Efficient 2PC for Constant Round Secure Equality Testing and Comparison

Tianpei Lu*‡, Xin Kang†‡, Bingsheng Zhang*§, Zhuo Ma†§, Xiaoyuan Zhang*, Yang Liu† and Kui Ren*

*The State Key Laboratory of Blockchain and Data Security, Zhejiang University, China

†Xidian University, China

lutianpei@zju.edu.cn, kangxin@stu.xidian.edu.cn,bingsheng@zju.edu.cn, mazhuo@mail.xidian.edu.cn, zhangxiaoyuan@zju.edu.cn, bcds2018@foxmail.com, kuiren@zju.edu.cn

Abstract

Secure equality testing and comparison are two important primitives that have been widely used in many secure computation scenarios, such as privacy-preserving machine learning, private set intersection, secure data mining, etc. In this work, we propose new constant-round two-party computation (2PC) protocols for secure equality testing and secure comparison. Our protocols are designed in the online/offline paradigm. Theoretically, for 32-bit integers, the online communication for our equality testing is only 76 bits, and the cost for our secure comparison is only 384 bits. Our benchmarks show that (i) our equality is 9× faster than the Guo et al. (EUROCRYPT 2023) and 15× of the garbled circuit scheme (EMP-toolkit). (ii) our secure comparison protocol is 3× faster than Guo et al. (EUROCRYPT 2023), 6× faster than both Rathee et al. (CCS 2020) and garbled circuit scheme.

‡Tianpei Lu and Xin Kang contributed equally to this work.

§Corresponding author: Bingsheng Zhang and Zhuo Ma.

Contents

I. Introduction

Secure multiparty computation (MPC) [7], [25], [49] enables multiple untrusted parties to perform joint computations without revealing their private data. In the early stages, general-purpose MPC protocols [25], [29], [49] were widely studied and significantly improved in performance. Recently, researchers have focused on specific functions that benefit from tailor-made protocol designs and achieve performance far beyond general-purpose implementations [37], [42], [52]. The secure comparison and equality testing problem as typical cases have been considered, in which the parties joint calculate $a > b$ or $a = b$ for the private input a, b without disclosing them. Secure comparison and equality testing enjoy numerous applications as fundamental building blocks for various primitives in privacy computing, such as federated learning, privacy-preserving machine learning, advertising bidding systems, biometric authentication, and so on. 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 [12], [16], especially for non-linear functions such as ReLU, MaxPool, and so on [52]. Furthermore, a series of works [29], [31], [36] evaluate arbitrary functions, including Sigmod, GeLU, softmax, etc., through piecewise function fitting based on comparison and polynomial evaluation.
- Private set intersection. Private Set Intersection (PSI) [14], [32], [44] 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 PSI instantiations, equality testing accounts for more than 50% of the total communication cost of the protocol [41]. Therefore, optimizing the communication cost of equality testing is of great importance for PSI.
- Secure Data Mining. Secure data mining [27], [39] can facilitate the identification of the most relevant items or patterns without exposing raw data. In secure data mining, secure comparison is used in data mining tasks such as identifying the *top-k* items [23], outlier detection [46], and other analytics, where comparisons are necessary to draw insights from distributed datasets without compromising data privacy. Therefore, optimizing the performance and efficiency of secure comparison can drive the development and application of data mining technologies.

In general, the performance improvement of secure comparison can benefit a wide range of MPC applications. A series of secure comparison protocols [37], [52] in multi-party scenarios have achieved significant improvements resulting in efficient secure comparison/equality testing. Nevertheless, in two-party computation (2PC) scenarios, secure comparison and equality testing remain major performance bottlenecks in practice. The state-of-the-art (SOTA) [14], [29], [40] unanimously lead to the conclusion that secure 2PC comparison/equality-testing is magnitude slower than other secure operations, e.g., secure multiplication.

A sequence of efforts [19], [40] has been made to optimize the communication of comparison or equality testing. In contrast, in these works, the benefit of communication volume inevitably makes sacrifices on the communication rounds, i.e., logarithmic rounds, suffered a poor performance in the high network delay. The other approaches focus on the constant-round protocols. The typical solutions are based on the garbled circuit [45], [50] scheme or the function secret sharing (FSS) [10], [11] scheme. The garbled circuit scheme requires massive communication and computation in the circuit evaluation (in the online phase), leading to inferior practical performance to the protocols with logarithmic rounds. FSS gains prior online communication compared to the garbled circuit scheme. In contrast, its online computation cost is close to 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. Migrating to the 2PC, the correlated keys should be performed under MPC (Equivalent to running massive PRGs under MPC), which is completely beyond the practical. Recently, a line of works [20], [26] design correlated keys generation protocols that move the PRGs evaluation to the local. As a trade-off, its computation cost is exponential to the data size n. Considering a large n, it is even impossible to output the result.

So far as we know, there doesn't exist a practical constant round secure comparison or equality testing protocol while holding very efficient online phase performance with a practical offline phase.

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 the inputs. We design low (constant) communication rounds protocols with relatively efficient offline communication volume, ultimately improving overall performance. We show that our protocols are secure against passive adversarial in the universal composability framework of Canetti [13].

<u>2-round Equality testing.</u> We first propose a dimension reduction scheme for a and b, reducing them to a' and b' such that $a' = b'$ if and only if $a = b$. The length of a' and b' is $\log n$. Subsequently, we construct a look-up table with linear communication. Both parties sample random numbers ε_0 and ε_1 and then share a look-up table $\vec{T}$ such that only the $(\varepsilon_0 + \varepsilon_1)^{\mathsf{th}}$ value is 1, this is $t_{\varepsilon_0+\varepsilon_1} = 1$, and all other values are 0 in the offline phase. The parties obtain the result of the equality testing by locally selecting the sharing of $t_{\varepsilon_0+\varepsilon_1+a'-b'}$ in the online phase.

<u>Secure comparison.</u> We propose a novel 3-round secure comparison protocol Π_{cmp} in the semi-honest setting. Intuitively, we start from the high bit and check the first different bit of a and b, where the value of a on the position of different bit corresponds to the result of the comparison. We construct a secret shared list $\{s\}_n$ that highlights such a position. Consequently, we let P_0 guess the comparison result Δ, and open all the possible positions of such comparison result, namely, the position ζ_i for $a_{\zeta_i} = \Delta$, to P_1. P_1 checks if there exists ζ_i, such that s_{ζ_i} is highlighted. If P_0 guess wrong (no highlighted s_{ζ_i}), P_1 set output $z_1 = 1$ to flap Δ. Otherwise P_1 outputs $z_1 = 0$.

<u>Performance.</u> Table I depicts the communication comparison between our protocols and SOTA 2PC solutions.

Our equality testing protocol requires 2 rounds of $O(n)$ bits communication in the online phase, which is close to function secret sharing scheme [20], [26], where our computation cost is much slighter than FSS (without invoking any PRF) leading to a faster online phase, i.e. the running time of FSS is over 7×

more than ours, in LAN/MAN/WAN setting. Moreover, compared to FSS, the offline of our protocol is magnitude efficient, i.e. over 1000× faster than it. For the other baseline – garbled circuit [19], [40], [49], [50], its online phase communication is 200× higher than ours. Specifically, our benchmark shows that in the MAN setting, our protocol achieves 15× prior performance.

Our secure comparison protocol requires 3 round of $2n + 2n \log n$ bits communication in the online phase. Similar to equality testing, our protocol outperforms the SOTA protocols. Compared to FSS [26], our protocol achieves over 3× online phase performance improvement, and over 1000× in the offline phase. Compared to the SOTA comparison CrypTflow2 [43], our protocol achieves over 6× improvement in both MAN and WAN settings.

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

A. Related work

The concept of secure comparison was first proposed by Yao [49], a.k.a, millionaire's problem. Subsequently, equality testing called socialist millionaires' problem [30] 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 [49]. Kolesnikov et al. [35] proposed a protocol for constructing universal circuits almost exclusively composed of XOR gates, which relies on the random oracle (RO) assumption. Then, they [34] optimize the assumption by allowing one party to garble circuits containing comparison gates, achieving secure comparison through AND gates. Zahur et al. [51] 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. [24] 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. [17] 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 [18] 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 [43], 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. [14] extends the idea to equality testing. Huang et al. [29] further optimized communication cost in CrypTflow2 [43] by replacing the OT with VOLE-type OT.

FSS-based-CMP/EQ. Function secret sharing (FSS) [10], [11] 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) [11] 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 [20], [26] 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. [19] 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. [40] 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.

II. Preliminaries

Notation. The frequently used notations are shown in Table II. 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 of shifting the column vector $\vec{X}$ down by i positions. In additional, 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 denote the matrix M as $\mathbf{M}$, and the element in the i^{th} row and j^{th} column of $\mathbf{M}$ 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 based on the Universal Composability (UC) framework [13], which follows the simulation-based security paradigm. In

TABLE I: 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 [48], [49]	-	$4\lambda n$	2
Generic Two-Party Computation	ABY [19]	$6\lambda n + n$	$2\lambda n + 6n$	3
	ABY2.0 [40]	$5\lambda n + 2n$	$\lambda n + 6n$	3
FSS-based-EQ	Half-Tree [26]	$(n+2)\lambda^{+}$	$2n$	1
	DPF [11]	$4n(\lambda+1) + \lambda + n^{\dagger}$	$2n$	1
OT-based-EQ	CO [18]	$3\lambda n$	$2n + 2\log n + 10$	$\log^{*} n + 1$
	CGS [14]	$\frac{3}{4}\lambda n + 8n$	$5n - 4$	$\log n + 4$
	Π_{eq_2} (Section III)	$\lambda \log n + 2n$	$2n + 2\log n + 2$	2
Secure Comparison				
GC-based-CMP	Yao [48], [49]	-	$4\lambda n$	2
HE-based-CMP	GSV [24]	-	$18\mu n + 8\mu$	9
Generic Two-Party Computation	ABY [19]	$6\lambda n + 17\lambda + n$	$2\lambda n + 20n$	3
	ABY2.0 [40]	$5\lambda n + 17\lambda + 2n$	$\lambda n + 9n$	3
FSS-based-CMP	Half-Tree [26]	$(n+2)\lambda^{+}$	$2n$	1
	DCF [10]	$4n(\lambda+1) + \lambda + n^{\dagger}$	$2n$	1
OT-based-CMP	CO [18]	$6\lambda n$	$6n + 4\log n$	$4\log^{*}\lambda + 5$
	Cryptflow2 [43]	$\lambda n + 16n$	$10n - 8$	$\log n + 4$
	Cheetah [29]	$\lambda n + 11n$	$10n - 8$	$\log n + 4$
	Π_{cmp}(Section IV)	$n\lambda \log n + n \log n + n$	$2n + 2n \log n$	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.

the UC framework, protocols are executed across multiple interconnected 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 $\mathcal{A}$

TABLE II: Notations

Notations	Descriptions
$\vec{A}$	The vector $\vec{A} := \{a_0, \ldots, a_{n-1}\}$.
a_i	The i^{th} element of vector $\vec{A}$, when the context is clear, we abuse a_i as the i^{th} bit of value a.
$[n]$	The index set $\{0, \ldots, n-1\}$.
$[1, n]$	The index set $\{1 \ldots, n-1\}$.
$([\cdot]_0^p, [\cdot]_1^p)$	The algorithm shares over $\mathbb{Z}_p$ owned by P_0 and P_1.
$\mathbf{1}\{b\}$	The indicator function, which evaluates to 1 when b is true and 0 when b is false.
(m, n)-OT	m-out-of-n OT.
$\mathsf{shift}(\vec{X}, i)$	Shift the column vector $\vec{X}$ down by i positions.
$\mathbf{M}$	The matrix M.
$m_{(i,j)}$	The element in the i^{th} row and j^{th} column of the matrix $\mathbf{M}$.

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 $\mathcal{Z}$.

Functionality $\mathcal{F}_{\mathsf{2pc}}^f$

$\mathcal{F}_{\mathsf{2pc}}^f$ 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.

Fig. 1: The Ideal Functionality $\mathcal{F}_{\mathsf{2pc}}^f$.

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 $\mathcal{F}_{\mathsf{2pc}}^f$ with the excuted function f. Both parties send their share to $\mathcal{F}_{\mathsf{2pc}}^f$, and $\mathcal{F}_{\mathsf{2pc}}^f$ 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:

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

Oblivious transfer. For an instance of $(1,2)$-OT [21], [22], the sender's inputs to the $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$ are the strings m_0 and $m_1 \in \{0,1\}^l$, and the receiver's input is a bit $i \in \{0,1\}$. The receiver obtains m_i from the $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$ and the sender receives no output. Random OT (ROT) [8] is a special case of OT where there is no input. 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. The $\mathcal{F}_{(n-1,n)\text{-}\mathsf{ROT}}$ [15] makes the sender obtains $\{m_0, \ldots, m_n\}$, while the receiver obtains a element $b \in [n]$ and $\{m_i\}$ for $i \in [n] \backslash \{b\}$. ROT is input-independent, which can be executed in the offline phase and OT can be constructed with linear communication in the online phase. The $(1,n)$-OT [38], [47] is a generalization of $(1,2)$-OT. The sender's inputs to the $\mathcal{F}_{(1,n)\text{-}\mathsf{OT}}$ are the n strings $\{m_0, \cdots, m_{n-1}\}$, and the receiver's input is a choose number $i \in [n]$. The receiver obtains m_i from the $\mathcal{F}_{(1,n)\text{-}\mathsf{OT}}$ and the sender receives no output. The $(1,n)$-OT can be constructed via $(1,2)$-OT.

Oblivious Linear Evaluation. Oblivious Linear Evaluation (OLE) [6], [33] is a foundational component in various secure computation protocols [28], [42], [44]. In the standard OLE protocol [6], P_0 receives random values a and b, while P_1 receives a random value u and $w = au + b$. A symmetric variant of OLE [33], known as product sharing [6], involves P_0 and P_1 each sampling a and b, respectively. After the protocol, P_0 obtains u and P_1 obtains v, satisfying the condition $ab = u + v$. This set of random values (a, b, u, v) constitutes an OLE tuple. In vector OLE (VOLE) [44], the parties have no input. P_0 obtains a random value u and a random vector $\vec{B}$. P_1 obtains a random vector $\vec{A}$ and the vector $\vec{V} = \vec{A}u + \vec{B}$, where $v_i = a_i u + b_i$.

Secure permutation.The secure permutation [15] 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 [15] currently is constructed by OT. We define the functionality $\mathcal{F}_{\mathsf{Permute}}$ as follow: the P_0 inputs a permutation π, and P_1 inputs a list $\vec{X} := \{x_0, \cdots, x_{n-1}\}$. After the protocol, they obtain the secret shares of the permuted list $\{x_{\pi(0)}, \cdots, x_{\pi(n-1)}\}$.

III. Equality Testing

In the equality testing, P_0 inputs an integer $a \in \mathbb{Z}_2^n$ and P_1 inputs an integer $b \in \mathbb{Z}_2^n$. After the protocol, both parties receive boolean shares of $\mathbf{1}\{a = b\}$, which equals 1 if and only if $a = b$, and 0 otherwise.

In this section, we first design a toy protocol for equality testing with one round of communication in the online phase. However, this design results in an $\mathcal{O}(2^n)$ communication complexity in the offline phase. To further balance the communication cost between the online and offline phases, we propose a dimension reduction scheme to optimize the toy protocol. This optimization allows for two rounds of communication in the online phase while ensuring that the $\mathcal{O}(n)$ communication complexity in the offline phase.

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 $\vec{T}_0 \oplus \vec{T}_1 = \mathsf{shift}(\vec{T}', \varepsilon_1)$.

Execution

1) P_0 and P_1 invoke $\mathcal{F}_{(N-1,N)\text{-ROT}}$:
   - 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) P_0 and P_1 left cycle shift the i^{th} row of $\mathbf{M}$ by i positions locally for $i \in [N]$.
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 $w_i = v_i \oplus u_{\varepsilon_1 + i}$, 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 $\vec{T}_1 := \mathsf{shift}(S', \varepsilon_1) \oplus \vec{W}$.

Fig. 2: The Vector Oblivious Shift Evaluation Protocol.

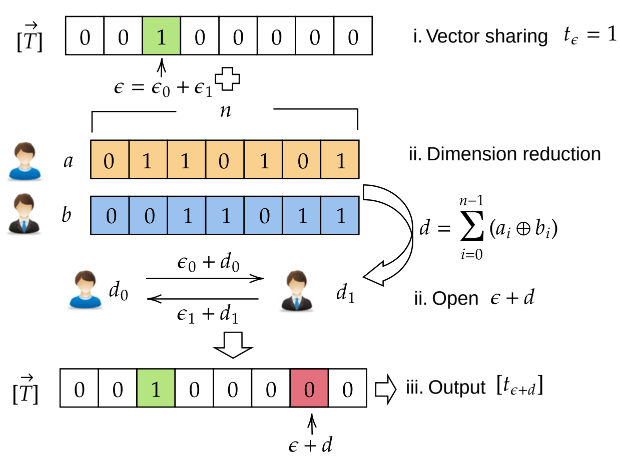


Fig. 3: The Overview of Equality Testing

A. One-round equality testing

We first describe a strawman example for the equality testing, then we design a one-round equality testing protocol.

A strawman example. A strawman example for equality testing is that 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 select t_b. Clearly, $t_b = 1$ if and only if $a = b$. Note that to enumerate all strings of length

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 $[e]^2_0$ and P_1 receives $[e]^2_1$, where $[e]^2_0 \oplus [e]^2_1 = \mathbf{1}\{a = b\}$.

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 $[e]^2_i = [t_w]_i$.

Fig. 4: One-Round Equality Testing.

n, the size of $\vec{T}$ is 2^n. For convenience, we define $N = 2^n$. However, the current design doesn't guarantee the privacy of the result of the equality testing, as the result is public to P_1. Keeping t_b secret, 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 P_1 privately select t'_b instead of t_b. Obviously, $s \oplus t'_b = 1$ if and only if $a = b$. Nevertheless, the process of P_1 privately selecting t'_b requires an instance of $\mathcal{F}_{(1,N)\text{-}\mathsf{OT}}$, which involves two rounds of communication and incurs an $\mathcal{O}(2^n)$ communication complexity in the online phase.

One-round equality testing. Our goal is to design an equality testing protocol that achieves one round of communication and $\mathcal{O}(n)$ communication complexity in the online phase. The protocol is described in Figure 4, and the overview is shown as follows. In the offline phase, P_0 and P_1 respectively pick offsets ε_0 and ε_1, 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 other words, P_0 picks an offset ε_0 and generate a binary vector $\vec{T'}$ with only $t'_{\varepsilon_0} = 1$. As the result, 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 $[t_i]_0 \oplus [t_i]_1 = t_i$. In the online phase, P_0 and P_1 reveal the value $w = \varepsilon_0 + \varepsilon_1 + a - b$, and then select $[t_w]_0$ 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 reveal w locally. Clearly, $[t_w]_0 \oplus [t_w]_1 = 1$ if and only if $a = b$.

We construct our offline phase based on a primitive – Vector Oblivious Shift Evaluation (VOSE). Before introducing VOSE, we propose the random VOSE.

Random Vector Oblivious Shift Evaluation. In the random VOSE, P_0 receives two random binary vectors $\vec{U}$ and $\vec{V}$, while 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 random VOSE can be built from $\mathcal{F}_{(N-1,N)\text{-}\mathsf{ROT}}$, Specifically, we describe the process as follows and show the overview in

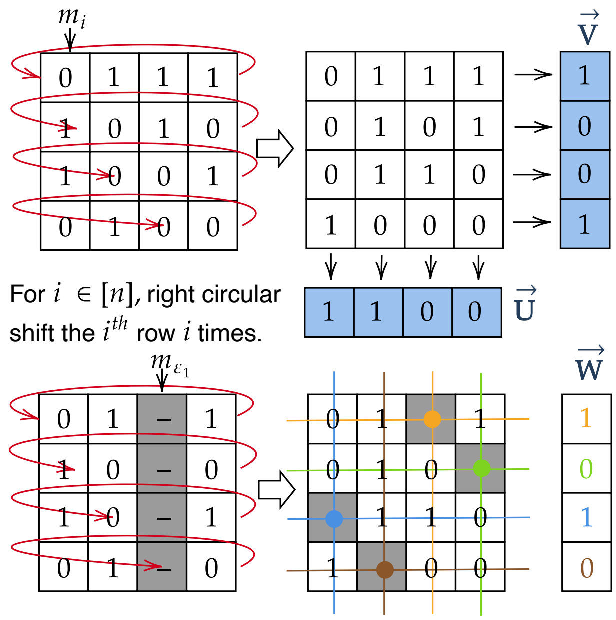


Fig. 5: The Overview of the Random VOSE.

Figure 5.

- P_0 and P_1 invoke an instance of $\mathcal{F}_{(N-1,N)\text{-}\mathsf{ROT}}$. After the protocol, P_0 receives N messages $\{m_0, \ldots, m_{N-1}\}$ and $m_i \in \mathbb{Z}_2^N$. 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 the complete $\mathbf{M}$, while P_1 can obtain the $\mathbf{M}$ except for the $\varepsilon_1{}^{\text{th}}$ column.
- P_0 and P_1 right circular shift the i^{th} row of $\mathbf{M}$ by i times for $i \in [N]$, and denote the new matrix as $\mathbf{M}'$.
- 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 $i \in [N]$ 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.
- P_1 computes $w_i = v_i \oplus u_{\varepsilon_1 + i \mod N}$ to generate $\vec{W} = \{w_0, \ldots, w_{N-1}\}$.

Through steps 2 and 3, 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 $\vec{W} = \mathsf{shift}(\vec{U}, \varepsilon_1) \oplus \vec{V}$.

<u>Vector Oblivious Shift Evaluation.</u> Based on the random VOSE, we elaborate on the implementation of VOSE in the following three steps. The protocol is shown in Figure 2.

- P_0 and P_1 invoke the random VOSE. After the protocol, P_0 receives $\vec{U} \in \mathbb{Z}_2^N$ and $\vec{V} \in \mathbb{Z}_2^N$, while P_1 receives the offset ε_1 and a vector $\vec{W} \in \mathbb{Z}_2^N$, 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 $\vec{T_1} = \mathsf{shift}(S', \varepsilon_1) \oplus \vec{W}$.

For the 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 $\vec{T_0} = \vec{V}$. As a result, we have $\vec{T_0} \oplus \vec{T_1} = \vec{T}$.

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

Input : P_0 inputs $[u]^2_0$; P_1 inputs $[u]^2_1$.

Output : P_0 receives $[v]^p_0$ and P_1 receives $[v]^p_1$, where $[v]^p_0 + [v]^p_1 = [u]^2_0 \oplus [u]^2_1$.

Offline:

1) P_0 samples $[r]^2_0$, and P_1 samples $[r]^2_1$.
2) P_0 and P_1 invoke $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$:
   - P_0 samples $[s]^p_0$.
   - P_0 as the sender inputs $m_0 = [s]^p_0 - [r]^2_0$ and $m_1 = [s]^p_0 - (1 - [r]^2_0)$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$;
   - P_1 as the receiver inputs $[r]^2_1$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$, and then receives $[s]^p_1 := m_{[r]^2_1}$.
3) P_0 sets $[t]^p_0 = [s]^p_0$ and P_1 set $[t]^p_1 = -[s]^p_1$.

Online:

1) For $i \in \{0,1\}$, P_i computes $[w]^2_i = [u]^2_i \oplus [r]^2_i$ and sends $[w]^2_i$ to P_{1-i}.
2) P_0 and P_1 computes $w = [w]^2_0 \oplus [w]^2_1$, locally.
3) P_0 computes $[v]^p_0 = w + [t]^p_0 - 2w[t]^p_0$, and P_1 computes $[v]^p_1 = [t]^p_1 - 2w[t]^p_1$.

Fig. 6: The Share Conversion Protocol.

Efficiency. In the offline phase, P_0 and P_1 invoke one time of $\mathcal{F}_{(N-1,N)\text{-}\mathsf{ROT}}$ and P_0 send $\vec{S'} \in \mathbb{Z}_2^N$. Therefore, the communication cost is $n\lambda + N$. In the online phase, the communication cost is $2n$ bits.

B. Two-round equality testing

We observe that, in the aforementioned protocol, the communication in the offline phase is closed to N bits, namely, 2^n, which is impractical in real-world applications. the overview is shown in Figure 3 To reduce the communication cost in the offline phase, we introduce a dimension reduction protocol that can diminish the overall communication, i.e. $O(n)$ bits communication. The overview of the protocol is shown in Figure 3.
Dimension reduction. The dimension reduction protocol is designed to reduce the integers $(a \in \mathbb{Z}_2^n, b \in \mathbb{Z}_2^n)$ to $(a' \in \mathbb{Z}_2^{\log n}, b' \in \mathbb{Z}_2^{\log n})$ such that $a' = b'$ if and only if $a = b$. The start point of generating a' and b' is that, $d = \sum_{i=0}^{n-1}(a_i \oplus b_i) = 0$ if and only if $a = b$. Notice that the entropy-bit of d is $\lceil \log n + 1 \rceil$. Thus, the arithmetic sharing of d, denoted as $[d]_0$ and $[d]_1$ where $d = [d]_0 + [d]_1$, are the expectional a' and b' such that $a' = [d]_0$ and $b' = -[d]_1$. The correctness is that $a' - b' = [d]_0 + [d]_1 = d$.

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 $[u]^2_0$ and $[u]^2_1$. After the protocol, they receive the arithmetic sharing $[v]^p_0$ and $[v]^p_1$, satisfying $[v]^p_0 + [v]^p_1 = [u]^2_0 \oplus [u]^2_1$. Here, $p > n$ is a prime.

An instance of sharing conversion can be easily constructed based on the $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$. In particular, P_0

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 $[e]^2_0 \oplus [e]^2_1 = \mathbf{1}\{a=b\}$.

Execution:

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 $([e]^2_0, [e]^2_0) \leftarrow \Pi^p_{\mathsf{eq}_1}([d]_0, -[d]_1)$.

Fig. 7: Two-Round Equality Testing.

samples $[s]^p_0$, and inputs $m_0 = [s]^p_0 - [u]^2_0$ and $m_1 = [s]^p_0 - (1 - [u]^2_0)$. P_1 inputs the selection bit $[u]^2_1$ and receives z, where $z = [s]^p_0 - ([u]^2_0 \oplus [u]^2_1)$. Consequently, P_0 sets $[v]^p_0 = [s]^p_0$ and P_1 sets $[v]^p_0 = -z$. For correctness, we have $[v]^p_0 + [v]^p_1 = [s]^p_0 - z = [s]^p_0 - [s]^p_0 - ([u]^2_0 \oplus [u]^2_1) = [u]^2_0 \oplus [u]^2_1$ as required. However, all computations are currently performed in the online phase, resulting in the communication complexity is $\mathcal{O}(n^2)$ and the round is 2.

Optimization of share conversion. We attempt to shift a significant portion of expensive operations to the offline phase, resulting in only a small amount of computation and communication in the online phase. The specific description is illustrated in Figure 6. In the offline phase, P_0 and P_1 generate a random sharing conversion pair, i.e. P_0 receives $([r]^2_0, [t]^p_0)$ and P_1 receives $([r]^2_1, [t]^p_1)$, such that $[t]^p_0 + [t]^p_1 = [r]^2_0 \oplus [r]^2_1$. In the online phase, P_0 computes $[w]^2_0 = [a]^2_0 \oplus [r]^2_0$ and sends it to P_1, while P_1 computes $[w]^2_1 = [a]^2_1 \oplus [r]^2_1$ and sneds it to P_0. Subsequently, P_0 and P_1 compute the public 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.

Protocol description. By filling in detailed descriptions, we complete our protocol, which is described in Figure 7. Next, we will explain our protocol step by step as follows.

- At step 1, P_0 and P_1 invoke n times of Π_{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 $[e] \leftarrow \Pi^p_{\mathsf{eq}_1}([d]_0, -[d]_1)$. Then, they output $[e]$ as the shared result of $\mathbf{1}\{a=b\}$.

Efficiency. In the offline phase, P_0 and P_1 invoke n times of $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$ and one times of $\mathcal{F}_{(p-1,p)\text{-}\mathsf{ROT}}$. In addition, P_0 send $\vec{S'} \in \mathbb{Z}^p_2$. The corresponding communication cost is $\lambda \log p + 2p + \lambda = \lambda\lceil \log(n+1)\rceil + 2n + 3$ bits. In the online phase, P_0 and P_1 send n bits to each other in the share conversion Π_{convert}, and send $p = \lceil \log(n+1) \rceil$ 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 $\mathcal{F}_{\mathsf{2PC}}$, where $\mathcal{F}_{\mathsf{eq}}$ receives a from honest P_0 or $\mathcal{S}$ and b from honest P_1 or $\mathcal{S}$, calculates $[e]_0^2 \oplus [e]_1^2 = \mathbf{1}\{a = b\}$ and sends $[e]_0^2$ to P_0 and $[e]_1^2$ to P_1. 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 $\{\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}, \mathcal{F}_{(1,N)\text{-}\mathsf{OT}}\}$-hybrid model against semi-honest PPT adversaries with statical corruption.

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 $\mathsf{Real}_{\Pi_{\mathsf{eq}_1},\mathcal{A},\mathcal{Z}}(1^\lambda)$. We consider the following cases:

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

- Upon receiving $(\mathsf{Input}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{eq}}$, $\mathcal{S}$ starts simulation.
- For the simulation of i^{th} times of Π_{convert}, $i \in [n]$,
  - Upon receiving $(\mathsf{Input}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{eq}}$, $\mathcal{S}$ starts simulation.
  - $\mathcal{S}$ picks random $[r_i]^2 \in \mathbb{Z}_2$ and emulates $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$ with input $[r_i]^2$;
  - When corrupted P_0 inputs $(m_{0,i}, m_{1,i})$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$, $\mathcal{S}$ records $(m_{0,i}, m_{1,i})$.
  - $\mathcal{S}$ calculates s_i and t_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 $w_i = [w_i]_0^2 \oplus [w_i]_1^2$ and $v_i = w_i + t_i - 2wt_i$.
- $\mathcal{S}$ calculate $d_0 = \sum_{i=0}^{n-1} v_i$.
- For the simulation of Π_{eq_1},
  - $\mathcal{S}$ emulates $\mathcal{F}_{(N-1,N)\text{-}\mathsf{ROT}}$ and forward the output $m_i \in \mathbb{Z}_2^p$ for $i \in [p]$ to P_0.
  - $\mathcal{S}$ generate the binary matrix $\mathbf{M}$ by using the $\{m_i\}_{i\in[p]}$ as the binary column vectors, and left cycle shift the i^{th} row of $\mathbf{M}$ by i positions locally for $i \in [p]$.
  - $\mathcal{S}$ computes $v_i = \bigoplus_{j=0}^{p-1} m_{(i,j)}$ to generate $\vec{T}_0$, such that $[t_i]_0 = v_i$; $\mathcal{S}$ computes $u_i = \bigoplus_{j=0}^{p-1} m_{(j,i)}$ to generate $\vec{U}$.
  - Upon receiving $\vec{S'}$ from P_0, $\mathcal{S}$ picks $w_1 \in \mathbb{Z}_p$ and acts as P_1 to send it to P_0. In addition, $\mathcal{S}$ computes $\vec{T'} = \vec{S'} \oplus \vec{U}$ with only $t'_{\varepsilon_0} = 1$ and extract ε_0.
  - Upon receiving $(\mathsf{Output}, [e]_0)$ from $\mathcal{F}_{\mathsf{eq}}$, $\mathcal{S}$ pick a random index ρ satisfying $[t_\rho]_0 = [e]_0$.
  - $\mathcal{S}$ computes $w_1 = \rho - (\varepsilon_0 + d_0)$ and acts as P_0 to send it to P_1.
  - Upon receiving $\vec{S'}$ from P_0, $\mathcal{S}$ picks $w_1 \in \mathbb{Z}_p$ and acts as P_1 to send it to P_0.

Claim 1. If $\mathsf{PRF}^{\mathbb{Z}_2}$, $\mathsf{PRF}^{\mathbb{Z}_p}$ and $\mathsf{PRF}^{\mathbb{Z}_2^p}$ are the secure pseudorandom functions with adversarial advantage $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2}}(1^\lambda, \mathcal{A})$, $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\lambda, \mathcal{A})$ and $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2^p}}(1^\lambda, \mathcal{A})$, then the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{eq}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and the real world $\mathsf{Real}_{\Pi_{\mathsf{eq}_2},\mathcal{A},\mathcal{Z}}(1^\lambda)$ are indistinguishable with advantage $\epsilon = n \cdot (\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2}}(1^\lambda, \mathcal{A}) + \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\lambda, \mathcal{A})) + p \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2^p}}(1^\lambda, \mathcal{A})$.

Proof. In the ideal world for simulating Π_{convert}, $[w_i]_1^2$ are picked random rather than calculated by $b_i \oplus [r_i]_1^2$. Therefore, the advantage in invoking Π_{convert} is $\epsilon_0 = n \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2}}(1^\lambda, \mathcal{A})$. In the ideal world for simulating Π_{eq_1}, m_i for $i \in [p]$ are the output of $\mathcal{F}_{(N-1,N)\text{-}\mathsf{ROT}}$. In addition, $w_1 = \rho - (\varepsilon_0 + d_0)$ are picked random rather than calculated by $\varepsilon_1 - d_1$, where the ρ is random. Therefore, the advantage in invoking Π_{eq_1} is $\epsilon_1 = \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\lambda, \mathcal{A}) + p \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2^p}}(1^\lambda, \mathcal{A})$. Therefore, the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{eq}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and the real world $\mathsf{Real}_{\Pi_{\mathsf{eq}_2},\mathcal{A},\mathcal{Z}}(1^\lambda)$ are indistinguishable with advantage $\epsilon = \epsilon_0 + \epsilon_1$. □

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

- Upon receiving $(\mathsf{Input}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{eq}}$, $\mathcal{S}$ starts simulation.
- For the simulation of i^{th} times of Π_{convert}, $i \in [n]$,
  - $\mathcal{S}$ picks random $[r_i]^2 \in \mathbb{Z}_2$, $[s_i]^p \in \mathbb{Z}_p$ and emulates $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$ with input $m_0 = [s_i]_0^p - [r_i]_0^2$, $m_1 = [s_i]_0^p - (1 - [r_i]_0^2)$;
  - 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 sends $m_{[r_i]_1^2}$ to P_1.
  - $\mathcal{S}$ calculates t_i with $m_{[r_i]_1^2}$.
  - $\mathcal{S}$ picks $[w_i]_0^2 \in \mathbb{Z}_2$ and acts as P_0 to send it to P_1.
  - Upon receiving $[w_i]_1^2$ from P_1, $\mathcal{S}$ calculates $w_i = [w_i]_0^2 \oplus [w_i]_1^2$ and $v_i = t_i - 2wt_i$.
- $\mathcal{S}$ calculate $d_1 = \sum_{i=0}^{n-1} v_i$.
- For the simulation of Π_{eq_1},
  - $\mathcal{S}$ picks $\varepsilon_1 \in \mathbb{Z}_p$ and $m_i \in \mathbb{Z}_2^p$ for $i \in [p] \backslash \{\varepsilon_1\}$, and acts as $\mathcal{F}_{(N-1,N)\text{-}\mathsf{ROT}}$ to send them to P_1.
  - $\mathcal{S}$ generate the binary matrix $\mathbf{M}$ by using the $\{m_i\}_{i \in [p] \backslash \{\varepsilon_1\}}$ as the binary column vectors, and left cycle shift the i^{th} row of $\mathbf{M}$ by i positions locally for $i \in [p]$.
  - $\mathcal{S}$ computes $w_i = v_i \oplus u_{\varepsilon_1+i}$ to generate $\vec{W}$, where $v_i = (\bigoplus_{j=0}^{\varepsilon_1-2} m_{(i,j)}) \oplus (\bigoplus_{j=\varepsilon_1}^{p_1} m_{(i,j)})$ and $u_i = (\bigoplus_{j=0}^{\varepsilon_1-2} m_{(j,i)}) \oplus (\bigoplus_{j=\varepsilon_1}^{p-1} m_{(j,i)})$.
  - $\mathcal{S}$ picks $\vec{S'} \in \mathbb{Z}_2^p$ and acts as P_0 to send it to P_1.
  - $\mathcal{S}$ computes $\vec{T}_1 := \mathsf{shift}(\vec{S'}, \varepsilon_1) \oplus \vec{W}$.
  - Upon receiving $(\mathsf{Output}, [e]_1)$ from $\mathcal{F}_{\mathsf{eq}}$, $\mathcal{S}$ pick 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.

Claim 2. If $\mathsf{PRF}^{\mathbb{Z}_2}$, $\mathsf{PRF}^{\mathbb{Z}_p}$ and $\mathsf{PRF}^{\mathbb{Z}_2^p}$ are the secure pseudorandom functions with adversarial advantage $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2}}(1^\lambda, \mathcal{A}), \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\lambda, \mathcal{A})$ and $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2^p}}(1^\lambda, \mathcal{A})$, then the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{eq}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and the real world $\mathsf{Real}_{\Pi_{\mathsf{eq}_2},\mathcal{A},\mathcal{Z}}(1^\lambda)$ are indistinguishable with advantage $\epsilon = n \cdot (\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2}}(1^\lambda, \mathcal{A})) + (n+1) \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\lambda, \mathcal{A}) + \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2^p}}(1^\lambda, \mathcal{A})$.

Proof. In the ideal world for simulating Π_{convert}, $[s_i]_1^p$ is picked random rather than calculated by $[s_i]_0^p - ([r_i]_0^2 \oplus [r_i]_1^2)$. Therefore, the advantage in this step is $\epsilon_2 = n \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\lambda, \mathcal{A})$ and the ϵ_0 is the same as in case 1. In additional, in the ideal world for simulating Π_{eq_1}, the offset $\varepsilon_1 \in \mathbb{Z}_p$ and m_i for $i \in [p] \backslash \{\varepsilon_1\}$ are picked random rather than the output of $\mathcal{F}_{(N-1,N)\text{-}\mathsf{ROT}}$. w_0 are picked random rather than calculated

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

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

Input:

- Upon receiving $(\mathsf{Input},\mathsf{sid},\mathcal{I},X)$ from $P_0 \in \mathcal{P}$, record $(\mathcal{I},X)$ and send $(\mathsf{Input},\mathsf{sid},P_0)$ to $\mathcal{S}$, where
  - $X := \{x_0,\cdots,x_{n-1}\} \in \mathbb{Z}_p^n$;
  - $\mathcal{I} \in \mathbb{Z}_n^k$;
- 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 $\mathcal{I}, X$ and Y are recorded, $\mathcal{F}_{\mathsf{ozc}}$ does:
  - set $b = 1$ if $\exists i \in \mathcal{I}, x_{\zeta_i} + y_{\zeta_i} = 0$.
  - set $b = 0$ otherwise.
- Send $(\mathsf{Output},\mathsf{sid},b)$ to P_1.

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

by $d_0 + \varepsilon_0$. Therefore, the advantage in this step is $\epsilon_3 = (n-1)\cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2^p}}(1^\lambda,\mathcal{A}) + 2\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\lambda,\mathcal{A})$. In conclusion, the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{eq}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and the real world $\mathsf{Real}_{\Pi_{\mathsf{eq}_2},\mathcal{A},\mathcal{Z}}(1^\lambda)$ are indistinguishable with advantage $\epsilon = \epsilon_0 + \epsilon_2 + \epsilon_3$. $\square$

This concludes the proof. $\square$

IV. Secure Comparison

In this section, we propose a novel secure comparison protocol where P_0 inputs a and P_1 inputs b, receiving the shared result $\mathbf{1}\{a > b\}$. We first give an overview of our protocol which is constructed by a new primitive – oblivious short-list zero check (OZC). Then we propose a two-round OZC protocol as the build block.

A. 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 is no different bits between a and b, we append 1 to the end of b and 0 to the end of a (In contrast, we append 1 to a and 0 to b for $\mathbf{1}\{a \geq b\}$). 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. At first step, we view a and b as the bitwise-XOR share of m. Namely, $m_i = a_i \oplus b_i$ for $i \in [n]$. The position ρ corresponds to the first non-zero bit of m. We introduce a transformation [37]

$\{s_i\}_{i\in[n]} = \phi(\{m_i\}_{i\in[n]})$ where the result list $\{s_i\}$ only contains an unique zero-value in the position ρ and non-zero value in other position.

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 $i = \rho$. For instance, if $a = 10010$ and $b = 10101$, we have $m = a \oplus b = 00111$, and then $s' = 00123$ and $s = 11012$. Analogously, it holds that $s_i \leq n$ (The maximum s_i takes n when $s'_{n-1} = n-1$ and $m_i = 0$). To avoid extra 0 caused by wrapping round, ϕ should be performed on $\mathbb{Z}_p$ where $p > n$, w.r.t. $[m_i]^p$ instead of $[m_i]^2$. We apply the sharing conversion protocol Π_{convert} in Sec. III to expand $[m_i]^2 \in \mathbb{Z}_2$ to $[m_i]^p \in \mathbb{Z}_p$.

Now we have shared list $\{[s_i]^p\}_{i\in[n]}$, where its zero element position ρ corresponds to the comparison result of a and b, this 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 issue, we introduce a new primitive – Oblivious Short-list Zero Check (OZC).

Oblivious Short-listed Zero Check. The OZC scheme checks if a shared list contains zero on a subsequence. We formalize its functionality in Fig. 15. In particular, an OZC scheme allows P_0 input k-dimension selective index set $\mathcal{I} := \{\zeta_0, \ldots, \zeta_{k-1}\}$, P_0 and P_1 input 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.

We construct our secure comparison protocol with OZC. At a high level, 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 $\mathcal{F}_{\mathsf{ozc}}$ (We assume there are k bits in a equal to Δ). P_0 and P_1 input aforemationed $\{[s_i]^p\}_{i\in[n]}$ as the shared list of $\mathcal{F}_{\mathsf{ozc}}$. P_1 will receive the zero check result z. 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 $z = 1$, P_0 successfully guesses the correct result $a_\rho = \Delta$. 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 $[c]_1 = z \oplus 1$.
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. The full description of our secure comparison protocol is depicted in Figure 9. Next, we explain our protocol step by step as follows.

- At step 1, P_0 and P_1 invoke $\Pi^p_{\mathsf{convert}}(a_i, b_i)$ 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 steps 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 $[c]^2_0 \oplus [c]^2_1 = \mathbf{1}\{a < b\}$.

<u>Execution:</u>

1) Let $p := \lceil \log(n) \rceil$, for $i \in [n]$, P_0 and P_1 invoke $[m_i] \leftarrow \Pi^p_{\mathsf{convert}}(a_i, b_i)$, and then P_j holds $[m_i] \in \mathbb{Z}_p$, P_1 holds $y_i \in \mathbb{Z}_p$.
2) P_0 sets $a_n = [m_n]_0 = 0$; P_1 sets $b_n = [m_n]_1 = 1$;
3) For $i \in [n+1]$, P_0 computes $[s_i]_0$, where $[s_i]_0 = \sum_{j=0}^{i}[m_j]_0 - 2 \cdot [m_i]_0 + 1$, and P_1 computes $[s_i]_1$, where $[s_i]_1 = \sum_{j=0}^{i}[m_j]_1 - 2 \cdot [m_i]_1 + 1$;
4) P_0 and P_1 sets $[s_{n+1}]_0 = [s_{n+1}]_1 = 1$;
5) P_0 picks $\Delta \leftarrow \{0, 1\}$;
6) P_0 sets $\mathcal{I} := \{\zeta_j\}_{j \in \mathbb{Z}_k} = \{i | a_i = \Delta, i \in \mathbb{Z}_{n+1}\}$, where we assume the size of $\mathcal{I}$ is k;
7) For $j \in \{k, \ldots, n\}$, P_0 appends $\zeta_j = n$ to get $n+1$-dimension vector $\mathcal{I}'$;
8) P_0 and P_1 invoke $\mathcal{F}_{\mathsf{ozc}}[n+1, n+2]$ with index list $\mathcal{I}$, shared list $\{[s_i]_0\}_{i \in [n+2]}$ and $\{[s_i]_1\}_{i \in [n+2]}$; P_1 receives $z \in \{0, 1\}$;
9) P_1 sets $[c]^2_1 = z \oplus 1$.
10) P_0 set $[c]^2_0 = \Delta$.

Fig. 9: The Comparison Protocol

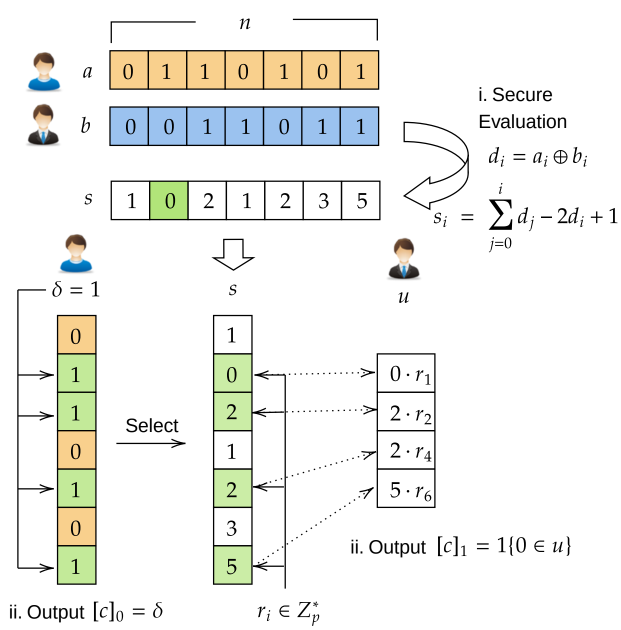


Fig. 10: The Overview of Secure Compariosn

- At steps 4, P_0 and P_1 sets $[s_{n+1}]_0 = [s_{n+1}]_1 = 1$ for dummy queries.

- At step 5-6, P_0 picks random Δ, records all indices i where $a_i = \Delta$, and denotes the set of these indices as $\mathcal{I}$. We assume the size of the set $\mathcal{I}$ is k, namely, $\mathcal{I} = \{\zeta_j\}_{j \in \mathbb{Z}_k}$.
- At step 7, to prevent the leakage of the hamming weight of a, P_0 pads the size of $\mathcal{I}$ 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 $\mathcal{F}_{\mathsf{ozc}}$. Specifically, P_0 inputs the index list $\mathcal{I} = \{\zeta_j\}_{j\in[n+1]}$ and the shared list $\{[s_i]_0\}_{i\in[n+1]}$, and P_1 inputs the shared list $\{[s_i]_0\}_{i\in[n+2]}$. After the 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 sets output $[c]_1 = z \oplus 1$ and P_0 set output $[c]_0 = \Delta$.

Our secure comparison protocol Π^n_{cmp} requires 1-round communication of $2n$ bits in the online phase for the n times invoking of Π_{convert} and one time $\mathcal{F}_{\mathsf{ozc}}[n+1, n+2]$.

Security. We define the functionality $\mathcal{F}_{\mathsf{cmp}}$ for secure comparison as an instance of $\mathcal{F}_{\mathsf{2PC}}$, where $\mathcal{F}_{\mathsf{cmp}}$ receives a and $[c]^2_0 \in \{0,1\}$ from honest P_0 or $\mathcal{S}$, receives b from honest P_1 or $\mathcal{S}$, calculates $[c]^2_1 = \mathbf{1}\{a > b\} \oplus [c]^2_0$ and sends to P_1. Next, we prove our protocol Π_{cmp} realizes functionality $\mathcal{F}_{\mathsf{cmp}}$.

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. 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 $\mathsf{Real}^{\mathcal{F}_{\mathsf{ozc}},\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}}_{\Pi_{\mathsf{cmp}},\mathcal{A},\mathcal{Z}}(1^\lambda)$. We consider the following cases:

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

- Upon receiving $(\mathsf{Input}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{cmp}}$, $\mathcal{S}$ starts simulation.
- For the simulation of i^{th} times of Π_{convert}, $i \in [n]$,
  - $\mathcal{S}$ picks random $[r_i]^2 \in \mathbb{Z}_2$ and emulates $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$ with input $[r_i]^2$;
  - When corrupted P_0 inputs $(m_{0,i}, m_{1,i})$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$, $\mathcal{S}$ records $(m_{0,i}, m_{1,i})$.
  - $\mathcal{S}$ calculate r_i and s_i with $m_{0,i}, m_{1,i}$;
  - $\mathcal{S}$ picks $[w_i]^2_1 \in \mathbb{Z}_2$ and acts as P_1 to send it to P_0.
  - Upon receiving $[w_i]^2_0$ from P_0, $\mathcal{S}$ calculate $a_i = [w_i]^2_0 \oplus [r_i]^2_0$
- $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{ozc}}$ with random list $\{s_i\}_{i\in\mathbb{Z}_{n+1}}$.
- When P_0 input $\mathcal{I}$ to $\mathcal{F}_{\mathsf{ozc}}$, $\mathcal{S}$ records $\mathcal{I}$ and calculates $\Delta := a_i$ for $i \in \mathcal{I} \wedge i \neq n+1$.
- If $a = 0$ or $a = 2^n - 1$ and $\mathcal{I} := \{n+1, \ldots, n+1\}$, set $\Delta = 1 \oplus a_0$.
- $\mathcal{S}$ sends $(\mathsf{Input}, \mathsf{sid}, a, \Delta)$ to external $\mathcal{F}_{\mathsf{cmp}}$.

Observe that P_0 locally set $[c]_0 = \Delta$, so that the output of ideal execution keeps consistent with the real execution. We show that the incoming message of P_0 in the ideal world is indistinguishable from the real world.

Claim 3. If $\mathsf{PRF}^{\mathbb{Z}_2}$ is the secure pseudorandom functions with adversarial advantage $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2}}(1^\lambda, \mathcal{A})$, then the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{cmp}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and the real world $\mathsf{Real}_{\Pi_{\mathsf{cmp}},\mathcal{A},\mathcal{Z}}(1^\lambda)$ are indistinguishable with advantage $\epsilon = \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2}}(1^\lambda, \mathcal{A})$.

Proof. In the ideal world, $[w_i]_1^2$ are picked random rather than calculated by $[a]_i^2 \oplus [r]_i^2$, which replace n $\mathsf{PRF}^{\mathbb{Z}_2}$ outputs to uniformly random; therefore, the overall advantage is $\epsilon = n \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2}}(1^\lambda, \mathcal{A})$. $\square$

Case 2: P_1 is corrupted. We construct the simulator $\mathcal{S}$ to simulates the interface of honest P_0

- Upon receiving $(\mathsf{Input}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{cmp}}$, $\mathcal{S}$ picks a.
- For the simulation of i^{th} times of Π_{convert}, $i \in [n]$,
  - $\mathcal{S}$ picks random $[r_i]^2 \in \mathbb{Z}_2$, $[s_i]^p \in \mathbb{Z}_p$ and emulates $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$with input $m_0 = [s_i]_0^p - [r_i]_0^2$, $m_1 = [s_i]_0^p - (1 - [r_i]_0^2)$;
  - When corrupted P_0 inputs $[r_i]_1^2$ to $\mathcal{F}_{(1,2)\text{-}\mathsf{OT}}$, $\mathcal{S}$ records $[r_i]_1^2$ and sends $m_{[r_i]_1^2}$ to P_1.
  - Upon receiving $[w_i]_1^2$ from P_1, $\mathcal{S}$ calculate $b_i = [w_i]_1^2 \oplus [r_i]_1^2$
  - $\mathcal{S}$ calculates $[w_i]_1^2 = [r_i]_1^2 \oplus a_i$ and acts as P_1 to send it to P_0.
- For simulation of Π_{cmp},
  - $\mathcal{S}$ calculates $\{s_i\}_{i\in[n+1]}$ with $\phi((a||0) \oplus (b||1))$, it holds that $s_\rho = 0$ and $\rho < n+1$.
  - Upon receiving $[c]_1^2$ from $\mathcal{F}_{\mathsf{cmp}}$, $\mathcal{S}$ does:
    - if $[c]_1^2 = 1$, set $\mathcal{I}' := \{n+1, n+1, \ldots, n+1, n+1\}$ with $n+1$ dimension.
    - if $[c]_1^2 = 0$, set $\mathcal{I}' := \{n+1, n+1, \ldots, n+1, \rho\}$ with $n+1$ dimension.
  - $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{ozc}}$ with random list $\{s_i\}_{i\in[n+2]}$ and selection list $\mathcal{I}$.

We first show that in the ideal world, P_1 reveives same output as the real world: If $\mathcal{I}' := \{n+1, n+1, \ldots, n+1, n+1\}$, $\mathcal{F}_{\mathsf{ozc}}$ will output all positive value $\{\beta_i \cdot s_{n+1}\}_{i\in[n/2]}$ to P_1 induce P_1 to output $[c]_1^2 = 1$. On the contrary, if $\mathcal{I}' := \{n+1, n+1, \ldots, n+1, \rho\}$, $\mathcal{F}_{\mathsf{ozc}}$ will output zero-contained list to P_1 such that P_1 output $[c]_1^2 = 0$. Next, we show that the incoming message of P_1 in the ideal world is indistinguishable with the real world.

Claim 4. For two sets of list $(\mathcal{I}, X, Y)$ and $(\mathcal{I}', X', Y')$, where

- $\mathcal{I} := \{\zeta_i\}_{i\in[k]} \in \mathbb{Z}_n^k$;$\mathcal{I}' := \{\zeta'_i\}_{i\in[k]} \in \mathbb{Z}_n^k$;
- $X := \{x_i\}_{i\in[n]} \in \mathbb{Z}_p^n$;$X' := \{x'_i\}_{i\in[n]} \in \mathbb{Z}_p^n$;
- $Y := \{y_i\}_{i\in[n]} \in \mathbb{Z}_p^n$;$Y' := \{y'_i\}_{i\in[n]} \in \mathbb{Z}_p^n$;

If it have

- $\mathcal{M} := \{x_i + y_i\}_{i\in[n]}$ and $\mathcal{M}' := \{x'_i + y'_i\}_{i\in[n]}$ only contains one 0 (denoted by m_ρ and m'_ρ), and contains non-zero value in the other position.
- The number of ρ contained in $\mathcal{I}$ and $\mathcal{I}'$ are both $\ell \in \{0, 1\}$.

It holds that

$$Pr[\mathcal{A}(\mathcal{F}_{\mathsf{ozc}}(\mathcal{I}_b, X_b, Y_b), \{\mathcal{I}_i, X_i, Y_i\}_{i\in[2]}) = b] < \frac{1}{2} + \mathsf{negl}$$

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

Input : Index list $\mathcal{I} := \{\zeta_i\}_{i\in[k]}$ 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:
  - $(\beta_i, r_i, u_i, v_i) \leftarrow \mathcal{F}_{\mathsf{ole}}[p]$, for $i \in [n]$.
  - $(\{\beta_j\}_{j\in[k-1]}, r, \{u_j\}_{j\in[k-1]}, \{v_j\}_{j\in[k-1]}) \leftarrow \mathcal{F}_{\mathsf{vole}}[p, k-1]$
- 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 $\mathcal{F}_{\mathsf{permute}}$:
  - 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:

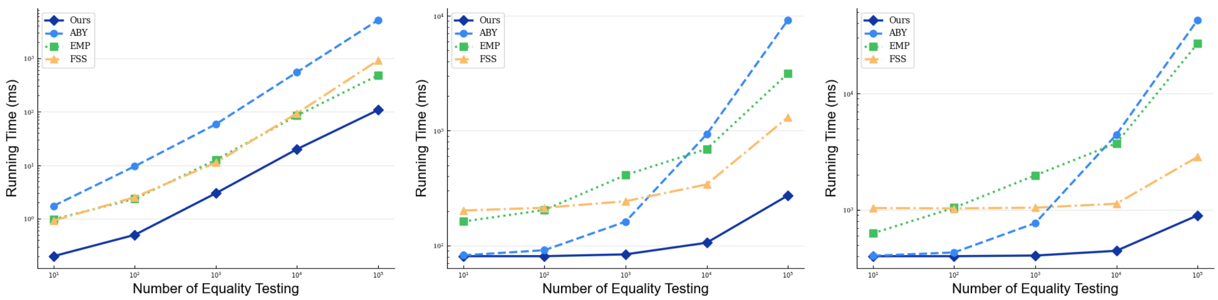
- P_1 sets $y'_i = y_i + r_i$ for $i \in [n]$ and sends the set $Y' = \{y'_0, \cdots, y'_n\}$ to P_0;
- P_0 sets
  - $t_i = \beta_{\zeta_i} \cdot (x_{\zeta_i} + y'_{\zeta_i}) - [w_{\pi^-(\zeta_i)}]_0$ for $i \in [k-t]$;
  - $s_i = \pi^-(\zeta_i)$ for $i \in [k-t]$;
  - $t_i = \beta_{n+i-k} \cdot (x_n + y'_n) - [w_{\pi^-(n+i-k)}]_0$ for $i \in [k-t, k]$;
  - $s_i = \pi^-(n+i-k)$ for $i \in [k-t, k]$;
- P_0 sends $\{t_i\}_{i\in[k]}$ and $\{s_i\}_{i\in[k]}$ to P_1.
- P_1 calculates $z_i = t_i - [w_{s_i}]$ for $i \in [k-t]$.
- P_1 outputs $z = \mathbf{1}\{0 \in \{z_0, \cdots, z_{k-1}\}\}$.

Fig. 11: The Oblivious Selective Multiplication Protocol

Claim 5. If $\mathsf{PRF}^{\mathbb{Z}_2^n}$ is the secure pseudorandom functions with adversarial advantage $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2^n}}(1^\lambda, \mathcal{A})$, then the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{cmp}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and the real world $\mathsf{Real}_{\Pi_{\mathsf{cmp}},\mathcal{A},\mathcal{Z}}(1^\lambda)$ are indistinguishable with advantage $\epsilon = \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2}}(1^\lambda, \mathcal{A})$.

Proof. In the ideal world, $[w_i]_1^2$ are calculated by randomly picked a, which replace random value to n to output of where the advantage is $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_2^n}}(1^\lambda, \mathcal{A})$. For the list $\{z_i\}_{i\in[n+1]}$, from Claim. 4, it is indistinguishable between the ideal world and real world. $\square$

This concludes the proof. $\square$



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

Fig. 12: The running time of equality testing protocol Π_{eq_2} compare with ABY [19], GC scheme implemented in EMP-toolkits [48] and DPF [26] in LAN/MAN/WAN setting. All benchmarks take the data length $n = 32$.

B. Realize $\mathcal{F}_{\mathsf{ozc}}$

We propose a naive 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 in the offline phase.

OLE-based implement. Recall the functionality $\mathcal{F}_{\mathsf{ozc}}$ accepts list $X := \{x_0, \dots, x_{n-1}\}$, $Y := \{y_0, \dots, y_{n-1}\}$ and a index list $\mathcal{I} := \{\zeta_0, \dots, \zeta_{k-1}\}$. $\mathcal{F}_{\mathsf{ozc}}$ sends the information of whether there exists $x_i + y_i = 0$ for $i \in \mathcal{I}$ to P_1. The naive approach to implementing OZC is to scale all selected items $x_{\zeta_i} + y_{\zeta_i}$ with non-zero random value β_i and directly reveal to P_1, namely, $c_i = \beta_i \cdot (x_{\zeta_i} + y_{\zeta_i})$. P_1 check whether there exist $c_i = 0$ for $i \in [k]$ to verify $x_i + y_i = 0$. To hide the index ζ_i, we let P_0 first take y_{ζ_i} using $(1, n)$-OT, then P_0 picks β_i and calculates $c_i = \beta_i \cdot (x_{\zeta_i} + y_{\zeta_i})$. Avoiding reveal y_{ζ_i} to P_0, we employ P_1 generate r to mask y_{ζ_i} and rewrite z as $[z] = \beta \cdot (x_{\zeta_i} + y_{\zeta_i} + r) - [\beta \cdot r]$. P_0 takes $y'_{\zeta_i} = y_{\zeta_i} + r$ from OT instead of y_{ζ_i}. For the part of $[\beta \cdot r]$, it can be produced by OLE tuple generation protocol with random $\beta \in \mathbb{Z}_p$ and $r \in \mathbb{Z}_p$, where P_0 holds $\{\beta, [\beta \cdot r]_0\}$ and P_1 holds $\{r, [\beta \cdot r]_1\}$. At present, P_1 can locally calculate $[c_i]_0 = \beta_i \cdot (x_{\zeta_i} + y'_{\zeta_i}) - [\beta \cdot r]_0$ and P_1 calculate $[c_i]_1 = -[\beta \cdot r]_1$. When P_0 reveal $[c_i]_0$ to P_1 for reconstruction c_i. The naive approach is illustrated in Fig. 16 (CF. Appendix. A-C).

<u>Remark.</u> To avoid the 0 caused by wrapping round $\beta_i \cdot (x_i + y_i)$ with non-zero $x_i + y_i$, β_i and p should be coprime. We exclude such cases by taking p as prime and $\beta_i \in \mathbb{Z}_p^*$.

Online Phase Communication Optimization. For k indices, Π_{ozc} requires invoking k times 1-out-of-n OT in the online phase, which is a huge communication cost. We optimize the online phase communication through the oblivious permutation. Our starting point is that y'_i in Π_{ozc} can be masked with different r_i and directly reveal to P_0. Instead of OT, P_0 can directly select y'_{ζ_i} and calculate $\beta_{\zeta_i}(x_{\zeta_i} + y'_{\zeta_i}) = \beta_{\zeta_i}(x_{\zeta_i} + y_{\zeta_i} + r_{\zeta_i})$. The challange is how to cancel $\beta_{\zeta_i} \cdot r_{\zeta_i}$ when P_1 doesn't know ζ_i. We introduce permutation tuples to address this issue. In particular, the permutation tuple $(\{\beta_i, r_i, [w_i]_0, [w_i]_1\}_{i \in [n]}, \pi)$ is generated in the offline phase, where it holds that,

- 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 $z_i = \beta_{\zeta_i}(x_{\zeta_i} + y_{\zeta_i} + r_{\zeta_i}) - \beta_{\zeta_i} \cdot r_{\zeta_i}$, we can replace $\beta_{\zeta_i} \cdot r_{\zeta_i}$ with $[w_{\pi(\zeta_i)}]_0 + [w_{\pi(\zeta_i)}]_1$. Namely, $z_i = \beta_{\zeta_i}(x_{\zeta_i} + y_{\zeta_i} + r_{\zeta_i}) - [w_{\pi(\zeta_i)}]_0 - [w_{\pi(\zeta_i)}]_1$. P_0 hold β_{ζ_i}, x_{ζ_i}, $y'_{\zeta_i} = y_{\zeta_i} + r_{\zeta_i}$, π and $[w_{\pi(\zeta_i)}]_0$ so that it can calculate $t_i = \beta_{\zeta_i}(x_{\zeta_i} + y_{\zeta_i} + r_{\zeta_i}) - [w_{\pi(\zeta_i)}]_0$. Since π is a uniformly random permutation, $\pi(\zeta_i)$ can be revealed to P_1 directly without information leakage about ζ_i. Consequantly, P_1 calculates $z_i = t_i - [w_{\pi(\zeta_i)}]_1$ which is equal to $\beta_{\zeta_i}(x_{\zeta_i} + y_{\zeta_i})$ and checks if there exists $z_i = 0$ for $i \in [k]$.

Privacy on dummy queries. The foregoing version of the protocol can not deal with the duplicated indices. Because the same index ζ_k will obtain the same permuted index $\pi(\zeta_k)$ which can not be directly revealed to P_1, leading to an incompatible with the original dummy queries approach. Our solution is to generate another $k-1$ dimension VOLE permutation tuple $(\{\beta_i, [w_i]_0, [w_i]_1\}_{i\in[k-1]}, r)$, where it holds

- $\beta_i \cdot r = [w_{\pi(i)}]_0 + [w_{\pi(i)}]_1$ 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 overall tuples, namely, $(\{\beta_i, r_i, [w_i]_0, [w_i]_1\}_{i\in[n+k-1]}, \pi)$ where $r_n = r_{n+1} \ldots = r_{n+k}$ corresponds to the r of VOLE tuple. We utilize VOLE tuples to deal with the duplicated indices. In particular, assume the last t items of $\mathcal{I}$ is duplicated indices, i.e. $\zeta_i = \eta$ for $i \in [k-t, k]$. P_0 sets $t_i = \beta_{n+i-k+t} \cdot (x_\eta + y_\eta + r_\eta) - [w_{\pi(n+i-k+t)}]_0$ and sends t_i and $\pi(n+i-k+t)$ to P_0. Analogously, P_1 can recover $z_i = \beta_{n+i-k+t} \cdot (x_\eta + y_\eta)$ for the duplicated index.

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 VOLEtuple 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 $i \in [n]$. In our benchmark, we use the SOTA protocol to realize $\mathcal{F}_{\mathsf{ole}}$ [33],$\mathcal{F}_{\mathsf{vole}}$ [44],$\mathcal{F}_{\mathsf{permute}}$ [15].

Our complete protocol design is illustrated in Figure. 11. Our oblivious short-list zero check protocol $\Pi_{\mathsf{ozc}}^{k,n,p}$ requires 2-round communication of $2 \cdot k \cdot p$ bits in the online phase. In the offline phase, it requires n times invoking of $\mathcal{F}_{\mathsf{ole}}[p]$, one time invoking of $\mathcal{F}_{\mathsf{vole}}[k-1, p]$ and one time invoking of $\mathcal{F}_{\mathsf{permute}}[n+k-1]$.

Theorem 3. The protocol Π_{ozc} as depicted in Fig. 11 UC realizes $\mathcal{F}_{\mathsf{ozc}}$ against semi-honest PPT adversaries who can statically corrupt up to one party.

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 $\mathcal{A}$, forwarding messages to/from $\mathcal{Z}$ and simulates the interface of honest P_1.

- $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{ole}}$, outputs (β_i, r_i, u_i, v_i) for $i \in [n]$ and sends (β_i, u_i) to P_0.

- $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{vole}}$, outputs $(\{\beta_j, r, u_j, v_j\}_{j\in[k-1]})$ and sends (β_j, u_j) to P_0.
- $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{permute}}$ with input v_i and record π.
- $\mathcal{S}$ picks random list $\{y'_i\}_{i\in[n]}$ and acts as P_1 to send it to P_0.
- Upon receiving $\{t_i\}_{i\in[k]}$ and $\{s_i\}_{i\in[k]}$, $\mathcal{S}$ does
  - calculate $\zeta_i = \pi(s_i)$ for $i \in [k]$.
  - calculate $x_{\zeta_i} = \beta^{-}_{\zeta_i}(t_i + [w_{\pi^-(\zeta_i)}])$
  - 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 $\mathcal{F}_{\mathsf{ozc}}$.

Observe that $\mathcal{F}_{\mathsf{ozc}}$ will the output each items $z_i = (x_{\zeta_i} + y_{\zeta_i}) \cdot \beta_i$ to P_1, which equals to z_i in the real world. We show that the incoming message of P_0 in the ideal world is indistinguishable with the real world.

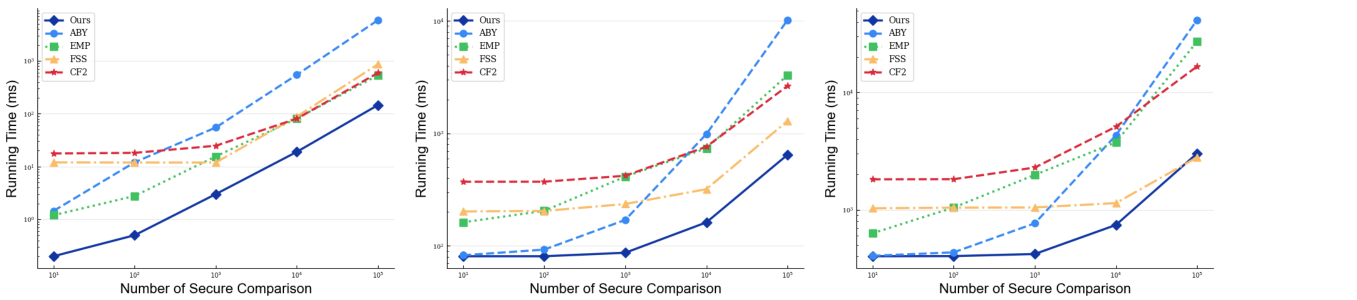
Claim 6. If $\mathsf{PRF}^{\mathbb{Z}_p}$ is the secure pseudorandom functions with adversarial advantage $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\lambda, \mathcal{A})$, then the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{cmp}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and the real world $\mathsf{Real}_{\Pi_{\mathsf{cmp}},\mathcal{A},\mathcal{Z}}(1^\lambda)$ are indistinguishable with advantage $\epsilon = n \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\lambda, \mathcal{A})$.

Proof. In the ideal world, $\{y'_i\}_{i\in[n]}$ are uniformly random, which replace n value of PRF output, where the advantage is $n \cdot \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_p}}(1^\lambda, \mathcal{A})$. □

Case 2: P_1 is corrupted. We construct the simulator $\mathcal{S}$ to simulates the interface of honest P_0

- In the offline phase, $\mathcal{S}$ does,
  - emulates $\mathcal{F}_{\mathsf{ole}}$, outputs (β_i, r_i, u_i, v_i) for $i \in [n]$ and sends (r_i, v_i) to P_1.
  - emulates $\mathcal{F}_{\mathsf{vole}}$, outputs $(\{\beta_j, r, u_j, v_j\}_{j\in[k-1]})$ and sends (r, v_j) to P_1.
  - picks random permutation π and emulates $\mathcal{F}_{\mathsf{permute}}$ with input π.
  - Record v_i when P_1 input it to $\mathcal{F}_{\mathsf{permute}}$.
- Upon 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 z from $\mathcal{F}_{\mathsf{ozc}}$, $\mathcal{S}$ does,
  - pick random list $\{r_0, \ldots, r_{k-1}\} \in (\mathbb{Z}_p^*)^k$.
  - pick random set $\mathcal{I}_1 := \{s_0, \ldots, s_{k-1}\} \in \mathbb{Z}^k_{n+k-1}$.
  - for $i \in [k]$, set $t_i = [w_{s_i}] + r_i$;
  - if $b = 0$, pick $\eta \leftarrow \mathcal{I}_1$ and set $t_i = t_i - r_i$.
  - act as P_0 to send $\{t_i\}_{i\in[k]}$ and $\mathcal{I}_1$ to P_1.

Claim 7. If $\mathsf{PRF}^{\mathbb{Z}^k_n}$ are the secure pseudorandom functions with adversarial advantage $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}^k_n}}(1^\lambda, \mathcal{A})$, then the ideal world $\mathsf{Ideal}_{\mathcal{F}_{\mathsf{cmp}},\mathcal{S},\mathcal{Z}}(1^\lambda)$ and the real world $\mathsf{Real}_{\Pi_{\mathsf{cmp}},\mathcal{A},\mathcal{Z}}(1^\lambda)$ are indistinguishable with advantage $\epsilon = \mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}^k_n}}(1^\lambda, \mathcal{A})$.



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

Fig. 13: The running time of Π_{cmp} compare with ABY [19], GC implemented in EMP [48], DCF [26] and CrypFlow2 [43] in LAN/MAN/WAN setting; take the data length $n = 64$; CF2 refers to CrypFlow2.

Proof. In the ideal world, $\{s_i\}_{i\in[k]}$ and $\{t_i\}_{i\in[k]}$ are randomly generated instead of calculated by (X, Y) and $(\pi, \mathcal{I})$. Obviously, the advantage betweent random set $\mathcal{I}_1$ and $\{s_i\}_{i\in[k]}$ which is calculated by ramdom permutation π in the real world is $\mathsf{Adv}_{\mathsf{PRF}^{\mathbb{Z}_n^n}}(1^\lambda, \mathcal{A})$. For $\{t_i\}_{i\in[k]}$, due to the random mask β_i and $[w_i]$ are generated from ole and vole, $\{t_i\}_{i\in[k]}$ in the real world and ideal world are indistinguishable. Therefore, the overall advantage is $\mathsf{PRF}^{\mathbb{Z}_n^k}$. □

This concludes the proof. □

V. Performance Evalutaion

In this section, we respectively implement our equality test (Section III) and secure comparison (Section IV), and compare their performance with the CrypTFlow2 [43], ABY [19], GC [3], FSS [26].

A. Experiment Setting

We implement our protocols in C++. For the $\mathcal{F}_{\mathsf{OT}}$, we utilize the OT library – libOTe [4]. For FSS, we implement the keys correlated generation scheme for benchmark [1]. For the garbled circuit, we utilize EMP-toolkits [3], which is integrated half-gate [51]. 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 $\lambda = 128$.

Equality testing. The equality testing running time of the online phase (for $n = 64$) is depicted in Fig. 12. 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 [26], while the computation cost of our protocol is more subtle than FSS, leading to a significant performance superiority.

In general, considering appropriate data size, the efficiency of our equality-testing is (i) over $2\times$ of the garbled circuit, over $7\times$ of the FSS, and over $40\times$ of ABY in the LAN setting; (ii) over $9\times$ of the FSS, over $15\times$ of garble circuit and over $50\times$ of ABY in both MAN and WAN settings. Fig. 17(a) depicts the offline running time compared to FSS (with the correlated keys generation) and ABY. (Considering $\mathcal{O}(2^n)$ computation complexity of FSS, we take $n = 16$). The offline running time of our protocol is over $1000\times$ faster than FSS and over $5\times$ faster than ABY.

Secure comparison. Fig. 18 depicts the online phase running time of secure comparison compared to ABY [19], GC [48], DCF [26] and CrypFlow2 [43] (Due to CrypTflow2 only support 64 bits, our benchmarks perform on $n = 64$). In most cases, our protocol outperforms other protocols in the online phase. In particular, the efficiency of our protocol is (i) over $3\times$ of the FSS/CrypTflow2/GC, and over $20\times$ of the ABY in the LAN setting; (ii) over $3\times$ of the FSS, over $6\times$ of GC/CrypTflow2 and over $15\times$ 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 $1000\times$ of FSS. As a trade-off, our offline phase is slightly slower than ABY.

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

VI. 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.

References

[1] Correlated fss keys generation. https://anonymous.4open.science/r/fss-28E7.

[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://anonymous.4open.science/r/2PC_eq_cmp-4C54.

[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, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In CCS, 2016.

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

[13] 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.

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

[15] 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.

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

[17] 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.

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

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

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

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

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

[28] 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.

[29] 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.

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

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

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

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

[34] 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.

[35] 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.

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

[37] 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.

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

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

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

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

[47] 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.

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

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

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

[51] 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.

[52] 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.

Appendix A

Other building block

This section gives other building blocks such as the OLE and the oblivious short-list zero check.

A. 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 $\mathcal{F}_{\binom{1}{2}\text{-OT}}$, 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 $\mathcal{F}_{\binom{1}{2}\text{-OT}}$; 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 $a \cdot b = c + d$.

Execution:

- 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 $\mathcal{F}_{\binom{1}{2}\text{-OT}}$:
  - 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 $d = \sum_{j=1}^{p} z_j$.

Fig. 14: The Oblivious Linear Evaluation Triple Generation Protocol

B. The functionality of oblivious short list zero check

In this section, we define the functionality of oblivious short-list zero checks. In particular, an OZC scheme allows P_0 input k-dimension selective index set $\mathcal{I} := \{\zeta_0, \ldots, \zeta_{k-1}\}$, P_0 and P_1 input 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.

C. Oblivious short-list zero check with OLE

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

Appendix B
Other benchmarks

In this section, we give more benchmars.

A. Offline of Equality Testing and Secure Comparison

Figure 17 shows the running time in the offline phase for the equality testing and secure comparison protocol compared with ABY [19] and DPF [26] in the LAN setting. The running time of our equality testing protocol in the offline phase is entirely superior to the DPF [26], outperforming ABY [19] when the

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

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

Input:

- Upon receiving $(\mathsf{Input}, \mathsf{sid}, \mathcal{I}, X)$ from $P_0 \in \mathcal{P}$, record $(\mathcal{I}, X)$ and send $(\mathsf{Input}, \mathsf{sid}, P_0)$ to $\mathcal{S}$, where
  - $X := \{x_0, \cdots, x_{n-1}\} \in \mathbb{Z}_p^n$;
  - $\mathcal{I} \in \mathbb{Z}_n^k$;
- 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 $\mathcal{I}, X$ and Y are recorded, $\mathcal{F}_{\mathsf{ozc}}$ does:
  - set $b = 1$ if $\exists i \in \mathcal{I}, x_{\zeta_i} + y_{\zeta_i} = 0$.
  - set $b = 0$ otherwise.
- Send $(\mathsf{Output}, \mathsf{sid}, b)$ to P_1.

Fig. 15: The Ideal Functionality $\mathcal{F}_{\mathsf{ozc}}$.

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

Input : Index list $\mathcal{I} := \{\zeta_i\}_{i\in[k]}$ 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:

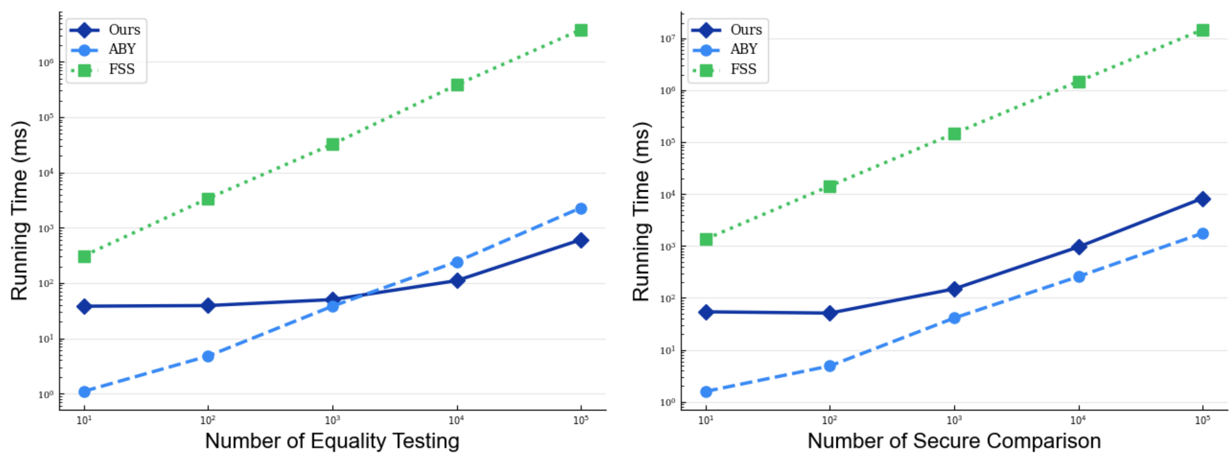
- P_0 and P_1 invoke n times $\{\beta_i, r_i, [t_i]_0^p, [t_i]_1^p\} \leftarrow \Pi_{\mathsf{ole}}$, where P_0 holds $\{\beta_i, [t_i]_0^p\}$, P_1 holds $\{r_i, [t_i]_1^p\}$.

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;

Fig. 16: The Oblivious Short-List Zero Check with OLE Protocol.

batch size exceeds 1000. Similarly, our secure comparison protocol is also based entirely on the DPF [26]. Although it is slower than ABY [19], the offline performance loss is acceptable for the overall protocol as it achieves a 15× improvement in running time over ABY during the online phase. In addition, to provide a more detailed comparison of the efficiency between our protocols and ABY, we present the offline running time of our protocols compared to ABY under LAN/MAN/WAN settings in the table III. The running time is given in milliseconds. The results indicate that the higher the bandwidth, the more significant the performance advantage of our protocol.



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

Fig. 17: The running time of offline phase on equality testing protocol Π_{eq_2} and secure comparison protocol Π_{cmp} compare with ABY [19] and DPF [26] in LAN setting.

B. 32-bit Secure comparison

Due to Cryptflow2 [43] only supporting the 64-bit secure comparison, we benchmark the running time of the secure comparison protocol Π_{cmp} compared with ABY [19], the GC scheme implemented in EMP-toolkits [48], and DCF [26] in LAN/MAN/WAN settings, where takes the elements size $n = 32$ in Figure 18. All benchmarks assume an input length of $n = 32$. The results show that our protocol achieves the best running time across all network settings and batch sizes.

C. Running time in different input length

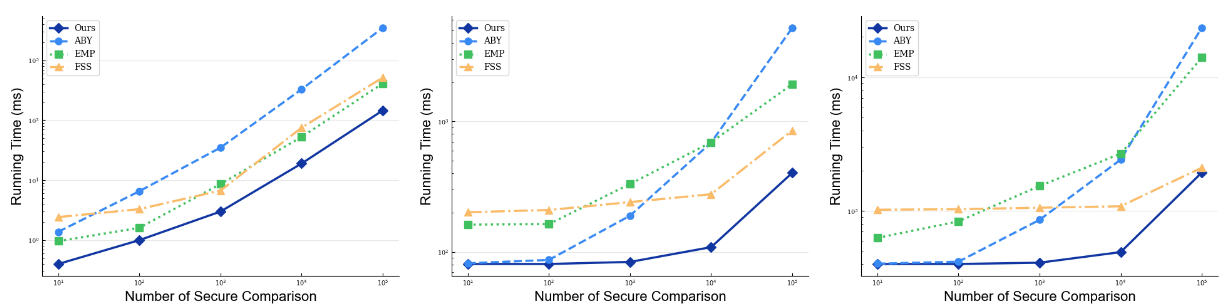
Table IV exhibits the online running time of our protocols for different input lengths and batch sizes, provided in milliseconds. The results show that under a WAN setting of 10Mbps, the online running time of our equality testing protocol remains nearly constant at less than 0.5s for batch sizes below 10000, and is approximately 1s for a batch size of 100000. For the secure comparison protocol, the running time is only 5s when the batch size is 100000. The performance is even better in other network environments.

D. Running time in different input length

Table IV exhibits the online running time of our protocols for different input lengths and batch sizes, provided in milliseconds. The results show that under a WAN setting of 10Mbps, the online running time

TABLE III: Offline running time of our protocols compared to ABY, under LAN/MAN/WAN settings. The running time is given in ms.

Batch size	100	1000	10000	100000
10Mbps 100ms				
Our equality testing	1457	2527	4221	15634
ABY equality testing	213	1477	4574	14434
Our secure comparison	3510	4344	10678	64996
ABY secure comparison	129	778	4056	14613
400Mbps 20ms				
Our equality testing	326	626	818	2649
ABY equality testing	46	361	1126	3891
Our secure comparison	809	1065	3175	22873
ABY secure comparison	45	321	1093	3440
20Gbps 0.01ms				
Our equality testing	41	51	137	818
ABY equality testing	4	38	237	2233
Our secure comparison	66	312	1991	18773
ABY secure comparison	5	41	258	1764



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

Fig. 18: The running time of secure comparison protocol Π_{eq_2} compare with ABY [19], GC scheme implemented in EMP-toolkits [48], DCF [26] and CrypFlow2 [43] in LAN/MAN/WAN setting. All benchmarks take the input length $n = 32$. CF2 refers to CrypFlow2. EMP refers to EMP-toolkits.

of our equality testing protocol remains nearly constant at less than 0.5s for batch sizes below 10000, and is approximately 1s for a batch size of 100000. For the secure comparison protocol, the running time is only 5s when the batch size is 100000. The performance is even better in other network environments.

TABLE IV: The online running time of our protocols in different input lengths and batch sizes, which is given in ms.

Protocol	Secure comparison				Equality testing			
Size / Batch	16	32	64	128	16	32	64	128
WAN 10Mbps 100ms								
100	401	401	402	404	401	401	401	402
1000	405	410	419	437	402	403	406	410
10000	449	492	741	1005	414	426	446	476
100000	1006	1935	3017	5097	500	741	897	1189
MAN 400Mbps 20ms								
100	81	81	81	82	81	81	81	81
1000	83	84	87	92	82	82	84	87
10000	98	109	162	198	89	96	106	121
100000	215	404	646	1034	129	210	273	434
LAN 20Gbps 0.01ms								
100	<1	<1	1	1	<1	<1	<1	<1
1000	2	3	4	11	1	2	3	5
10000	14	19	35	44	6	12	20	34
100000	72	146	275	449	47	61	108	222