\mathcal{F}_{(1,2)\text{-OT}}$, $\mathcal{S}$ records $[r_i]_1{}^2$, sends $[s]_1^p = m_{[r_i]_1^2}$ to P_1, and denotes $-[s]_1^p$ as $[t]_1^p$.
  - $\mathcal{S}$ picks $[w_i]_0^2 \in \{0,1\}$ and acts as P_0 to send it to P_1.
  - Upon receiving $[w_i]_1^2$ from P_1, $\mathcal{S}$ computes $w_i = [w_i]_0^2 \oplus [w_i]_1^2$ and $t'_i = [t]_1^p - 2w_i[t]_1^p$.
- $\mathcal{S}$ calculate $d_1 = \sum_{i=0}^{n-1} t_i'$.
- For the simulation of Π_{eq_1},
  - When P_0 invokes $\mathcal{F}_{(2n-1,2n)\text{-ROT}}$, $\mathcal{S}$ selects ε_1 and $m_i \in \{0,1\}^{2n}$ for $i \in [2n\backslash\{\varepsilon_1\}]$ and forwards them to P_1. Note that $2^{\lceil \log p \rceil} = 2^{\log n+1} = 2n$.
  - $\mathcal{S}$ generates the binary matrix $\mathbf{M}$ by using the set $\{m_i\}_{i\in[2n]\backslash\varepsilon_1}$ as the binary column vectors, omitting the $\varepsilon_1{}^{\text{th}}$ column. Then, for each $i \in [2n]$, $\mathcal{S}$ performs a right circular shift on the i^{th} row by an offset of i.
  - For $i \in [2n]$, $\mathcal{S}$ computes the XOR of all elements in the i^{th} row and the $(\varepsilon_1 + i)^{\text{th}}$ column of the binary matrix $\mathbf{M}$ to generate w_i.
  - Upon receiving $(\mathsf{Output}, [e]_1)$ from $\mathcal{F}_{\mathsf{eq}}$, $\mathcal{S}$ samples $\vec{T}_1 \in \mathbb{Z}_2^{2n}$ such that each bit of it is not all equal to $[e]_1 \oplus 1$, and then computes $\vec{S'} = \vec{T}_1 \oplus \vec{W}$. $\mathcal{S}$ acts as P_0 to send it to P_1.
  - $\mathcal{S}$ picks a random ρ satisfying $[t_\rho]_1 = [e]_1$.
  - $\mathcal{S}$ computes $w_0 = \rho - (\varepsilon_1 + d_1)$ and acts as P_0 to send it to P_1.

Indistinguishability. We show that the incoming message and the output of P_1 in the ideal world are indistinguishable from the real world.

Claim 2. *The ideal world* $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{eq}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ *and the real world* $\mathsf{Real}^{\mathcal{F}_{(1,2)\text{-OT}},\mathcal{F}_{(n-1,n)\text{-OT}}}_{\Pi_{\mathsf{eq}_1},\mathcal{A},\mathcal{Z}}(1^\lambda)$ *are perfectly indistinguishable.*

Proof. There are five parts of incoming messages that are different between $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{eq}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and $\mathsf{Real}^{\mathcal{F}_{(1,2)\text{-OT}},\mathcal{F}_{(n-1,n)\text{-OT}}}_{\Pi_{\mathsf{eq}_1},\mathcal{A},\mathcal{Z}}(1^\lambda)$.

- $[s]_1^p$ are generated from the random $[s]_0^p$, which is indistinguishable between the ideal world and the real world.
- In the ideal world, $[w_i]_0^2$ are calculated with dummy a' rather than real a.
- The output of $\mathcal{F}_{(2n-1,2n)\text{-ROT}}$ to P_1 is indistinguishable between the ideal world and the real world.
- In the ideal world, $\vec{S'}$ is a random vector rather than $\vec{T'} \oplus \vec{U}$.
- In the ideal world, $w_0 = \rho - (\varepsilon_1 + d_1)$ rather than $\varepsilon_0 + d_0$.

For the second part, due to randomly picked $[r_i]_0^2$, $[r_i]_0^2 \oplus a_i$ are uniformly random whether a_i is input by P_0 in the real world or picked random in the ideal world. For the fourth part, $\vec{S'} = \vec{T'} \oplus \vec{U}$ in the real world, where the $\vec{U}$ is a uniformly random vector. Therefore, $\vec{S'}$ is uniformly random in both the ideal world and the real world. For the fifth part, $w_0 = \rho - (\varepsilon_1 + d_1)$ in the ideal world, where ρ is a random index. In the real world, $w_0 = \varepsilon_0 + d_0$, where ε_0 is a random index. Therefore, w_1 is uniformly random in both the ideal and real worlds. □

This concludes the proof. □

C.2 Proof of Theorem. 2

Proof. To prove Thm. 2, we construct a PPT simulator $\mathcal{S}$, such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{cmp}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and the real world $\mathsf{Real}^{\mathcal{F}_{\mathsf{ozc}},\mathcal{F}_{(1,2)\text{-OT}}}_{\Pi_{\mathsf{cmp}},\mathcal{A},\mathcal{Z}}(1^\lambda)$. We consider the following cases:

Case 1: P_0 is corrupted. We construct the simulator $\mathcal{S}$ which internally runs $\mathcal{A}$ and simulates $\mathcal{F}_{\mathsf{ozc}}$ and $\mathcal{F}_{(1,2)\text{-OT}}$, forwarding messages to/from $\mathcal{Z}$ and simulates the interface of honest P_1.

Upon receiving $(\mathsf{Input}, \mathsf{sid}, P_1)$ from $\mathcal{F}_{\mathsf{cmp}}$, $\mathcal{S}$ acts as P_1 does as follows.

- For the simulation of i^{th} times of $\Pi^{2\to p}_{\mathsf{convert}}$, $i \in [n]$,

– $\mathcal{S}$ picks random $[r_i]_1^2 \in \mathbb{Z}_2$ and input it to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$;

– When corrupted P_0 inputs $(m_{0,i}, m_{1,i})$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$, $\mathcal{S}$ records $(m_{0,i}, m_{1,i})$; $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$ outputs $[s]_1^p = m_{[r_i]_1^2}$ to $\mathcal{S}$.

– $\mathcal{S}$ calculate r_i and s_i with $m_{0,i}, m_{1,i}$;

– $\mathcal{S}$ picks $[w_i]_1^2 \in \mathbb{Z}_2$ and acts as P_1 to send it to P_0.

– Upon receiving $[w_i]_0^2$ from P_0, $\mathcal{S}$ calculate $a_i = [w_i]_0^2 \oplus [r_i]_0^2$

- $\mathcal{S}$ sends $(\mathsf{Input}, \mathsf{sid}, a)$ to external $\mathcal{F}_{\mathsf{cmp}}$.
- $\mathcal{S}$ picks random list $\{s_i\}_{i \in \mathbb{Z}_{n+1}} \leftarrow (\mathbb{Z}_p)^{n+1}$ and input to internal $\mathcal{F}_{\mathsf{ozc}}$.
- When P_0 input I to $\mathcal{F}_{\mathsf{ozc}}$, $\mathcal{S}$ records I and calculates $\Delta := a_i$ for $i \in I \wedge i \neq n+1$.
- If $a = 0$ or $a = 2^n - 1$ and $I := \{n+1, \ldots, n+1\}$, set $\Delta = 1 \oplus a_0$.
- $\mathcal{S}$ sends $(\mathsf{Modify}, \mathsf{sid}, \Delta)$ to external $\mathcal{F}_{\mathsf{cmp}}$.

Indistinguishability. We show that the incoming message and the output of P_0 in the ideal world are indistinguishable from the real world.

Claim 3. *The ideal world* $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{cmp}}, \mathcal{S}, \mathcal{Z}}(1^\lambda)$ *and the real world* $\mathsf{Real}^{\mathcal{F}_{\mathsf{ozc}}, \mathcal{F}_{(1,2)\text{-}\mathsf{OT}}}_{\Pi_{\mathsf{cmp}}, \mathcal{A}, \mathcal{Z}}(1^\lambda)$ *are perfectly indistinguishable.*

Proof. Observe that P_0 locally set $[c]_0 = \Delta$, so that the output of ideal execution keeps consistent with the real execution. In addition, there are two parts of incoming messages that are different between $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{cmp}}, \mathcal{S}, \mathcal{Z}}(1^\lambda)$ and $\mathsf{Real}^{\mathcal{F}_{\mathsf{ozc}}, \mathcal{F}_{(1,2)\text{-}\mathsf{OT}}}_{\Pi_{\mathsf{cmp}}, \mathcal{A}, \mathcal{Z}}(1^\lambda)$.

- In the ideal world, $[w_i]_1^2$ for each invoking of $\Pi^{2 \to p}_{\mathsf{convert}}$ are picked random rather than calculated by $[b]_i^2 \oplus [r]_i^2$;
- The output of $\mathcal{F}_{\mathsf{ozc}}$ is indistinguishable between the ideal world and the real world.

For the first part, $[r]_i^2$ is uniformly random for P_0 such that $[b]_i^2 \oplus [r]_i^2$ is also uniformly random. Obviously, $[w_i]_1^2$ and $[b]_i^2 \oplus [r]_i^2$ is indistinguishable. For the second part, $\mathcal{F}_{\mathsf{ozc}}$ only outputs a message to P_1 so that corrupted P_0 receives no message. □

Case 2: P_1 is corrupted. We construct the simulator $\mathcal{S}$ which internally runs $\mathcal{A}$ and simulates $\mathcal{F}_{\mathsf{ozc}}$ and $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$, forwarding messages to/from $\mathcal{Z}$ and simulates the interface of honest P_0.

Upon receiving $(\mathsf{Input}, \mathsf{sid}, P_0)$ from $\mathcal{F}_{\mathsf{cmp}}$, $\mathcal{S}$ does as follows.

- Picks $a \leftarrow \mathbb{Z}_{2^n}$;
- For the simulation of i^{th} times of $\Pi^{2 \to p}_{\mathsf{convert}}$, $i \in [n]$,

  – $\mathcal{S}$ picks random $[r_i]_0^2 \in \mathbb{Z}_2$, $[s_i]_0^p \in \mathbb{Z}_p$ and inputs $m_0 = [s_i]_0^p - [r_i]_0^2$, $m_1 = [s_i]_0^p - (1 - [r_i]_0^2)$ to internal $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$;

  – When corrupted P_1 inputs $[r_i]_1^2$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$, $\mathcal{S}$ records $[r_i]_1^2$ and $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$ sends $m_{[r_i]_1^2}$ to P_1;

  – When P_0 receives $[w_i]_1^2$ from P_1, $\mathcal{S}$ calculates $b_i = [w_i]_1^2 \oplus [r_i]_1^2$;

  – $\mathcal{S}$ calculates $[w_i]_0^2 = [r_i]_0^2 \oplus a_i$ and acts as P_1 to send it to P_0;

- $\mathcal{S}$ calculates $\{s_i\}_{i \in [n+1]}$ with $\phi((a||0) \oplus (b||1))$. There exists $\rho \in [n+1]$ such that $s_\rho = 0$.
- $\mathcal{S}$ sends $(\mathsf{Input}, \mathsf{sid}, b)$ to external $\mathcal{F}_{\mathsf{cmp}}$.
- Upon receiving $(\mathsf{Output}, \mathsf{sid}, [c]_1^2)$ from $\mathcal{F}_{\mathsf{cmp}}$, $\mathcal{S}$ does:

  – if $[c]_1^2 = 1$, set $I' := \{n+1, n+1, \ldots, n+1, n+1\}$ with $n+1$ dimension.

  – if $[c]_1^2 = 0$, set $I' := \{n+1, n+1, \ldots, n+1, \rho\}$ with $n+1$ dimension.

- When corrupted P_1 inputs $\{[s_i]_1\}_{i \in [n+2]}$ to $\mathcal{F}_{\mathsf{ozc}}$, $\mathcal{S}$ calculates $[s_i]_0 = s_i - [s_i]_1$ for $i \in [n+2]$.
- $\mathcal{S}$ inputs $\{[s_i]_0\}_{i \in [n+2]}$ and selection list I to $\mathcal{F}_{\mathsf{ozc}}$.

Indistinguishability. We show that the incoming message and the output of P_0 in the ideal world are indistinguishable from the real world.

Claim 4. *The ideal world* $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{cmp}}, \mathcal{S}, \mathcal{Z}}(1^\lambda)$ *and the real world* $\mathsf{Real}^{\mathcal{F}_{\mathsf{ozc}}, \mathcal{F}_{(1,2)\text{-}\mathsf{OT}}}_{\Pi_{\mathsf{cmp}}, \mathcal{A}, \mathcal{Z}}(1^\lambda)$ *are perfectly indistinguishable.*

Proof. We first show that in the ideal world, P_1 reveives same output as the real world: If $I' := \{n+1, n+1, \ldots, n+1, n+1\}$, $\mathcal{F}_{\mathsf{ozc}}$ will output $z = 0$ to P_1 induce P_1 to output $[c]_1^2 = 1 \oplus z = 1$. On the contrary, if $I' := \{n+1, n+1, \ldots, n+1, \rho\}$, $\mathcal{F}_{\mathsf{ozc}}$ will output 1 to P_1 such that P_1 output $[c]_1^2 = 0$.

In addition, there are two parts of incoming messages that are different between $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{cmp}}, \mathcal{S}, \mathcal{Z}}(1^\lambda)$ and $\mathsf{Real}^{\mathcal{F}_{\mathsf{ozc}}, \mathcal{F}_{(1,2)\text{-}\mathsf{OT}}}_{\Pi_{\mathsf{cmp}}, \mathcal{A}, \mathcal{Z}}(1^\lambda)$.

- In the ideal world, $[w_i]_1^2$ are calculated with dummy a rather than real a;
- The output of $\mathcal{F}_{\mathsf{ozc}}$ is indistinguishable between the ideal world and the real world.

For the first part, due to randomly picked $[r_i]_0^2$, $[r_i]_0^2 \oplus a_i$ are uniformly random whether a_i is input by P_0 in the real world or picked random in the ideal world; for the second part, we have proven the output of $\mathcal{F}_{\mathsf{ozc}}$ is consistent between the ideal world and the real world. □

This concludes the proof. □

C.3 Proof of Theorem. 3

Proof. To prove Thm. 3, we construct a PPT simulator $\mathcal{S}$, such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{ozc}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and the real world $\mathsf{Real}_{\Pi_{\mathsf{ozc}},\mathcal{A},\mathcal{Z}}(1^\lambda)$. We consider the following cases:

Case 1: P_0 is corrupted. We construct the simulator $\mathcal{S}$ which internally runs $\mathcal{A}$ and simulates $\mathcal{F}_{\mathsf{ole}}$, $\mathcal{F}_{\mathsf{vole}}$ and $\mathcal{F}_{\mathsf{permute}}$ internal, forwarding messages to/from $\mathcal{Z}$ and simulates the interface of honest P_1.

Upon receiving $(\mathsf{Input},\mathsf{sid},P_1)$ from $\mathcal{F}_{\mathsf{cmp}}$, $\mathcal{S}$ does as follows.

- When P_0 invokes $\mathcal{F}_{\mathsf{ole}}$, $\mathcal{F}_{\mathsf{ole}}$ outputs (β_i,r_i,u_i,v_i) for $i\in[n]$ to $\mathcal{S}$; $\mathcal{S}$ forwards (β_i,u_i) to corrupted P_0.
- When P_0 invokes $\mathcal{F}_{\mathsf{vole}}$, $\mathcal{F}_{\mathsf{vole}}$ outputs (β_j,r,u_j,v_j) for $j\in[k-1]$ to $\mathcal{S}$; $\mathcal{S}$ forwards (β_j,u_j) to corrupted P_0.
- When P_0 sends permutation π to $\mathcal{F}_{\mathsf{permute}}$, $\mathcal{S}$ sends $\{v_i\}_{i\in[n+k-1]}$ to $\mathcal{F}_{\mathsf{permute}}$;
- Upon receiving $\{[v_{\pi(i)}]\}_{i\in[n+k-1]}$ from $\mathcal{F}_{\mathsf{permute}}$; $\mathcal{S}$ forwards $\{[v_{\pi(i)}]_0\}_{i\in[n+k-1]}$ to corrupted P_0.
- $\mathcal{S}$ picks random list $\{y'_i\}_{i\in[n]}\leftarrow(\mathbb{Z}_p)^n$ and acts as P_1 to send it to P_0.
- Upon receiving $\{[d_i]_0\}_{i\in[k]}$ and $\{s_i\}_{i\in[k]}$ from P_0, $\mathcal{S}$ does
  - calculate $\zeta_i=\pi(s_i)$ for $i\in[k]$.
  - calculate $x_{\zeta_i}=\beta^-_{\zeta_i}([d_i]_0+[w_{\pi^-(\zeta_i)}])$
  - set $x_j\leftarrow\mathbb{Z}_p$ for $j\in[n]\backslash\{\zeta_i\}_{i\in[k]}$
  - send $(\mathsf{Input},\mathsf{sid},\{\zeta_i\}_{i\in[k]},\{x_j\}_{j\in[n]})$ to $\mathcal{F}_{\mathsf{ozc}}$.

Indistinguishability. We show that the incoming message P_0 and the output in the ideal world are indistinguishable from the real world.

Claim 5. *the ideal world* $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{ozc}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ *and the real world* $\mathsf{Real}^{\mathcal{F}_{\mathsf{ole}},\mathcal{F}_{\mathsf{vole}},\mathcal{F}_{\mathsf{permute}}}_{\Pi_{\mathsf{ozc}},\mathcal{A},\mathcal{Z}}(1^\lambda)$ *are perfectly indistinguishable.*

Proof. Observe that input $((\mathsf{Input},\mathsf{sid},\{\zeta_i\}_{i\in[k]},\{x_j\}_{j\in[n]}))$ of $\mathcal{F}_{\mathsf{ozc}}$ in the ideal world is the same as the real world, which makes the output of P_1 equals to the real world execution. In addition, the incoming message corrupted P_0 received different from the real world is $\{y'_i\}_{i\in[k]}$. In the ideal world, $\{y'_i\}_{i\in[k]}$ is picked random rather than mask random r_i to input y_i. Since r_i is a random secret for P_0, y_i+r_i is uniformly random for P_0 and is indistinguishable from y'_i.

□

Case 2: P_1 is corrupted. We construct the simulator $\mathcal{S}$ which internally runs $\mathcal{A}$ and simulates $\mathcal{F}_{\mathsf{ole}}$, $\mathcal{F}_{\mathsf{vole}}$ and $\mathcal{F}_{\mathsf{permute}}$ internal, forwarding messages to/from $\mathcal{Z}$ and simulates the interface of honest P_0.

Upon receiving $(\mathsf{Input},\mathsf{sid},P_0)$ from $\mathcal{F}_{\mathsf{ozc}}$, $\mathcal{S}$ acts as P_0 do as follows.

- When P_1 invokes $\mathcal{F}_{\mathsf{ole}}$, $\mathcal{F}_{\mathsf{ole}}$ outputs (β_i,r_i,u_i,v_i) for $i\in[n]$ to $\mathcal{S}$; $\mathcal{S}$ forwards (r_i,v_i) to corrupted P_1;
- When P_1 invokes $\mathcal{F}_{\mathsf{vole}}$, $\mathcal{F}_{\mathsf{vole}}$ outputs (β_j,r,u_j,v_j) for $j\in[k-1]$ to $\mathcal{S}$; $\mathcal{S}$ forwards (r,v_j) to corrupted P_1;
- When P_1 sends $\{v_i\}_{i\in[k+n-1]}$ to $\mathcal{F}_{\mathsf{permute}}$, $\mathcal{S}$ picks permutation π and sends it to $\mathcal{F}_{\mathsf{permute}}$; $\mathcal{S}$ records $\{v_i\}_{i\in[k+n-1]}$;
- Upon receiving $\{[v_{\pi(i)}]\}_{i\in[n+k-1]}$ from $\mathcal{F}_{\mathsf{permute}}$, $\mathcal{S}$ forwards $\{[v_{\pi(i)}]_1\}_{i\in[n+k-1]}$ to corrupted P_1;
- When P_0 receiving $\{y'_i\}_{i\in[n]}$ from P_1, $\mathcal{S}$ does,
  - calculate $y_i=y'_i-r_i$ for $i\in[n]$.
  - send $(\mathsf{Input},\mathsf{sid},\{y_i\}_{i\in[n]})$ to $\mathcal{F}_{\mathsf{ozc}}$.
- Upon receiving $(\mathsf{Output},\mathsf{sid},z)$ from $\mathcal{F}_{\mathsf{ozc}}$, $\mathcal{S}$ does,
  - pick random list $\{r_0,\dots,r_{k-1}\}\in(\mathbb{Z}^*_p)^k$.
  - pick random set $I':=\{s_0,\dots,s_{k-1}\}\in\mathbb{Z}^k_{n+k-1}$.
  - for $i\in[k]$, set $[d_i]_0=[w_{s_i}]_1+r_i$;
  - if $z=1$, pick $\eta\leftarrow I'$ and set $[d_\eta]_0=[d_\eta]_0-r_\eta$.
  - act as P_0 to send $\{[d_i]_0\}_{i\in[k]}$ and I_1 to P_1.

Indistinguishability. We show that the incoming message of P_1 and the output in the ideal world are indistinguishable from the real world.

Claim 6. *the ideal world* $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{ozc}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ *and the real world* $\mathsf{Real}^{\mathcal{F}_{\mathsf{ole}},\mathcal{F}_{\mathsf{vole}},\mathcal{F}_{\mathsf{permute}}}_{\Pi_{\mathsf{ozc}},\mathcal{A},\mathcal{Z}}(1^\lambda)$ *are perfectly indistinguishable.*

Proof. Obliviously, the input of $\mathcal{F}_{\mathsf{ozc}}$ is the same as the real world. When P_1 evaluates $d_i=[d_i]_0-[w_{s_i}]_1$, considering the output of $\mathcal{F}_{\mathsf{ozc}}$, if $z=1$, the item $d_\eta=0$ leads to P_1 output 1; otherwise, all items $d_i=r_i\in\mathbb{Z}^*_p$ are positive, leading to P_1 output 0. Therefore, the output is indistinguishable from the real world. In addition, for the incoming message of P_1, index set $I:=\{s_i\}_{i\in[k]}$ and $\{[d_i]_0\}_{i\in[k]}$ are generated through

- pick random list $\{r_0,\dots,r_{k-1}\}\in(\mathbb{Z}^*_p)^k$.
- pick random set $I':=\{s_0,\dots,s_{k-1}\}\in\mathbb{Z}^k_{n+k-1}$.
- for $i\in[k]$, set $[d_i]_0=[w_{s_i}]_1+r_i$;
- if $z=1$, pick $\eta\leftarrow I'$ and set $[d_\eta]_0=[d_\eta]_0-r_\eta$.

rather than,

- $[d_i]_0=\beta_{\zeta_i}\cdot(x_{\zeta_i}+y'_{\zeta_i})-[w_{\pi^-(\zeta_i)}]_0$ for $i\in[k-t]$;
- $s_i=\pi^-(\zeta_i)$ for $i\in[k-t]$;
- $[d_i]_0=\beta_{n+i-k}\cdot(x_n+y'_n)-[w_{\pi^-(n+i-k)}]_0$ for $i\in[k-t,k]$;
- $s_i=\pi^-(n+i-k)$ for $i\in[k-t,k]$;

Since π is a random permutation, set $\{s_i = \pi^-(\zeta_i)\}_{i\in[k]}$ is uniformly random in the distribution $\mathbb{Z}_{n+k-1}^k$, which is indistinguishabe from the ideal world. For the list $\{[d_i]_0\}_{i\in[k]}$, considering $z = 0$, in the real world, each item d_i is positive, it indicates that the distribution of $[d_i]_0 = d_i + [w_{s_i}]_1$ is determined by sum of random value in $\mathbb{Z}_p^*$ and random value in $\mathbb{Z}_p$ which is same as the ideal world. $\square$

This concludes the proof. $\square$