SIM: Secure Interval Membership Testing and Applications to Secure Comparison

Albert Yu
Purdue University
yu646@purdue.edu

Donghang Lu
Purdue University
lu562@purdue.edu

Aniket Kate
Purdue University
aniket@purdue.edu

Hemanta K. Maji
Purdue University
hmaji@purdue.edu

***Abstract*—The offline-online model is a leading paradigm for practical secure multi-party computation (MPC) protocol design that has successfully reduced the overhead for several prevalent privacy-preserving computation functionalities common to diverse application domains. However, the prohibitive overheads associated with secure comparison – one of these vital functionalities – often bottlenecks current and envisioned MPC solutions. Indeed, an efficient secure comparison solution has the potential for significant real-world impact through its broad applications.**

This work identifies and presents SIM, a secure protocol for the functionality of *interval membership testing*. This security functionality, in particular, facilitates secure less-than-zero testing and, in turn, secure comparison. A key technical challenge is to support a fast online protocol for testing in large rings while keeping the precomputation tractable. Motivated by the map-reduce paradigm, this work introduces the innovation of (1) computing a sequence of intermediate functionalities on a partition of the input into input blocks and (2) securely aggregating the output from these intermediate outputs. This innovation allows controlling the size of the precomputation through a granularity parameter representing these input blocks' size – enabling application-specific automated compiler optimizations.

To demonstrate our protocols' efficiency, we implement and test their performance in a high-demand application: privacy-preserving machine learning. The benchmark results show that switching to our protocols yields significant performance improvement, which indicates that using our protocol in a plug-and-play fashion can improve the performance of various security applications. Our new paradigm of protocol design may be of independent interest because of its potential for extensions to other functionalities of practical interest.

I. Introduction

Privacy-enhancing technologies, such as *secure multi-party computation* (MPC), are essential for bridging the data utility and privacy chasm. MPC [20], [46] allows mutually distrusting parties to compute over their private data without revealing any non-essential information. From initial conceptual prototypes in calculating prices of Danish sugar beet market [7], evaluating gender pay disparities in Boston [29], detecting tax fraud in Estonia [5], and preventing satellite collisions [22], advances in computer hardware have inspired a recent revolution in MPC research and technologies (see [18]).

As more memory and high-end processors become more available, a leading paradigm for practical MPC protocol design is the *offline-online model* [4], [9], [12], [13], [21], [25], [32], [33], [39], which offloads most computationally and cryptographically complex operations to an *offline precomputation* step. The *online phase* is a fast protocol that uses the output of the precomputation to perform the intended computation securely. Protocol design in this model proceeds by identifying essential atomic functionalities prevalent across diverse high-impact application domains. For example, atomic functionalities like multiplication, (multi-variate) polynomial evaluation, inner product, and comparison have representative applications in arithmetic circuit evaluation [32], decision-tree evaluation [19], [42], [43], private set intersection [14], [23], [38], and neural network training [1], [26], [34], [39], respectively. Individually, each of these atomic functionalities is sufficient to emulate any computation, especially any other atomic functionality listed above. However, in some cases, custom-built precomputations for individual atomic functionalities have demonstrated the possibility of reducing the security overhead of MPC technologies.

The *secure comparison* functionality, which compares whether one secret value is greater than another or not, is essential in various privacy-sensitive domains like machine learning [35], [44], [45], data analytics [6], and auctions [2], [7]. Furthermore, it is also essential for domains relying on linear programming and dynamic programming – prospective applications include, for example, optimization and biomedical research, respectively. Practical solutions for secure comparison shall create exciting collaborative opportunities in these application domains by meeting the privacy expectations of the actors. Therefore, it is not surprising that there are many works that present secure comparison as a key building block, such as ABY [16], ABY2.0 [36], ABY3 [34], FALCON [45], AriaNN [39], and SecureNN [44], or focus solely on the secure comparison, such as [10], [8], and [33]. This paper presents a versatile framework for designing *secure comparison protocols* in the offline-online model – allowing for performance optimizations in light of network quality (for example, latency, bandwidth, and throughput) and application-specific considerations.

Ishai et al. [25] offer a simple MPC approach that employs function tables to achieve an efficient online phase by precomputing the function for all possible inputs in the offline phase, such that the online phase is simply a table-lookup. This approach pushes as much computation as possible to the offline phase, resulting in a highly efficient online phase but exponentially high precomputation for typical input sizes. This idea may work for small input sizes, but the size of the function table grows exponentially with the input size, thus making many applications unpractical. Applying the *map-*

reduce framework [15] to [25], this work presents SIM, a general framework to construct efficient secure computation protocols for secure interval membership testing functionality, in turn, leading to efficient secure protocols for the "less than zero" testing and "greater than" functionalities.

This work explores a construction that leverages problem size reduction to shift many but not all computations to the offline phase, which significantly improves the performance of the online phase compared to other traditional methods while keeping the size of function tables computationally tractable and practical. In our secure comparison protocol, we divide the original input into multiple blocks, perform membership testing on each block, and combine block results into the final answer. These steps significantly reduce the size of function tables, thus making them practical and efficient. To the best of our knowledge, this is the first work that brings the use of precomputed function tables into the realm of practical MPC.

Our construction is in the *offline-online model*, where a third (offline) party prepares a suitable (input-independent) precomputation for two or more online parties (say) Alice and Bob in the offline phase (as in [40], [41], [47]). Online parties Alice and Bob respectively have private inputs $[x]_1$ and $[x]_2$ for the online phase of the secure computation protocol, which are additive secret shares of the input x (i.e., $x = [x]_1 + [x]_2$). At the end of the online protocol between Alice and Bob, they obtain the additive secret shares of the output (bit) of interval testing. Our construction is secure against semi-honest (honest-but-curious) adversaries in a two-online-party setting. Further, we achieve malicious security efficiently by extending our system setting to three or more online parties.

For an n-bit input, we divide the input into k blocks of ℓ-bits each such that $n = k \cdot \ell$. Our online phase of secure membership testing has three communication rounds, and its communication cost is $k \cdot \ell + 2k + 1$ bits. The precomputation results in the memory requirement of around $k(2^\ell + \ell 3^k)$ bits per input. We compare our work with various state-of-the-art comparison protocols, including ABY [16], ABY2.0 [36], ABY3 [34], BLAZE [37], and FALCON [45]. Overall, our work has lower communication complexity and outperforms most works with respect to round complexity.

To demonstrate the performance of our protocols, we implement our protocols on the state-of-the-art Falcon framework [45]. The performance benchmark illustrates that our secure comparison is about 2× faster than Falcon's secure comparison protocol, with 4× cheaper communication. In a semi-honest setting, by only replacing Falcon's secure comparison protocol with ours, the neural network training efficiency has improved to around 1.3× for a network with ReLU activation function and around 1.25× for a network with MaxPool activation function. In the malicious setting, we achieve an even larger performance improvement (1.35× faster and 5× cheaper communication for Network A).

A. Structure of the paper

In Section II, we provide a general picture of our protocols and explain how it works with a concrete example. In Section III, we introduce the background and the system setting of our protocols. Then we formalize the use of function tables in MPC in Section IV.

We formalize and provide the detailed construction of our protocol in Section V, which is followed by the cost analysis and security analysis (Section VI). In Section VII, we show how to achieve malicious security efficiently. Then in Section VIII, we illustrate how our protocol can help with building high-level applications such as privacy-preserving neural network training/inference. Meanwhile we provide our implementation details and the benchmark results there. In Section IX, we show the potential of our protocols to be generalized in multiple ways. Finally, we cover the related works in Section X.

II. Solution Overview

We consider computations over a ring $\mathbb{Z}_{2^n}$ that represents the set of integers $\{-2^{n-1}, \ldots, -1, 0, 1, \ldots, 2^{n-1}-1\}$ along with integer addition and multiplication modulo 2^n as the operators. Here, a subset $I \subseteq \mathbb{Z}_{2^n}$ is an *interval* if there are $i, j \in \mathbb{Z}_{2^n}$ such that $I = \{i, i+1, \ldots, j\}$.[1] Let indicator function $\mathbb{1}_I \colon \mathbb{Z}_{2^n} \to \{0,1\}$ represent the *membership functionality* for interval I. For any $x \in \mathbb{Z}_{2^n}$, we have $\mathbb{1}_I(x) = 1$ if $x \in I$; otherwise, $\mathbb{1}_I(x) = 0$. In particular, when $I = N$ is the set of all "negative elements" for an n-bit value, where $N := \{-2^{n-1}, -\left(2^{n-1}-1\right), \ldots, -1\}$, the functionality $\mathbb{1}_N$ tests whether $x \in \mathbb{Z}_{2^n}$ is "less than zero" or not.

A. An Illustrative Example for Secure Less-Than-Zero Computation

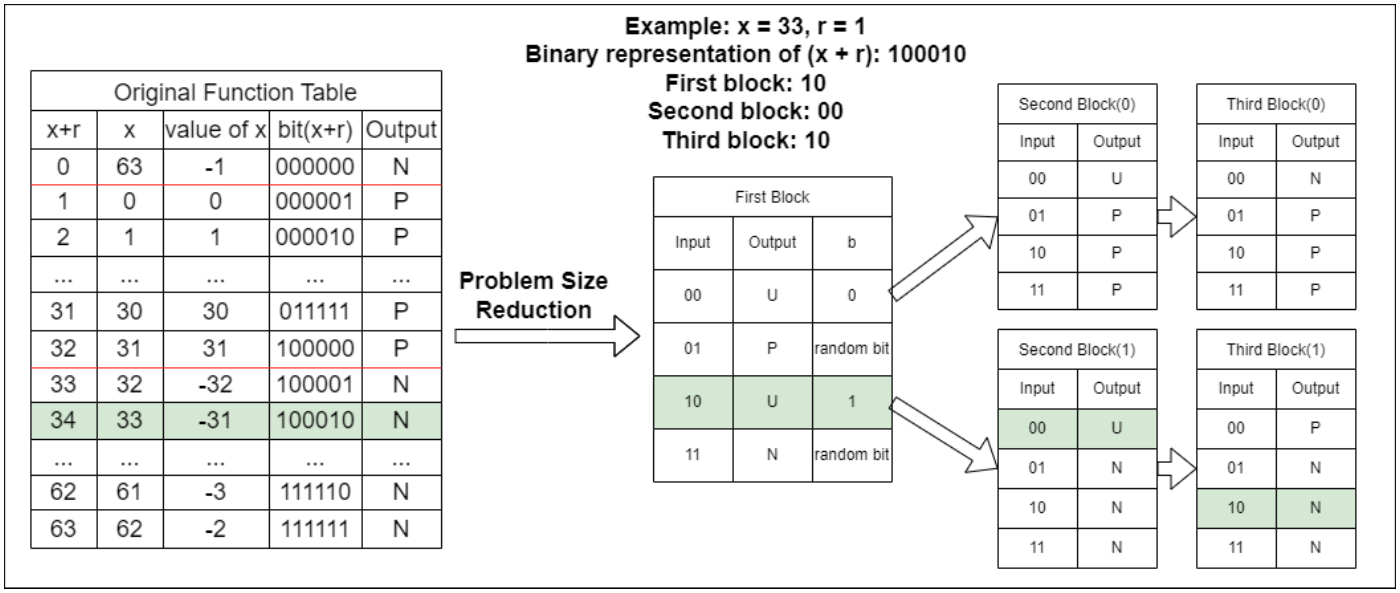
Towards illustrating the solution strategy, we first present a simplified example of secure less-than-zero functionality for two parties (Alice and Bob) for $n = 6$-bit inputs. (See Figure 1.) For ease of exposition, assume that the objective of the parties is to determine the answer $s = \mathbb{1}_N(x)$ in the clear for secret shared input x.

With a uniformly random key $r \leftarrow \mathbb{Z}_{2^n}$ that is hidden from both parties, parties reconstruct the corresponding masked secret $y = (x + r)$. In the example, we consider $x = 33$ and $r = 1$, which results in $y = 33 + 1 = 34$ in the clear. Next, we equivalently need to compute bit s indicating whether the blinded output $y = (x + r)$ is an element of the interval $N + r$ (whether y is in the interval $[33, \ldots, 63, 0]$).

Let $\overrightarrow{y}$ represent the $n = 6$-bit (two's complement) binary representation of the element y (100010). For illustrative purposes, consider the number of blocks $k = 3$ and each block length $\ell = n/k = 6/3 = 2$. For brevity, let $y_1 = \overrightarrow{y}_{1,\ell}$ (the most significant ℓ bits of $\overrightarrow{y}$), $y_2 = \overrightarrow{y}_{\ell+1,2\ell}$ (the middle ℓ-bits of $\overrightarrow{y}$), and $y_3 = \overrightarrow{y}_{2\ell+1,3\ell}$ (the least significant ℓ bits of $\overrightarrow{y}$). For this example, $y_1 = 10$, $y_2 = 00$, $y_3 = 10$.

Level 1 search. We use the first block to perform a level 1 search. Consider the interval $J(y_1) = \{y_1 \| 0^{2\ell}, \ldots, y_1 \| 1^{2\ell}\} \subset \mathbb{Z}_{2^n}$ ($J(10) = \{100000, \ldots, 101111\}$). Intuitively, this interval contains all values that begin with y_1. Let $f_1 \colon \{0,1\}^\ell \to F_3$ be the function defined as follows. If the interval $J(y_1) \subseteq$

[1] Note that, with respect to the $+1$ operator, the elements of $\mathbb{Z}_{2^n}$ embed on a *size-*2^n *circle* $0 \to 1 \to \cdots \to 2^{n-1}-1 \to -2^{n-1} \to \cdots \to -1 \to 0$, a *one-dimensional torus* (informally, a cycle). The mentioned intervals are defined over such a one-dimensional torus. Therefore, if $i = -1$ and $j = 1$, then the corresponding interval is $I = \{-1, 0, 1\}$. For $i = 1$ and $j = -1$, the corresponding interval is $I = \{1, 2, \ldots, 2^{n-1}-1, -2^{n-1}, \ldots, -2, -1\} = \mathbb{Z}_{2^n} \setminus \{0\}$.



Original Function Table

x+r	x	value of x	bit(x+r)	Output
0	63	-1	000000	N
1	0	0	000001	P
2	1	1	000010	P
...	...	...	...	...
31	30	30	011111	P
32	31	31	100000	P
33	32	-32	100001	N
34	33	-31	100010	N
...	...	...	...	...
62	61	-3	111110	N
63	62	-2	111111	N

First Block

Input	Output	b
00	U	0
01	P	random bit
10	U	1
11	N	random bit

Second Block(0)

Input	Output
00	U
01	P
10	P
11	P

Third Block(0)

Input	Output
00	N
01	P
10	P
11	P

Second Block(1)

Input	Output
00	U
01	N
10	N
11	N

Third Block(1)

Input	Output
00	P
01	N
10	N
11	N

Fig. 1. An example of problem size reduction where $\lambda = 6, k = 3, \ell = 2$. The left-hand side is the overall function table before problem size reduction. The right-hand side is the function table structure used in our protocol. "U" means Undetermined, "P" means Positive, and "N" means negative. The red lines in the original function tables indicate the gaps when numbers change the sign bit. If any row in function tables includes the gaps, its result is "Undetermined". A significant observation is that there could be at most 2 such gaps, which is the reason why we only need two tables for all the blocks except the first block.

$N + r$ then $y \in N$, we define $f_1(y_1) = 1$. If the interval $J(y_1) \subseteq \mathbb{Z}_{2^n} \setminus (N + r)$ then $y \notin N + r$ and we define $f_1(y_1) = 0$. However, it is also possible that $J(y_1)$ has non-empty intersections with both $N + r$ and $\mathbb{Z}_{2^n} \setminus (N + r)$, in which case it is *uncertain* whether $x \in N + r$ or not. We define $f_1(y_1) = 2$ indicating this uncertainty. Intuitively, the function f_1 is a *coarse-grained* membership testing functionality (and the granularity parameter ℓ determines the granularity of this search). In this example, we can see that if y_1 is 11 then $f_1(y_1) = 1$, since all values of y starting with 11 are Negative. If y_1 is 01 then $f_1(y_1) = 0$, otherwise $f_1(y_1) = 2$.

Alice and Bob compute the output $a_1 = f_1(y_1)$ securely using the precomputed function table. However, there is a subtlety. Obtaining the answer a_1 in the clear reveals additional information about x, rendering the protocol insecure. In particular, if x is close to either of the end-points of N, then the probability of $a_1 = 2$ is high. However, if x is far from both the end-points of N, then the probability of $a_1 = 2$ is low. Therefore, parties obtain $\widetilde{a}_1 \in F_3$ instead, the masking of a_1 using a random shared r_1 that is hidden from them. In the concrete example, we omit this subtlety, and instead directly present the answering using P, N, and U for Positive, Negative, and Undetermined, respectively.

Observe that there can be at most two values of y_1 such that $f_1(y_1) = 2$. The function $f_1(y_1)$ also needs to produce an *auxiliary information* $b \in \{0, 1\}$, representing whether the uncertainty stems from the inclusion of the starting point of the interval or the end point of the interval. Note that this auxiliary information is masked in our protocol, but we omit the masking in this example for ease of understanding. The utility of this auxiliary information shall become apparent in the discussion below.

Level 2 search. Next, parties need to perform a finer-grained search if $f_1(y_1) = 2$ using the second block of inputs. Alice and Bob continue to the second level of the search while being oblivious to whether $f_1(y_1) = 2$ or not. They test whether the interval $J(y_1 \| y_2) = \{y_1 \| y_2 \| 0^\ell, \dots, y_1 \| y_2 \| 1^\ell\}$ is entirely inside the interval $N + r$, entirely outside the interval $N + r$, or partially intersecting both these sets, indicating uncertainty. We define $a_2 = f_2(y_2, b)$ to be this function. We emphasize that one needs b to reconstruct y_1. Again, parties use precomputed function tables to securely compute $\widetilde{a}_2$, the masking of a_2 with a random share r_2.

In this example, we assume $b = 1$. The parties then look up the entry for $y_2 = 00$ in the table for the second block with $b = 1$. This is testing whether the interval $\{100000, 100001, \dots 100011\}$ is entirely negative, entirely positive, or partially negative and partially positive.

Observe that in this search, irrespective of whether $b = 0$ or $b = 1$, there can be at most one y_2 such that $f_2(y_2) = 2$. This property is essential to ensure that we do not need to generate any *additional auxiliary information* for this search; otherwise, the domain of auxiliary information would have increased exponentially in k (the total number of search levels).

Level 3 search. Alice and Bob continue with their final search, oblivious to the fact whether their previous searches are uncertain or not. They test whether the (singleton) interval $J(y_1 \| y_2 \| y_3) = \{y_1 \| y_2 \| y_3\}$ is inside of outside $N + r$ or not. These cases are exhaustive because a singleton interval cannot be uncertain. Define the function $a_3 = f_3(y_3, b)$ to be this function. Parties securely compute the encryption $\widetilde{a}_3$ (using the secret key r_3) of the output a_3 using precomputed function tables. Note that although we describe these searches in multiple levels, one can perform all k block searches in parallel as long as the *auxiliary information* b is known.

Reconstruction of answer. Given the (masked) answers

$\widetilde{a}_1, \widetilde{a}_2, \widetilde{a}_3$, and using built-in r_1, r_2, and r_3, the aggregation function g, finds the smallest index $i \in \{1, 2, 3\}$ such that $a_i \neq 2$ and $a_j = 2$, for all $j \in \{1, \dots, i-1\}$. Finally, the output of g is $s = a_i$. Parties compute this aggregation function securely using another precomputed function table to obtain the output s.

Intuitively, the reconstruction function inherently accounts for the masking of the intermediate outputs and outputs the first value that is not Undetermined.

Efficiency. Through our approach, the number of rows in all the function tables is 24. As a comparison, if we directly use one single function table, the number of rows is $2^6 = 64$. Therefore, our approach significantly reduces the size of precomputed function table. This effect is larger with the increase of n. (e.g. when $n = 64$, our approach requires 15×2^8 rows, while one single function table requires 2^{64} rows)

B. Protocol Steps of SIM

The above discussed technique extends to any interval I that satisfies $|I| \geqslant 2^{n-\ell}$. [2] We conclude the procedures of secure interval-membership testing below:

1) As $\mathbb{1}_I(x) = \mathbb{1}_{I+r}(y = x + r)$, for any $r \in \mathbb{Z}_{2^n}$, where $I + r$ is the interval $\{x + r \colon x \in I\}$, the precomputation samples an element $r \leftarrow \mathbb{Z}_{2^n}$ uniformly random, hidden from Alice and Bob. Alice and Bob reconstruct $y = x + r$ during the online phase.
2) Let $\overrightarrow{y} \in \{0,1\}^n$ represent the (two's complement) binary representation of the element $y \in \mathbb{Z}_{2^n}$. Let $\overrightarrow{y_1}, \dots, \overrightarrow{y_k} \in \{0,1\}^\ell$ represent the partition of $\overrightarrow{y}$ into k blocks of ℓ-bit strings, i.e., $\overrightarrow{y} = \overrightarrow{y_1} \| \cdots \| \overrightarrow{y_k}$. For $i \in \{1, \dots, k\}$, we determine special functions $f_i \colon \{0,1\}^\ell \to F_3$, and Alice and Bob interactively obtain the encryption of the output $a_i = f_i\left(\overrightarrow{y_i}\right)$ using the secret key r_i, which is unknown to them. The precomputation step establishes the keys $(r_1, \dots, r_k)$ and appropriate precomputations (c.f., [25]) that help in the secure computation of the functions $f_1, \dots, f_k$.
3) Let $\widetilde{a}_1, \dots, \widetilde{a}_k$ represent the encryptions of the intermediate outputs $a_1, \dots, a_k$, respectively, mentioned above. Alice and Bob interactively aggregate the answer $s = g\left(\widetilde{a}_1, \dots, \widetilde{a}_k\right)$ (in a secret-shared manner), where the function $g \colon F_3^k \to \{0,1\}$ depends on the secret keys $r_1, \dots, r_k$, and satisfies the identity $g\left(\widetilde{a}_1, \dots, \widetilde{a}_k\right) = \mathbb{1}_I(x)$. At the end of the online phase, Alice and Bob, respectively, obtain the additive secret shares $[s]_1$ and $[s]_2$ of the secret s.

III. Preliminaries

A. System Model

We consider an asymmetric three-party setting where three parties P_1, P_2, P_3 perform different tasks for the protocol. P_3 is only involved in the offline phase and does not participate in the online phase, similar to [40], [41], [47].

We assume point-to-point authenticated and secure communication channels between the parties. The communication channels are assumed to be bounded-synchronous [3], meaning that the protocol operates in rounds. In each round, parties can perform local computations and send messages. By the end of the round, all parties are guaranteed to receive the messages sent to them in this round. Our protocol also works in the standard offline/online model.

SIM tolerates a semi-honest adversary who can corrupt at most one party. We emphasize that the adversary can corrupt any one of the parties, in particular, the adversary is allowed to corrupt P_3, who is only involved in the offline phase of the protocol. We extend our protocol against the malicious adversary in Section VII.

B. Two-Party Secret Sharing Based MPC

We use a three-party MPC (3PC) setting where two parties participate in the online phase while the third party only produces the offline data and is absent from the online phase. Therefore, our construction closely follows two-party MPC (2PC) protocols for the online phase, and we summarize the relevant 2PC concepts below.

1) Linear Two-Party Secret Sharing: A linear secret-sharing scheme is a secret-sharing scheme where parties only need to perform local operations on their shares to perform linear operations on the secret-shared value. In this work, we use a specific linear secret-sharing scheme, known as additive secret sharing. To share a secret s, the first share $[s]_1$ is generated uniformly at random, and the second share is set such that the sum of the two shares is the secret. To reconstruct the secret s, the two parties add their respective shares together. We can see that additive secret sharing indeed has the linear property. For example, to add two secrets s and t, parties simply need to locally add their share of s to their share of t.

Formally, the dealer shares a secret $s \in \mathbb{Z}_{2^n}$ to two parties P_1, P_2 by generating shares $[s]_1, [s]_2 \in \mathbb{Z}_{2^n}$ such that each party P_i holds $[s]_i$ and $[s]_1 + [s]_2 = s$. We use $[s]$ to denote the secret sharing of s. Here, for public constants $a, b \in \mathbb{Z}_{2^n}$ and secret shared values $[s], [u]$, the following identity holds: $[a \cdot s + b \cdot u] = a \cdot [s] + b \cdot [u]$.

2) MPC based on secret sharing: In two-party secret-sharing-based MPC, the function inputs are secret-shared to the two parties P_1, P_2, where operations are then carried out on the secret shares. When using a linear secret-sharing scheme, the addition of two secret shared values becomes the local addition of the respective shares.

When we need to multiply two values, we can use state-of-the-art techniques [4], [16], which require the parties to communicate with each other to perform the multiplication of two secret shared values.

C. Offline-Online Model and Secure Precomputed Function (SPF) Tables

For our secure comparison construction, we follow the standard offline-online model of MPC, where the computation is separated into an (input-independent) offline phase and an online phase. In the offline phase, input-independent correlated

[2] If $|I|$ is smaller, then both of its endpoints may fall within one uncertain interval of the level 1 search. Given this possibility, one needs to generate auxiliary information in level 2 search because parties are oblivious to whether this event has already occurred or not. This generation of auxiliary information in every level of the search increases the input domains of the search functions and the aggregation function exponentially in k.

secret sharings are generated that are consumed during the online phase.

As the communication and round complexity of the online phase are critical for the performance of an MPC protocol, our goal is to optimize the online phase and try to push more cryptographically expensive operations to the offline phase. In particular, we build upon the secure precomputed function (SPF) table approach introduced by Ishai et al. [25], which offers a very efficient online MPC phase for computing any function.

The intuition behind the SPF protocol is to precompute the function output for all possible inputs to generate function tables in the offline phase, which allows the online phase to be simple function lookups (which is highly efficient). (We formalize the original Ishai et al.'s protocol in Section IV.)

This approach manages the small input domains well; however, the size of the SPF table increases exponentially with the input size, and it is not practical to use for applications using moderate-sized input domains. While our construction uses SPF tables to achieve a highly efficient online phase, as a key novelty, we significantly reduce the function table size.

IV. Formalizing and Extending SPF Compiler

To illustrate how the function table is used in MPC, We first formalize the original (input-hiding) SPF protocol [25] as a compiler. We then introduce our (function-hiding) SPF compiler.

A. Original (Input-Hiding) SPF Compiler

Consider a 2PC with two parties P_1 and P_2 holding inputs x and y respectively, and the parties want to jointly compute $f(x,y)$ for some function f. Here, function $f\colon X_1 \times X_2 \to Y$, where $(X_1,+)$, $(X_2,+)$, and $(Y,+)$ are groups.

The input-hiding SPF compiler [25] takes function f as the input and outputs an MPC protocol that securely realizes the function. In particular, the compiler takes a matrix $\{f(x_1,x_2)\}_{(x_1,x_2)\in X_1\times X_2}$ as input and outputs (i) a 2-party correlation Q_f (so-called the precomputation/offline phase), and (ii) a 2-party online protocol Φ_f. Notice that the compiler depends only on the input-output behavior of the function, and it is *independent* of how the functionality f is realized. To emphasize this property, we use f to mean the input-output behavior of f.

Figure 2 describes the generation of the precomputation Q_f and the online protocol Φ_f. In the offline phase, the offline party prepares a random element r, then it constructs function tables of the input $x+r$ for all x in the input domain. Finally, it secret-shares the SPF table and sends the shared table together with $[r]$ to online parties. In the online phase, online parties add the input $[x]$ and $[r]$ locally, then reconstruct $x+r$. Finally, online parties check the SPF table using $x+r$ to get the final output.

B. Function-Hiding SPF Compiler

Towards developing our efficient protocol, we formalize a variant of the input-hiding compiler [25]. Compared to the input-hiding compiler, our function-hiding compiler also takes one round and has a lower communication cost. This variant takes as input a function from a family of functions, and outputs a secure protocol that does not hide the input to the function, but instead hides the function within the family of functions. We call it the function-hiding compiler and use this variant to construct our secure comparison protocol.

Description of the correlation Q_f.

1) Pick uniformly random offsets $\alpha_1 \in X_1$ and $\alpha_2 \in X_2$
2) Generate $F^{(1)} = \left\{f^{(1)}(x_1,x_2)\right\}_{(x_1,x_2)\in X_1\times X_2}$, where each element $f^{(1)}(x_1,x_2)$ is uniformly and independently chosen from the group Y.
3) Let $F^{(2)} = \left\{f^{(2)}(x_1,x_2)\right\}_{(x_1,x_2)\in X_1\times X_2}$ be the unique matrix defined by the identity $f(x_1,x_2) = f^{(1)}(x_1+\alpha_1, x_2+\alpha_2) + f^{(2)}(x_1+\alpha_1, x_2+\alpha_2)$, for every element $(x_1,x_2) \in X_1 \times X_2$.
4) Send $(\alpha_1, F^{(1)})$ to Party 1 and $(\alpha_2, F^{(2)})$ to Party 2.

Description of the 2-party protocol Φ_f.

Private inputs. For $i \in \{1,2\}$, Party i has private input $x_i \in X_i$.
Correlated private randomness. For $i \in \{1,2\}$, Party i has correlated private randomness $(\alpha_i, F^{(i)})$ given by the correlation Q_f.
Description of the protocol.

1) For $i \in \{1,2\}$, party i broadcasts $x_i' = x_i + \alpha_i$. (Round 1)
2) For $i \in \{1,2\}$, party i outputs $y_i = F^{(i)}(x_1', x_2')$.

Fig. 2. The original compiler's procedure to generate the offline precomputation Q_f and the online protocol Φ_f.

Consider a function $f\colon X \to Y$, where $(Y,+)$ is a group. Our variant takes the matrix $\{f(x)\}_{(x)\in X}$ as input and outputs (i) a 2-party correlation P_f, and (ii) a 2-party online protocol Π_f.

Description of the correlation P_f.

1) Generate $F^{(1)} = \left\{f^{(1)}(x)\right\}_{(x)\in X}$, where each element $f^{(1)}(x)$ is uniformly and independently chosen from the group Y.
2) Let $F^{(2)} = \left\{f^{(2)}(x)\right\}_{(x)\in X}$ be the unique matrix defined by the identity $f(x) = f^{(1)}(x) + f^{(2)}(x)$, for every element $(x) \in X$.
3) Send $(F^{(1)})$ to party 1 and $(F^{(2)})$ to party 2.

Description of the 2-party protocol Π_f.

Public inputs. There is a public input $x \in X$ known by all parties.
Correlated private randomness. For $i \in \{1,2\}$, party i has correlated private randomness $(F^{(i)})$ given by the correlation P_f.
Description of the protocol.

1) For $i \in \{1,2\}$, party i broadcasts $y_i = F^{(i)}(x)$. (Round 1)
2) For $i \in \{1,2\}$, party i outputs $y = y_1 + y_2$.

Fig. 3. Our variant's procedure to generate the offline protocol P_f and the online protocol Π_f.

The generation of the precomputation P_f and the online protocol Π_f is defined in Figure 3. In the offline phase, the offline party prepares the SPF tables for all possible plaintext input x, then secret-shares it to online parties. In the online phases, the online parties simply check the SPF table using the plaintext input, then reconstruct the final output.

V. Secure Comparison Protocol using Precomputed Function Table

As a showcase of SIM, we introduce our protocol of secure comparison in the form of less-than-zero (LTZ) operation. The protocol takes secret shares $[x]$ as the input, and outputs $[s]$ such that $s = f_{LTZ}(x)$.

A. Protocol Overview

The goal of the protocol is to compute the $f_{LTZ}(\cdot)$ for a secret-shared $[x] \in \mathbb{Z}_{2^n}$. First, both online parties compute $[x+r]$ and open it, where r is a random mask generated during the offline phase. Second, they divide the binary representation of $x+r$ into k blocks, where each block has ℓ bits such that $k \cdot \ell = n$.[3]Third, the online parties perform membership testing functions for all the blocks, and get the intermediate outputs, labeled $a_1, a_2, \dots a_k$. The intermediate outputs can either be Positive(P), Negative(N), or Undetermined(U)[4]. As the membership testing function has small input domains, the offline party can precompute the function tables for it such that the online phase of the block membership testing is extremely fast. Finally, the parties open the intermediate outputs and use the intermediate outputs as inputs to the recombination function table to produce the final output.

In the offline phase, the offline party P_3 generates the precomputation and function tables needed for the protocol, and secret shares them to two online parties P_1 and P_2. We have designed multiple sub-functionalities to collaboratively achieve $f_{LTZ}(\cdot)$, and they are explained in detail below.

B. Building Blocks

We start by introducing the building blocks used in our protocol. Then we explain how to use these functions to construct a full protocol.

We use superscripts to denote public parameters of functions, while subscripts denote secret parameters that need to be kept hidden from the online parties. The secret parameters set during the offline phase are $r \in \mathbb{Z}_{2^n}$, $r_1, r_2, \dots, r_k \in F_3$, and $\beta \in \{0,1\}$. r is used to mask the input x, r_i is used to mask the i-th intermediate result, and β is used to mask the selection bit.

The functions $SLTZ^i_{r_i,r,\beta}$ are the coarse-grained membership testing functions that test if a block of input can directly tell us the overall output. $SLTZ^i_{r_i,r,\beta}$ takes public parameter $i \in \{1, 2, \dots, k\}$, which represents which of the k block this function is for, and it is defined as a family of functions over the secret parameters r, r_i, and β.

We start from the first block, the block result is positive if all values that depend on this block (has this block input as prefix) are positive. It is negative if all values that depend on this block are negative, and it is undetermined if some values are positive and some are negative. When the block result of the first block is undetermined, we take the first two blocks as the prefix and check the second block. Since revealing this intermediate result could reveal information on the input, we mask them using the secret parameter r_i.

Formally, we define a function $SLTZ^i_{r_i,r,\beta}(y_1y_2\dots y_\ell, \widetilde{b}) : \{0,1\}^\ell \times \{0,1\} \to F_3$ such that

$$SLTZ^i_{r_i,r,\beta}(y_1\dots y_\ell, \widetilde{b}) = \begin{cases} 0 + r_i, & \text{if } z_0 \notin N + r \text{ and} \\ & z_1 \notin N + r \\ 1 + r_i, & \text{if } z_0 \in N + r \text{ and} \\ & z_1 \in N + r \\ 2 + r_i, & \text{otherwise.} \end{cases}$$

where $\overrightarrow{z_0} = \underbrace{\text{prefix}_{i,r,\beta\oplus\widetilde{b}} \| y_1y_2\dots y_\ell \| 00\dots0}_{n\text{-bits}}$ and $\overrightarrow{z_1} = \underbrace{\text{prefix}_{i,r,\beta\oplus\widetilde{b}} \| y_1y_2\dots y_\ell \| 11\dots1,}_{n\text{-bits}}$ and $\text{prefix}_{i,r,\beta,\widetilde{b}} \in \{0,1\}^{\ell\cdot(i-1)}$.

Additionally, if $z_0 \in N + r$ and $z_1 \notin N + r$, then $\text{prefix}_{i+1,r,0} = \text{prefix}_{i,r,0} \| y_1y_2\dots y_\ell$. If $z_0 \notin N + r$ and $z_1 \in N + r$, then $\text{prefix}_{i+1,r,1} = \text{prefix}_{i,r,1} \| y_1y_2\dots y_\ell$. Besides, we use value 0 in the output above to indicate ”Positive”, 1 as ”Negative”, and 2 as ”Undetermined”.

Next, we explain the function $BLTZ_{r,\beta}$, and the use of the selection bit $\widetilde{b}$ in $SLTZ^i_{r_i,r,\beta}(y_1y_2\dots y_\ell, \widetilde{b})$. We note that the result of Undetermined can only appear in at most two rows of the function table of the first block. The reason is that the block result is Undetermined if and only if the numbers represented by that row contain both positive numbers and negative numbers. As the numbers in the function table are consecutive and in ascending order, we only have two such cases: When a block contains both 0 and -1, and when a block contains both $2^{n-1}-1$ and -2^{n-1}.

Since there are at most two uncertain rows in the first block, we build two function tables for the second block (and all the following blocks), one table for each uncertain row in the first block. The input $\widetilde{b}$ of $SLTZ^i_{r_i,r,\beta}(y_1y_2\dots y_\ell, \widetilde{b})$ is used to indicate if the first group of the tables are used or the second. The goal of $BLTZ_{r,\beta}$ is to determine $\widetilde{b}$ given the first block of the input, such that b can be available for following $SLTZ^i_{r_i,r,\beta}$ executions. We use $\widetilde{b}$ to represent the randomly masked version of b, and $\widetilde{b}$ will be reconstructed during the online phase.

Formally, we define a function $BLTZ_{r,\beta}(y_1y_2\dots y_\ell) : \{0,1\}^\ell \to \{0,1\}$ such that

$$BLTZ_{r,\beta}(y_1y_2\dots y_\ell) = \begin{cases} 0 \oplus \beta, & \text{if } z_0 \in N + r, z_1 \notin N + r \\ 1 \oplus \beta, & \text{otherwise.} \end{cases}$$

Where $\overrightarrow{z_0} = \underbrace{y_1y_2\dots y_\ell \| 00\dots0}_{n\text{-bits}}$ and $\overrightarrow{z_1} = \underbrace{y_1y_2\dots y_\ell \| 11\dots1}_{n\text{-bits}}$.

The $RECOMB_{r_1,r_2,\dots r_k}$ function takes the intermediate outputs produced by $SLTZ^{i,\widetilde{b}}_{r_i,r,\beta}$ and combines them to produce the final output. Intuitively, $RECOMB_{r_1,r_2,\dots r_k}$ outputs the first intermediate output that is not undetermined.

[3] If n is not a multiple of k, then one chooses the k block-lengths such that any two block-lengths are either identical or differ by one. For example, if $n = 8$ and $k = 3$ then block-lengths are $3, 3,$ and 2. For the simplicity of the presentation, this minor detail is omitted; however, our implementation addresses this subtlety while creating the partitions.

[4] The reconstructed intermediate values are randomly masked such that the actual values are kept secret.

Algorithm 1: Less Than Zero **LTZ**($[x]$)

```
Input           : $[x]$
Output          : $[s]$
Pre-computation: , $[r], P_{BLTZ_{r,\beta}}, \left\{P_{SLTZ^i_{r_i,r,\beta}}\right\}_{i\in\{1,2,\dots,k\}}$,
                  $P_{RECOMB_{r_1,r_2,\dots r_k,\tilde{b}}}$
$[y] = [x] + [r]$
$y =$ Open($[y]$) // Round 1
$[\tilde{b}] = \Pi_{BLTZ_{r,\beta}}\left(\overrightarrow{y}_{1,\ell}\right)$
$\tilde{b} =$ Open($[\tilde{b}]$) // Round 2
for $i \leftarrow 1$ to $k$ do
    $[\tilde{a}_i] = \Pi_{SLTZ^i_{r,\beta,r_i}}\left(\overrightarrow{y}_{(i-1)\cdot\ell+1\ ,\ i\cdot\ell}, \tilde{b}\right)$
$(\tilde{a}_1, \tilde{a}_2, \dots \tilde{a}_k) =$ Open($[\tilde{a}_1], [\tilde{a}_2], \dots [\tilde{a}_k]$)
    // Round 3
$[s] = \Pi_{RECOMB_{r_1,r_2,\dots r_k}}(\tilde{a}_1, \tilde{a}_2, \dots \tilde{a}_k)$
return $[s]$
```

Formally, function $RECOMB_{r_1,r_2,\dots r_k}(\tilde{a}_1, \tilde{a}_2, \dots \tilde{a}_k) : F_3^k \to \{0,1\}$ offers

$$RECOMB_{r_1,\dots r_k}(\tilde{a}_1, \dots \tilde{a}_k) = \begin{cases} 0, & \text{if } \exists i \text{ such that} \\ & \quad \forall j < i, (\tilde{a}_j - r_j) = 2 \\ & \quad \text{and } (\tilde{a}_i - r_i) = 0 \\ 1, & \text{otherwise.} \end{cases}$$

C. Protocol Details

We now present how to use the compiler presented in Section IV-B and the functions presented in Section V-B to build a complete protocol for Less Than Zero (LTZ).

1) Online Phase: Algorithm 1 describes the online phase of the protocol. During the online phase, our protocol proceeds in three rounds. In the first round, the online parties P_1 and P_2 compute and open $y = x + r$. Next, online parties divide the binary representation of y into k pieces of ℓ bits each. Parties use the first ℓ bits of y as input to evaluate $\Pi_{BLTZ_{r,\beta}}$ and get the result $[\tilde{b}]$. In the second round, the parties open $[\tilde{b}]$. Then the parties use $\tilde{b}$ and the i^{th} blocks of y as inputs to evaluate $\Pi_{SLTZ^i_{r,\beta,r_i}}$ for all $i \in \{1, 2, \dots, k\}$ and receive shares of the k outputs $\tilde{a}_1, \tilde{a}_2, \dots, \tilde{a}_k$. In the third round, the parties open all the intermediate results $[\tilde{a}_i]$. Using them as inputs, the parties run the online phase $\Pi_{RECOMB_{r_1,r_2,\dots,r_k}}$ to receive secret shares of the final output $[s]$.

2) Offline Phase: During the offline phase, P_3 generates independent and uniformly random values $r \in \mathbb{Z}_{2^n}$, $\beta \in \{0, 1\}$, and r_i for $i \in \{1, 2, \dots, k\}$. Using those values, P_3 selects the corresponding functions from the family of functions. Specifically, P_3 selects $BLTZ_{r,\beta}$, $SLTZ^i_{r_i,r}$ for $i \in \{1, 2, \dots, n\}$, and $RECOMB_{r_1,r_2,\dots r_k}$. P_3 then acts as our variant of the SPF compiler with those functions as inputs and generates the offline correlated randomness specified by the compiler. P_3 then sends the offline correlated randomness to P_1 and P_2, as well as generates secret shares of r to send to P_1 and P_2. Specifically, the offline correlated randomness are $P_{BLTZ_{r,\beta}}$, $P_{SLTZ^i_{r_i,r}}$ for $i \in \{1, 2, \dots, k\}$, and $P_{RECOMB_{r_1,r_2,\dots r_k}}$.

The correctness of our protocol comes from the way the function tables are defined. We highlight a few things related to the correctness of our protocol. Firstly, while some of the intermediate results can be Undetermined, our protocol always outputs a correct result at the end. Due to the structure of the Less Than Zero problem, there are always at most two possible Undetermined values for the first block. Since we account for that by creating two sets of tables for the second block onward, we are essentially dealing with one Undetermined value. For that one Undetermined value of the first block, there can be at most one Undetermined value in the second block since it contains only one point where the values change between positive and negative. Since there is only one point of change, once we reach the last block, we are always able to determine if the value is positive or negative.

VI. Protocol Analysis

A. Cost Analysis and Comparison

Our protocol is technically a 3PC protocol, but the online phase works as a pure 2PC protocol, we present comparisons with state-of-the-art 3PC comparison protocols as well as 2PC protocols.

1) Three-party Computation Protocols: We first present a fully parameterized theoretical performance comparison in Table I, then present a few selected parameters to provide concrete performance numbers in Table II. For our work, the offline phase communication cost consists of three parts: the block function tables (SLTZ), the selection bit table (BLTZ), and the recombination table (RECOMB). BLTZ has $2^\ell = 2^{\frac{n}{k}}$ rows[5], where each row has a one-bit element. Each SLTZ table has $2^\ell = 2^{\frac{n}{k}}$ rows with two-bit elements, and we have in total $2k - 1$ of them (the first block has one SLTZ table, and all following blocks have two SLTZ tables each.). Thus the size of all the SLTZ tables is $(2k - 1) \cdot 2^{\frac{n}{k}} \cdot 2$. RECOMB table has 3^k rows, and each row contains a, n-bit secret sharing. The overall offline precomputation communication/storage cost is given by the summation of the three parts above. As for the online communication cost, the reconstruction of $x + r$ in the first round requires the communication of one n-bit share. The reconstruction of k SLTZ results requires $k \cdot 2$ communication. Besides, the reconstruction of the selection bit takes one-bit communication.

In general, our protocol has the least online communication cost among all recent works. Besides, our protocol has constant round complexity, whereas most recent works required $O(\log n)$ rounds.

To make the numbers concrete in Table II, we present the performance numbers for when inputs are $n = 32$ bits, and for our protocol, we divide the input into $k = 4$ blocks of $\ell = 8$ bits each. We also assume OT takes 2 rounds. We chose these numbers because they provide the most balanced values.

2) Two-Party Computation Protocols: Similarly, we present a fully parameterized theoretical performance comparison in Table III, then select a group of parameters to provide concrete performance numbers in Table IV for various two-party protocols, since our online protocol is designed for two

[5] For simplicity, we replace ℓ with $\frac{n}{k}$ to keep the number of variables to two.

TABLE I. Comparison of Performance For Various three-party Protocols (Assuming inputs are n-bit ring elements, and are divided into k blocks in our protocol. κ is the security parameter of OT used in garble circuit protocols)

	This Work	ABY3* [34]	BLAZE* [37] (arithmetic circuit)	BLAZE* [37] (garbled circuit)	FALCON [45]
(Precomputation) Offline Communication (bits)	$(4k-1)2^{n/k} + n3^k$	$24n$	$9n$	$(5\kappa + 2) \cdot n$	$O(n)$ (Estimated)
Online Communication (bits)	$n + 2k + 1$	$18 \cdot n$	$9n$	κn	$28n$
Online Round	3	$1 + \log(n)$	$1 + \log(n)$	2	$1 + \log(n)$

* The output of ABY3 [34] and BLAZE [37] are not arithmetic sharing. Therefore some additional sharing conversion (costing one or more rounds) is required if their secure comparison results are used in arithmetic circuits.

TABLE II. Comparison of Performance For Various three-party Protocols (with $n = 32$, $k = 4$, $\ell = 8$, $\kappa = 128$)

	This Work	ABY3 [34]	BLAZE (arithmetic circuit) [37]	BLAZE (garbled circuit) [37]	FALCON [45]
(Precomputation) Offline Communication (bits)	6432	768	288	20544	×
Online Communication (bits)	41	576	288	4096	896
Online Round	3	6	6	2	6

* The offline cost of Falcon is omitted as we could not obtain concrete numbers.

online parties. In general, we outperform the recent works in terms of communication cost and achieve the same constant round complexity as all other works.

3) Offline Phase Cost: We also explain the theoretical computation complexity and actual execution time for our offline phase below. As each row of all function tables only requires $O(1)$ computation, the computation cost of the offline phase can be counted by the number of rows of all the function tables. Therefore, we have the computation cost and communication cost to be $O(k \cdot 2^{\ell} + 3^k)$.

B. Security Analysis

We consider a static semi-honest adversary that can corrupt at most one party. Therefore, either the offline party or one of the online parties could be corrupted.

The ideal functionality $\mathcal{F}_{LTZ}$ is defined as follows: $\mathcal{F}_{LTZ}$ receives input x_1 from P_1 and input x_2 from P_2. $\mathcal{F}_{LTZ}$ calculates $x = x_1 + x_2$. $\mathcal{F}_{LTZ}$ computes s such that $s = 1$ if x is less than zero (in the set $\{-2^{n-1}, -(2^{n-1}-1), \ldots - 1\}$, and $s = 0$ otherwise. $\mathcal{F}_{LTZ}$ sends s to P_1 and P_2. [6]

Theorem 1. *Our LTZ protocol securely realizes the ideal functionality $\mathcal{F}_{LTZ}$ in the presence of a PPT adversary $\mathcal{A}$ who can corrupt at most one party as semi-honest.*

Proof: To prove this, it suffices to prove Lemma 2 and Lemma 1. ■

Lemma 1. *Our LTZ protocol is secure in the presence of a PPT adversary $\mathcal{A}$ who corrupts P_3 as semi-honest.*

Proof: Since P_3 is only involved in the input-independent offline phase, adversary $\mathcal{A}$ gains no information. ■

Lemma 2. *There exists a PPT Simulator $\mathcal{S}$ that can simulate the adversary $\mathcal{A}$'s view in LTZ when $\mathcal{A}$ corrupts either P_1 or P_2 such that the simulated view is indistinguishable from the view of the real execution.*

Without loss of generality, let $\mathcal{A}$ be a probabilistic polynomial time (PPT) adversary who corrupts P_1 (since P_1 and P_2 are symmetric). We construct a PPT simulator $\mathcal{S}$ that simulates the adversary $\mathcal{A}$'s view.

To simulate the offline phase, the simulator $\mathcal{S}$ generates uniformly random values as entries for P_1's share of the various function tables as well as for P_1's share of r and sends them to $\mathcal{A}$. Note that $\mathcal{S}$ records P_1's share of r as r'.

To simulate the online phase, in the first round, $\mathcal{S}$ samples a uniformly random value as $[y]_2$ and sends it to $\mathcal{A}$ to simulate opening $[y = x + r]$. At the same time, $\mathcal{S}$ receives $[y]_1$. $\mathcal{S}$ can recover the input $[x]_1$ of P_1 by computing $[x]_1 = [y]_1 - r'$ where r' is recorded from the precomputation step. In the second round, $\mathcal{S}$ samples a uniformly random bit as $[\widetilde{b}]_2$ and sends it to $\mathcal{A}$ to simulate opening $[\widetilde{b}]$. In the third round, $\mathcal{S}$ samples k uniformly random element from F_3 as $[\widetilde{a}_1]_2, [\widetilde{a}_2]_2, \ldots [\widetilde{a}_k]_2$ and records and sends them to $\mathcal{A}$ to simulate opening $[\widetilde{a}_i]$. At the same time, $\mathcal{S}$ receives $[\widetilde{a}_1]_1, [\widetilde{a}_2]_1, \ldots [\widetilde{a}_k]_1$.

Using $[\widetilde{a}_1]_1, [\widetilde{a}_2]_1, \ldots, [\widetilde{a}_k]_1$ and $[\widetilde{a}_1]_2, [\widetilde{a}_2]_2, \ldots [\widetilde{a}_k]_2$, $\mathcal{S}$ computes $a_1 = [\widetilde{a}_1]_1 + [\widetilde{a}_1]_2, a_2 = [\widetilde{a}_2]_1 + [\widetilde{a}_2]_2, \ldots \widetilde{a}_k = [\widetilde{a}_k]_1 + [\widetilde{a}_k]_2$. $\mathcal{S}$ now looks at the function table $RECOMB$ it sent to $\mathcal{A}$ during the offline phase and locates the output of $\widetilde{a}_1, \widetilde{a}_2, \ldots \widetilde{a}_k$ in the function table, labeling it as $[s]_1$.

To simulate opening $[s]$, $\mathcal{S}$ first sends the input $[x]_1$ of P_1 to the ideal functionality, and receives the output s^*. $\mathcal{S}$ can then calculate $[s]_2$ such that $[s]_2 = s^* - [s]_1$, and send it to $\mathcal{A}$ to simulate opening $[s]$.

We now show that this simulation is indistinguishable from the real protocol execution. Since the precomputation that P_1 receives in the real execution are all secret shares of values, they are all independent and uniformly random, which has identical distribution to what $\mathcal{S}$ generates for precomputation.

[6] For the purpose of the proof, let us assume that Party 1 and Party 2 open the secret-sharing of the output s immediately. Note that this ensures that even the output distribution is indistinguishable between the simulated execution and real execution, and the proof still holds if the parties do not open the output and the ideal functionality sends the secret shares to the parties.

TABLE III. Comparison of Performance For Various two-party Protocols (Assuming inputs are n-bit ring elements, and are divided into k blocks with each block being ℓ bits in our protocol. κ is the security parameter of oblivious transfer used in garble circuit protocols)

	This Work	Garbled Circuit [24]	ABY [16]	ABY2.0 [36]
(Precomputation) Offline Communication	$(4k-1)2^{n/k} + n3^k$	κ	$4n\kappa$	$4n\kappa + n$
Communication	$n + 2k + 1$	$n\kappa$	$2n\kappa + n$	$2n\kappa$
Round	3	2	2	2

* The output of ABY [16] and ABY2.0 [36] are not arithmetic sharing. Therefore some additional sharing conversion (costing one or more rounds) is required if their secure comparison results are used in arithmetic circuits.

TABLE IV. Comparison of Performance For Various two-party Protocols (with $n = 32$, $k = 4$, $\ell = 8$, $\kappa = 128$)

	This Work	Garbled Circuit [24]	ABY [16]	ABY2.0 [36]
(Precomputation) Memory Usage	6432	128	16384	16416
Communication	41	4096	8224	8192
Round	3	2	2	2

In the first round, since r is generated uniformly at random in the real execution, $x + r$ is also uniformly at random, and the simulated view has identical distribution to the real execution. In the second round, since β is generated uniformly at random in the real execution, $\widetilde{b}$ is uniformly random, and the simulated view has identical distribution to the real execution. We highlight the nontriviality that since only one value $\widetilde{b}$ is opened from the function tables of $BLTZ_{r,\beta}$, the value of $\widetilde{b}$ is independent of anything in the transcript prior to opening $\widetilde{b}$. In the third round, since r_is are generated uniformly at random in the real execution, each $\widetilde{a}_i$ is also uniformly at random, and the simulated view has identical distribution to the real execution. We once again highlight the nontriviality that only one value from each $SLTZ^i_{r,\beta,r_i}$ is opened, ensuring that each r_i is effectively only used once, making all $\widetilde{a}_i$ independent and uniformly random, which is indistinguishable for the real execution and the simulated view.

Since the final output $[s]$ is a secret share, it is indistinguishable from a value sampled from the uniform distribution. Note that the $\mathcal{S}$ is able to simulate the correct output with random shares, which means the joint distribution of the output and the view is identical for the real execution and the simulation.

We note that this proof is not limited to only LTZ, and can be generalized to other functions mentioned in Section IX-A.

VII. Malicious Security

We now present how we can extend our protocol to achieve malicious security.

A. System Model

Malicious security can be achieved in a two-online-party setting if we apply the same methods provided in [25], however, it requires a decent amount of additional cost because of the usage of message authentication code (MAC). Therefore, to achieve a highly-efficient malicious secure protocol, we require a different system model, where one offline party and **three** online parties are needed. We assume there exists an adversary that can control up to one party. In this honest-majority setting, we can directly use replicated secret sharing to achieve secure-with-abort online phase at no additional cost. This model can be extended such that there are $n \geqslant 3$ servers and $t < \frac{n}{2}$ corrupted servers.

B. Offline Phase

The offline phase of our protocol can be converted to maliciously secure using the standard cut-and-choose method [30], [31]. Essentially, the offline party prepares μ sets of precomputations, where μ is the statistical security parameter. As an additional step to the offline phase, the online parties randomly choose $\mu - 1$ sets of precomputation and open them by sending all values and tables from those $\mu - 1$ sets to each other. The online parties can then locally verify that the function tables and values are generated according to protocol specifications. If any of the sets of precomputation fails the check, the online parties abort and potentially issue punishments towards the offline party. The probability that the offline party can generate an inconsistent offline phase and not be detected is $2^{-\mu-1}$.

C. Online Phase

With the offline phase with the cut-and-choose procedure, we assume that we have one set of precomputed values and that all precomputation is performed according to the protocol specifications.

Since our protocol is in the honest majority setting, it can easily be transformed to tolerate a malicious adversary that controls at most 1 party. For example, we can use the same replicated secret sharing scheme used in FALCON [45], which will provide security with abort against malicious adversaries. The parties can simply check if the replicated shares sent by the other two parties are consistent or not. If not, the parties simply abort. Such a transformation does not affect the cost of our protocol beyond the natural increase in cost due to the maliciously secure framework. For example, replicated secret sharing will require two shares to be sent instead of one during reconstruction but does not otherwise affect our protocol's performance.

We implement our protocols using FALCON. We refer the readers to Section VIII-C for detailed benchmark and performance analysis.

D. Security Analysis

In this section, we informally argue about the security against the malicious adversary.

Theorem 2. *Assuming the existence of a malicious-secure replicated secret sharing scheme, Our LTZ protocol securely realizes the ideal functionality $\mathcal{F}_{LTZ}$ in the presence of a PPT adversary $\mathcal{A}$ who can corrupt at most one party as malicious under the secure-with-abort setting.*

Following the same idea mentioned in Ishai et al. [25], we need to ensure that a malicious party does not choose to report a secret share from a wrong entry in the function table. By using the malicious-secure replicated secret sharing scheme, the honest parties can detect malicious behavior by reporting inconsistencies between the secret shares of the malicious party and the honest parties. At which point the honest parties can abort the protocol. Then a similar proof to the semi-honest security proof (Theorem 1) applies. Together, our protocol achieves malicious security under the secure-with-abort setting.

VIII. High Level Application: Privacy-preserving Neural Network Training/Inference

To illustrate the performance of our protocols, we develop prototypes for our secure comparison protocols. Additionally, to demonstrate the effect of our protocol in a real-world application, we choose to implement our protocol for the privacy-preserving neural network training/inference application.

The state-of-the-art works in this area are Falcon [45] and SecureNN [44]. We observe that the main bottleneck is the evaluation of the ReLU activation function, where the secure comparison protocol is the core building block. In addition to ReLU, the Division and Maxpool functions have high costs, and the secure comparison is also the main reason. Therefore, we think the secure comparison is one of the main bottlenecks, meaning that the use of our protocol should provide a significant performance improvement. We choose to implement our protocol in C++ and embed it into the Falcon [45] framework by replacing the derivative of ReLU (Falcon's secure comparison implementation) with our protocol.

A. Implementation details

As our protocol has two online parties whereas Falcon has three online parties, we need to properly embed our protocol into the Falcon framework. To begin with, Falcon uses replicated secret sharing. If we only consider the first two parties P_1 and P_2, they actually hold a two-party additive secret sharing of the input. Therefore, we can run our secure comparison on P_1 and P_2 without any other changes. However, we still need to make sure the final output s is a three-party replicated share (e.g. $s = s_1 \oplus s_2 \oplus s_3$, and each party P_i holds s_i and s_{i+1}). To achieve this, we allow the offline party P_3 to generate s_3 and s_1 in advance and take them as the final shares of P_3. In the offline phase, P_3 sends s_1 to P_1 and s_3 to P_2. In the final recombination table, the shares of s_2 are stored instead of the shares of the final output in our original protocol. Therefore, P_1 and P_2 need an additional round to reconstruct s_2 using the shares.

We can think of the LTZ as a black box, Falcon provides 3-party replicated secret sharings as the input to LTZ. Inside the box, LTZ operates with 2-party additive secret sharing, and the output of LTZ is a 3-party replicated secret sharing again. The security of this construction is straightforward in a semi-honest setting.

B. Evaluation Results

To compare our protocol against the secure comparison used by Falcon, we choose to evaluate the performances on Network A and Network C from Falcon. Network A is a 3 layer fully connected network with the ReLU activation function after each layer. Network C is a 4 layer network with 2 convolutional and 2 fully-connected layers. This network also uses both Max Pooling and ReLU.

To begin with, we run the micro-benchmark that only considers the performance of the secure comparison. The micro-benchmark is launched in AWS clusters using three t3.2xlarge instances (8 cores and 32GB RAM). The three instances are located in different regions and have an average round trip time of around 130ms. We call this testing environment the distributed setting" for the rest of the paper. The experiments show that our LTZ takes 0.790 seconds and 0.163MB of data transmission per party to do 128 secure comparisons. Meanwhile, falcon's secure comparison takes 1.353 seconds and 0.606MB bandwidth. So we can expect a 1.7× efficiency improvement in running time and 4× improvements in communication here.

Then we run some neural network tests in the distributed setting. We tested two versions of the codebase: the first one is the original Falcon codebase, and the other is the Falcon codebase only with the LTZ function replaced by our protocol. Therefore all the performance difference is caused by the replacement of the LTZ function. For the neural network training, we run 15 forward-backward pass iterations just to show the performance difference. The benchmark result is available in Table V. In general, the neural network training time with our LTZ is around 30% more efficient. For the inference, the running time is improved at around 1.4×. The communication of our protocol is also significantly cheaper than Falcon.

We also provide the execution time and communication of the offline phase in Table VI. The result shows that the cost is acceptable for neural network use cases.

a) Tuning Parameters for Better Performance: Our protocol is highly flexible since the parameters such as k and ℓ are all flexible to change, and different parameter combinations lead to different offline/online performances. Therefore, the users of our protocol can pick the proper parameters to fit into their application. As an illustrative example, we test the offline/online performance of different k, ℓ combinations using the same testing environment. Notice that when we fix k, the best offline phase performance can be achieved if the size of all the blocks are the same (or closer to each other). The reason is that the size of the recombination table is fixed with k, and the offline phase performance is bottlenecked by the largest block table. The size of the largest block table is most optimized if we set the sizes of all blocks to be the same. We follow this setting in our experiments and only include parameter k in the benchmark figure.

Figure 4 shows the offline phase performance with different parameter settings. When k is very small, the size of each

TABLE V. Online Phase Benchmark: privacy-preserving neural network training/inference with our LTZ vs Falcon's LTZ. The experiments are run using AWS t3.2xlarge instances with 130ms ping. Communication is measured by the total MB sent per party. parameter setting: $n = 32, k = 4, \ell = 8$.

Network	Mode	Online time(s)		Communication (MB)	
		This work	Falcon	This work	Falcon
Network A	training inference	6.21s 1.83s	10.14s 3.16s	22.61MB 0.65MB	33.98MB 1.57MB
Network C	training inference	22.45s 11.45s	38.94s 22.39s	584.21MB 79.392MB	1123.35MB 199.949MB

TABLE VI. Offline Phase Benchmark for $n = 32, k = 4, \ell = 8$. (Assuming inputs are n-bit ring elements, and are divided into k blocks with each block being ℓ bits in our protocol.)

Network	Mode	#Comparisons	Execution Time	Communication
Network A	training inference	606720 34048	780s 44s	488 MB 28 MB
Network C	training inference	19968000 1324800	7.16 hr 1688s	16.1 GB 1.1 GB

* The benchmark is executed in AWS clusters with t3.2xlarge instances.

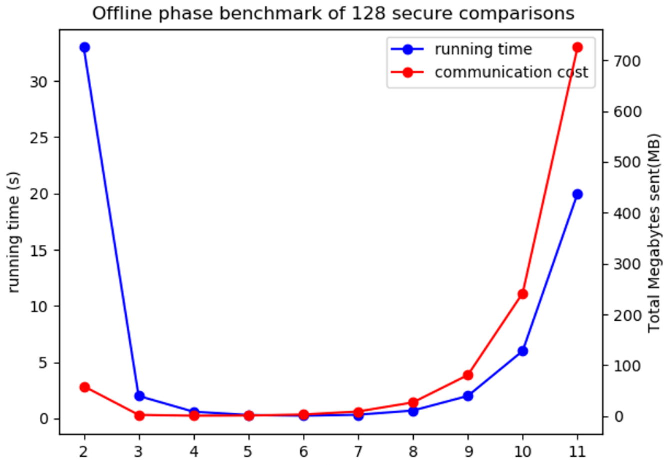


Fig. 4. Offline phase benchmark with different parameter settings ($n = 32$, k is the number of blocks). The experiments are run using AWS t3.2xlarge instances with 130ms ping.

block function table is large, thus the offline phase is costly. Similarly, when k is very large, the size of the recombination table is large, leading to an expensive offline phase. Therefore, if the users pursue a more efficient offline phase, an intermediate k is a good option. Figure 5 illustrates the online phase performance. It can be seen that the online performance increases linearly with k. The reason is that the communication complexity and computation complexity of our protocol are both linear with k. Therefore, the users can pick the smallest k that they can afford (in terms of the offline phase) to get the best online phase performance. As for the overall performance including both the online and offline phases, the pattern is almost the same as the offline phase since the offline phase is the main bottleneck taking around 90% to 95% of the overall cost.

C. Implementation and Evaluation of the Malicious Secure Version of Our Protocol

We implement the malicious secure version of our protocol using FALCON [45] in a three-online-party setting. As we only need share reconstructions in the online phase, we directly use the replicated secret sharing and the corresponding malicious-secure share reconstruction functions provided by FALCON. Accordingly, the offline party generates function tables with replicated secret sharing as the function table output.

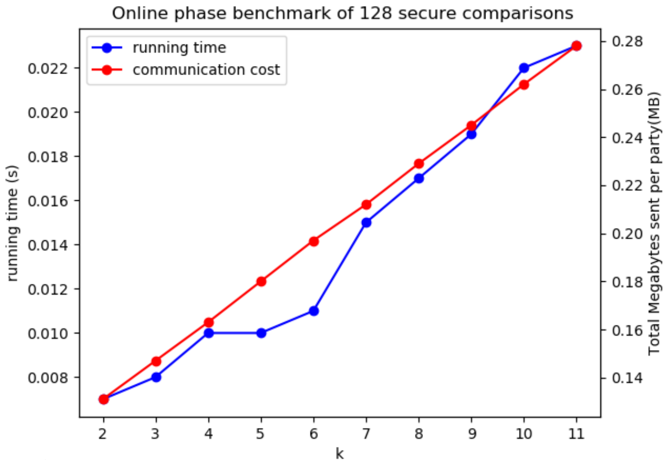


Fig. 5. Online phase benchmark with different parameter settings ($n = 32$, k is the number of blocks). The experiments are run using AWS t3.2xlarge instances with 130ms ping.

We test the performance of the protocols in AWS, where three servers are instantiated as c5.9xlarge (36 cores and 72GB RAM) instances in the same region. For illustrative purposes, we test the performance of both training and inference with Network A. The benchmark result is shown in Table VII. Compared with the semi-honest case, the main difference is the communication cost, which increases significantly in both our protocols and Falcons protocols. However, we see that our protocol achieves a larger performance gain than the semi-honest case. The communication cost of our protocol is around 5× cheaper than FALCON in a malicious setting, while in the semi-honest case it is only around 2×. The reason is that the malicious building block significantly increases the cost of secure comparison, and because our secure comparison protocol requires less reconstruction, the performance gain becomes inherently larger. To conclude, as the MPC building blocks (e.g. malicious secure share reconstruction) become more expensive, the advantage of our protocols will be more obvious.

IX. Generalization

A. General Functions

The idea of problem size reduction could be generalized to solve more problems beyond less-than-zero. Since we are

TABLE VII. Malicious-security benchmark: privacy-preserving neural network training/inference with our LTZ vs Falcon's LTZ. The experiments are run using AWS c5.9xlarge instances located in the same region. Communication is measured by the total MB sent per party. parameter setting: $n = 32, k = 4, \ell = 8$.

Network	Mode	Online time(s)		Communication (MB)	
		This work	Falcon	This work	Falcon
Network A	training	3.83s	5.18s	58MB	216MB
	inference	0.218s	0.298s	1.6MB	10.5MB

using function tables for blocks of input, our protocol can be generalized to a class of functions, which we define as *block determinable functions*.

We define block determinable functions as functions where the overall output of the function is determined by first examining the inputs block by block. More formally, it satisfies the following identity : $f(x_1 \| x_2 \| \cdots \| x_k) = h\left(g_1(x_1), g_2(x_2), \ldots, g_k(x_k)\right)$.

Furthermore, to achieve better performance, the function should have an additional property where either the functions g_is have small ranges (preferably one to two bits), or the function h can be performed efficiently through MPC, e.g. secure addition. In addition to Less Than Zero, Hamming distance and parity are also block determinable functions with significant potential for impact through their privacy-preserving applications.

Note that the above block determinable functions follow a strategy similar to the map-reduce paradigm, where functions g_is resembles mapping and function h resembles reduction operation.

We elaborate on two example functions that can be solved by our protocols, secure hamming distance evaluation and secure equal to zero evaluation.

1) Secure Hamming Distance: Consider a setting when online parties P_1 and P_2 hold two secret shared bit strings of the same length $X = (x_1 x_2 \cdots x_n)$ and $Y = (y_1 y_2 \cdots y_n)$. These two strings are represented as ring elements in Z_{2^n}. The hamming distance is defined as $h(X, Y) = \sum_{i=1}^{n} x_i \bigoplus y_i$.

If we build up one precomputed function table to solve this problem, the size of function table is $2^n \times 2^n$ since the function $h(X, Y)$ has two inputs of size 2^n. The size of this table is too large to be used in practice when n is large.

This efficiency problem can be solved by applying the problem size reduction here. Similarly, in the online phase, we can reconstruct randomly masked X and Y and divide them into k pieces where each piece is ℓ bit, such that $n = k \times \ell$. Then we build function tables for each piece so that each function table returns the hamming distance of the corresponding piece. The size of each table is 2^l and there are in total k function tables.

For hamming distance, there is no need to build up a recombination table to combine block results, since the recombination steps can be done by easily summing up all the block results.

If we set $k = \ell = \sqrt{n}$, then the problem size reduction reduces the offline cost from $O(2^n)$ to $O(\sqrt{n} \times 2^{\sqrt{n}})$. Assuming $n = 64$, this is a $2^{53}\times$ improvement.

2) Secure Equal to Zero: Equal to Zero can be seen as a special case of the Less Than Zero function. Instead of having 3 possible intermediate outputs (Positive, Negative, Unknown), EQZ will have two possible intermediate outputs (True and False). If a block directly determines that the value is not equal to zero, the intermediate output will be False. A value is overall equal to zero if all blocks return True. While we may still need a recombination table, since the intermediate output space is reduced, our recombination table will only have 2^k instead of 3^k rows.

B. Going Beyond Two Online Parties

In this section, we describe how to turn our protocol from a 3-PC protocol to include more online parties. In the original protocol, both online parties execute the same computations, the only difference is that the contents of their function tables are different. Therefore, we can leverage it to extend our protocol to accommodate more online parties. If we consider n-online parties, the offline party will generate function tables for each online party, the only difference is that the output of function tables will be (n, n) additive secret sharing instead of 2-party secret sharing[7]. The online parties will follow the same online phase, and the only difference is that they will use (n, n) secret sharing scheme to reconstruct the secrets. In this case, the offline party becomes a single point of failure, as the privacy is broken if the adversary knows the randomness generated by the offline party. Therefore, we can only use this idea when there is a trustworthy third party who is willing to take care of the offline phase.

As an alternative, in some applications, we can also let the clients who provide private inputs generate the precomputation. If the clients don't generate correct precomputation or leak it to the adversary, the privacy/correctness of their own data will be broken, thus they have no motivation to become a "corrupted offline party". In this setting, we don't need an offline party anymore and the precomputation will be taken as part of the inputs of the MPC protocol. For instance, this idea could be used in a case where the online parties hold a neural network model privately in the form of secret sharing, and clients want their private input to be evaluated on the private neural network.

X. Related Works

In [17], Escudero et al. present an efficient secure comparison protocol through the use of novel precomputation values called edabits. The communication complexity of the protocol is $O(n)$, and in our case, we only need to reconstruct approximately two n-bit ring elements. Additionally, their round complexity is logarithmic while our protocol is constant round. An alternate way of using edabits is briefly mentioned, in

[7] (n, t) ring-based secret sharing could also be used.

which edabits are used to convert an arithmetic secret sharing to a garbled circuit setting. While this reduces the round complexity, it greatly increases the communication complexity due to the use of garbled circuits.

In [33], Makri et al. uses an adaptation of the BIT-LT protocol by Damgård et al. [11] and edaBits [17] to compare a secret-shared value with a public value (Less Than Constant protocol). Such a comparison protocol requires $\log n{+}2$ rounds when our protocol requires only constant rounds. Additionally, Makri et al. presented an optimized protocol for LTZ (presented as ReLU). However, the optimized LTZ protocol still requires $\log n$ rounds.

Ryffel et al. [39] extend [9] to perform secure comparison with similar performance using function secret sharing. Compared with them, our protocol provides both the semi-honest version and the malicious secure version while AriaNN only realizes solutions for the semi-honest setting. Besides, their construction achieves computational security while our solution is perfectly secure. Additionally, their protocol has a non-zero error rate. Although in their setting, the error rate is low enough to not significantly affect their application, the error rate may be larger and more significant for other applications and settings. In comparison, our protocol always outputs a correct answer.

There are also plenty of research focusing on privacy-preserving machine learning, thus the secure comparison protocol is one of the key building blocks in these works. For instance, ABY3 [34] uses bit extraction to get the most significant bit of an element, which costs $O(log(n))$ rounds and $O(n)$ reconstructions. Many works afterward such as SWIFT [27] follow this method to do secure comparison. Additionally, SWIFT reports an amortized online communication cost of $9n-6$ bits, which is also higher than our protocol.

ABY [16] and ABY2.0 [36] make use of a parallel prefix adder (PPA) to achieve the extraction of the most significant bit. The circuit itself has $O(\log n)$ depth, thus constant round complexity can only be achieved through the garble circuit method. As a result, ABY2.0 designs efficient protocols to transfer secret sharing from arithmetic form and garble circuit form efficiently. However, this secure comparison requires more communication than our protocol due to the use of the garble circuits. Besides, our protocol works only in the arithmetic world, therefore it can be easily generalized to include more online servers. This generalization is not straightforward for ABY2.0.

Damgård et al. [13] present some efficient building blocks for privacy-preserving machine learning. Their secure comparison protocol follows a similar idea as edabits [17], where a random share and its bit-wise shares are used to accelerate the computation. Their protocol has $O(log(n))$ round complexity as a bit-wise less than operation is required. Compared to their work, our protocol only takes three rounds for arbitrary size inputs.

Most relevant to our work, SecureNN achieves secure comparison through the use of share conversion between the original even ring to an odd ring in order to extract the most significant bit (MSB) of a value, which directly determines its sign. This construction requires opening $O(n)$ secret sharings in the online phase, thus it is outperformed by our protocol in communication cost. Falcon performs secure comparison (called Derivative of ReLU in their paper) through the use of wrap functions and local computations. The wrap function is essentially a function that computes the carry bit when shares are added together. For a n bit input, both SecureNN and Falcon require $O\left(\log n\right)$ rounds and $O\left(n\right)$ bits of communication.

Tetrad [28] follows the direction of ABY2.0, where they construct a mixed-protocol framework and use a garble circuit to extract the most significant bit. Compared with their solution, our protocol is built in pure arithmetic circuits, thus no share transfer is needed. Meanwhile, our protocol can be generalized to n parties much more easily.

XI. Conclusion

In this work, we have proposed SIM, a secure interval testing protocol that leverages function tables to achieve both an efficient online phase and a practical offline phase. As the first step, we introduced the abstract functionality of testing membership in an interval. Similar to the map-reduce methodology, we have presented a solution approach for partitioning input into input blocks, computing a sequence of intermediate functionalities on those input blocks, and finally securely aggregating the output from these intermediate outputs. We have illustrated that our protocol significantly improves the performance of high-level applications such as privacy-preserving machine learning.

As a step towards generalization, we have defined a notion of block determinable functions and also proposed an approach for going beyond two online parties. We believe the proposed techniques have the potential to be applied to other MPC problems as well as generalized to further settings and may be of interest independent of the less-than-zero function.

References

[1] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. QUOTIENT: Two-party secure neural network training and prediction. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 1231–1247. ACM Press, November 11–15, 2019. 1

[2] Abdelrahaman Aly and Sara Cleemput. An improved protocol for securely solving the shortest path problem and its application to combinatorial auctions. Cryptology ePrint Archive, Report 2017/971, 2017. https://eprint.iacr.org/2017/971. 1

[3] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 590–609, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany. 4

[4] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991. 1, 4

[5] Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation. In Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015: 19th International Conference on Financial Cryptography and Data Security*, volume 8975 of *Lecture Notes in Computer Science*, pages 227–234, San Juan, Puerto Rico, January 26–30, 2015. Springer, Heidelberg, Germany. 1

[6] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis. In *International Conference on Financial Cryptography and Data Security*, pages 57–64. Springer, 2012. 1

[7] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *FC 2009: 13th International Conference on Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343, Accra Beach, Barbados, February 23–26, 2009. Springer, Heidelberg, Germany. 1

[8] 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 Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 871–900, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. 1

[9] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 341–371, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany. 1, 13

[10] Geoffroy Couteau. New protocols for secure equality test and comparison. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18: 16th International Conference on Applied Cryptography and Network Security*, volume 10892 of *Lecture Notes in Computer Science*, pages 303–320, Leuven, Belgium, July 2–4, 2018. Springer, Heidelberg, Germany. 1

[11] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, pages 285–304. Springer, 2006. 13

[12] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18, Egham, UK, September 9–13, 2013. Springer, Heidelberg, Germany. 1

[13] Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure mpc over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1102–1120, 2019. 1, 13

[14] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In Radu Sion, editor, *FC 2010: 14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159, Tenerife, Canary Islands, Spain, January 25–28, 2010. Springer, Heidelberg, Germany. 1

[15] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. 2

[16] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *ISOC Network and Distributed System Security Symposium – NDSS 2015*, San Diego, CA, USA, February 8–11, 2015. The Internet Society. 1, 2, 4, 9, 13

[17] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for mpc over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 823–852, Cham, 2020. Springer International Publishing. 12, 13

[18] David Evans, Vladimir Kolesnikov, and Mike Rosulek. 2018. 1

[19] Irene Giacomelli, Somesh Jha, Ross Kleiman, David Page, and Kyonghwan Yoon. Privacy-preserving collaborative prediction using random forests. *CoRR*, abs/1811.08695, 2018. 1

[20] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press. 1

[21] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Concretely efficient large-scale MPC with active security (or, TinyKeys for TinyOT). In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 86–117, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany. 1

[22] Brett Hemenway, Steve Lu, Rafail Ostrovsky, and William Welser IV. High-precision secure computation of satellite collision probabilities. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16: 10th International Conference on Security in Communication Networks*, volume 9841 of *Lecture Notes in Computer Science*, pages 169–187, Amalfi, Italy, August 31 – September 2, 2016. Springer, Heidelberg, Germany. 1

[23] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *ISOC Network and Distributed System Security Symposium – NDSS 2012*, San Diego, CA, USA, February 5–8, 2012. The Internet Society. 1

[24] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security 2011: 20th USENIX Security Symposium*, San Francisco, CA, USA, August 8–12, 2011. USENIX Association. 9

[25] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 600–620, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany. 1, 2, 4, 5, 9, 10

[26] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018: 27th USENIX Security Symposium*, pages 1651–1669, Baltimore, MD, USA, August 15–17, 2018. USENIX Association. 1

[27] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. SWIFT: Super-fast and robust privacy-preserving machine learning. Cryptology ePrint Archive, Report 2020/592, 2020. https://eprint.iacr.org/2020/592. 13

[28] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. Tetrad: Actively secure 4pc for secure training and inference. In *Proceedings 2022 Network and Distributed System Security Symposium*. Internet Society, 2022. 13

[29] Andrei Lapets, Nikolaj Volgushev, Azer Bestavros, Frederick Jansen, and Mayank Varia. Secure mpc for analytics as a web application. In *2016 IEEE Cybersecurity Development (SecDev)*, pages 73–74, 2016. 1

[30] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 1–17, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. 9

[31] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of Cryptology*, 25(4):680–722, October 2012. 9

[32] Donghang Lu, Albert Yu, Aniket Kate, and Hemanta Maji. Polymath: Low-latency mpc via secure polynomial evaluations and its applications. *Proceedings on Privacy Enhancing Technologies*, 2022(1):396–416, 2022. 1

[33] Eleftheria Makri, Dragos Rotaru, Frederik Vercauteren, and Sameer Wagh. Rabbit: Efficient comparison for secure multi-party computation. In *FC 2021: 25th International Conference on Financial Cryptography and Data Security*, Virtual, March 1–5, 2021. Springer, Heidelberg, Germany. 1, 13

[34] Payman Mohassel and Peter Rindal. ABY^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 35–52, Toronto, ON, Canada, October 15–19, 2018. ACM Press. 1, 2, 8, 13

[35] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press. 1

[36] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved mixed-protocol secure two-party computation. Cryptology ePrint Archive, Report 2020/1225, 2020. https://eprint.iacr.org/2020/1225. 1, 2, 9, 13

[37] Arpita Patra and Ajith Suresh. BLAZE: Blazing fast privacy-preserving machine learning. In *ISOC Network and Distributed System Security Symposium – NDSS 2020*, San Diego, CA, USA, February 23-26, 2020. The Internet Society. 2, 8

[38] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014: 23rd USENIX Security Symposium*, pages 797–812, San Diego, CA, USA, August 20–22, 2014. USENIX Association. 1

[39] Theo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis R. Bach. Ariann: Low-interaction privacy-preserving deep learning via function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 2022:291 – 316, 2022. 1, 13

[40] Peter Scholl, Nigel P. Smart, and Tim Wood. When it's all just too much: Outsourcing MPC-preprocessing. In Máire O'Neill, editor, *16th IMA International Conference on Cryptography and Coding*, volume 10655 of *Lecture Notes in Computer Science*, pages 77–99, Oxford, UK, December 12–14, 2017. Springer, Heidelberg, Germany. 2, 4

[41] Nigel P. Smart and Titouan Tanguy. TaaS: Commodity MPC via triples-as-a-service. Cryptology ePrint Archive, Report 2019/957, 2019. https://eprint.iacr.org/2019/957. 2, 4

[42] Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. Privacy-preserving decision trees evaluation via linear functions. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017: 22nd European Symposium on Research in Computer Security, Part II*, volume 10493 of *Lecture Notes in Computer Science*, pages 494–512, Oslo, Norway, September 11–15, 2017. Springer, Heidelberg, Germany. 1

[43] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, and Lei Yu. Privacy-preserving inductive learning with decision trees. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 57–64, 2017. 1

[44] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26–49, July 2019. 1, 10

[45] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. FALCON: honest-majority maliciously secure framework for private deep learning. *CoRR*, abs/2004.02229, 2020. 1, 2, 8, 9, 10, 11

[46] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press. 1

[47] Qizhi Zhang, Lichun Li, Shan Yin, and Juanjuan Sun. Mpc protocol for g-module and its application in secure compare and relu, 2021. 2, 4

Appendix A
Security Proof for General Function

We now provide the security proof for a general function f with the form $f(x) = h(g_1(x_1), g_2(x_2), \ldots, g_k(x_k))$.

Theorem 3. *Our protocol securely realizes the ideal functionality $\mathcal{F}_f$ in the presence of a PPT adversary $\mathcal{A}$ who can corrupt at most one party as semi-honest.*

Proof: To prove this, it suffices to prove Lemma 4 and Lemma 3. ■

Lemma 3. *Our protocol is secure in the presence of a PPT adversary $\mathcal{A}$ who corrupts P_3 as semi-honest.*

Proof: Since P_3 is only involved in the input-independent offline phase, adversary $\mathcal{A}$ gains no information. ■

Lemma 4. *There exists a PPT Simulator $\mathcal{S}$ that can simulate the adversary $\mathcal{A}$'s view in our protocol when $\mathcal{A}$ corrupts either P_1 or P_2 such that the simulated view is indistinguishable from the view of the real execution.*

Without loss of generality, let $\mathcal{A}$ be a probabilistic polynomial time (PPT) adversary who corrupts P_1 (since P_1 and P_2 are symmetric). We construct a PPT simulator $\mathcal{S}$ that simulates the adversary $\mathcal{A}$'s view.

To simulate the offline phase, the simulator $\mathcal{S}$ generates uniformly random values as entries for P_1's share of the various function tables as well as for P_1's share of r and sends them to $\mathcal{A}$. Note that $\mathcal{S}$ records P_1's share of r as r'.

To simulate the online phase, $\mathcal{S}$ samples a uniformly random values as $[y]_2$ and sends it to $\mathcal{A}$ to simulate opening $[y = x + r]$. At the same time, $\mathcal{S}$ receives $[y]_1$. $\mathcal{S}$ can recover the input $[x]_1$ of P_1 by computing $[x]_1 = [y]_1 - r'$ where r' is recorded from the precomputation step. Next, $\mathcal{S}$ samples k uniformly random elements as $[\widetilde{a}_1]_2, [\widetilde{a}_2]_2, \ldots [\widetilde{a}_k]_2$ and records and sends them to $\mathcal{A}$ to simulate opening $[\widetilde{a}_i]$. At the same time, $\mathcal{S}$ receives $[\widetilde{a}_1]_1, [\widetilde{a}_2]_1, \ldots [\widetilde{a}_k]_1$.

Using $[\widetilde{a}_1]_1, [\widetilde{a}_2]_1, \ldots, [\widetilde{a}_k]_1$ and $[\widetilde{a}_1]_2, [\widetilde{a}_2]_2, \ldots [\widetilde{a}_k]_2$, $\mathcal{S}$ computes $\widetilde{a}_1 = [\widetilde{a}_1]_1 + [\widetilde{a}_1]_2, \widetilde{a}_2 = [\widetilde{a}_2]_1 + [\widetilde{a}_2]_2, \ldots \widetilde{a}_k = [\widetilde{a}_k]_1 + [\widetilde{a}_k]_2$. $\mathcal{S}$ now looks at the function table for $h(\cdot)$ it sent to $\mathcal{A}$ during the offline phase, and locates the output of $\widetilde{a}_1, \widetilde{a}_2, \ldots \widetilde{a}_k$ in the function table, labels it $[s]_1$.

To simulate opening of $[s]$, $\mathcal{S}$ first sends the input $[x]_1$ of P_1 to the ideal functionality, and receive the output s^*. $\mathcal{S}$ can then calculate $[s]_2$ such that $[s]_2 = s^* - [s]_1$, and send it to $\mathcal{A}$ to simulate opening s.

We now show that this simulation is indistinguishable from the real protocol execution.

Since the precomputation that P_1 receives in the real execution are all secret shares of values, they are all independent and uniformly random, which has identical distribution as what $\mathcal{S}$ generates for precomputation. Since r is generated uniformly at random in the real execution, $x + r$ is also uniformly at random, the simulated view has identical distribution to the real execution. Since r_is are generated uniformly at random in the real execution, each $\widetilde{a}_i$ is also uniformly at random, the simulated view has identical distribution to the real execution. We highlight the nontriviality that only one value from each $g(\cdot)$ is opened, ensuring that each r_i is effectively only used once, making all $\widetilde{a}_i$ independent and uniformly random, which is indistinguishable between the real execution and the simulated view.

Since the final output $[s]$ is a secret share, it is indistinguishable from a value sampled from the uniform distribution. Note that the $\mathcal{S}$ is able to simulate the correct output with random shares, which means the joint distribution of the output and the view is identical for the real execution and the simulation.