Communication Efficient Secret Sharing with Dynamic Communication-Computation Conversion

Zhenghang Ren*, Xiaodian Cheng*, Mingxuan Fan*, Junxue Zhang*†, Cheng Hong‡
* iSing Lab,Hong Kong University of Science and Technology, † Clustar, ‡ Alibaba Group
Email: {zrenak, xchengaq, mfanae, jzhangcs}@cse.ust.hk, vince.hc@alibaba-inc.com

***Abstract*—Secret Sharing (SS) is widely adopted in secure Multi-Party Computation (MPC) with its simplicity and computational efficiency. However, SS-based MPC protocol introduces significant communication overhead due to interactive operations on secret sharings over the network. For instance, training a neural network model with SS-based MPC may incur tens of thousands of communication rounds among parties, making it extremely hard for real-world deployment.**

To reduce the communication overhead of SS, prior works *statically* convert interactive operations to equivalent non-interactive operations with extra computation cost. However, we show that such static conversion misses chances for optimization, and further present SOLAR, an SS-based MPC framework that aims to reduce the communication overhead through *dynamic* communication-computation conversion. At its heart, SOLAR converts interactive operations that involve communication among parties to *equivalent* non-interactive operations within each party with extra computations and introduces a speculative strategy to perform opportunistic conversion when CPU is idle for network transmission. We have implemented and evaluated SOLAR on several popular MPC applications, and achieved 1.6-8.1 times speedup in multi-thread setting compared to the basic SS and 1.2-8.6 times speedup over static conversion.

I. Introduction

Secret Sharing (SS) is one of the most famous primitives to construct MPC [16], [19] programs because of its simplicity and computational efficiency [6]. SS protects privacy by randomly splitting the secret data to multiple sharings and distributing to the parties. The secret data is reconstructed only when the parties reach an agreement. During evaluation of the function, the parties operates on secret sharings with the protocol and get the result which, after reconstruction, equals to the result produced by the same function evaluated with plaintext directly.

SS incurs heavy communication overhead because commonly used operations such as multiplications involve communication between parties. For example, the "millionaire function" that takes two secret-shared numbers as input and outputs a secret shared boolean value, would require more than five rounds of interaction [29], which may cost hundreds of milliseconds on the Internet.

It has been explored that the communication round complexity can be reduced by converting multiple consecutive operations to a single "flattened" operation that requires less communication rounds. Recent works such as ABY2.0 [25] and multi-fan-in gates [24] have leveraged this opportunity and designed specified protocols. For example, ABY2.0 designed a MUL3 protocol that produces the multiplication of three secret sharings with only one round of interaction, while the original multiplication protocol requires two rounds.

However, it also brings extra computation overhead to convert operations for saving communication. To limit computation overhead, current frameworks with conversion adopts *static* conversion strategy, which converts every three or four operations into a flattened operation. But static conversion may miss the chance for optimization as we have identified certain opportunities to convert at most eight operations without too much computation overhead in MPC programs.

To exploit more chances to save communication, we build SOLAR, a SS based MPC framework that performs *dynamic* communication-computation conversion on operations of secret sharing. With dynamic conversion, SOLAR maximizes the speedup over basic SS protocols without manually setting parameters. Moreover, extra computation is limited by interrupting conversion process when the conversion overhead exceeds its benefit.

Our basic idea to achieve dynamic conversion is to model the computation and communication cost in runtime and increase the conversion size until the time spent in conversion has surpassed a threshold or timeslot, which indicates that the computation overhead is too large. When time spent in conversion surpasses timeslot, SOLAR interrupts conversion to avoid increasing total cost. However, two major challenges need to be addressed to achieve maximum speedup through the dynamic conversion.

1) Adjusting timeslot is difficult in runtime because the speedup relates to both reduced communication and increased computation, which are difficult to model accurately.
2) Interrupting conversion wastes the computation already spent and the operation needs re-run which introduces extra cost.

The key observation to address the above challenges is that the original operations could serve as indicators of timeslot, because the time spent in interaction is an accurate estimation of saved communication cost. Moreover, original operations can also serve as the efficient fallback for interrupted conversions. Specifically, during the evaluation of an

This paper is supported in part by Hong Kong RGC TRS T41-603/20-R, and the Turing AI Computing Cloud (TACC) [33]

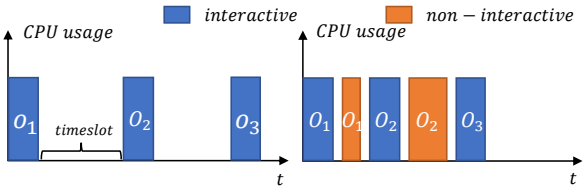


Fig. 1. An example of converting interactive operations into non-interactive operations and conducted while CPU is idle. Assume O_1, O_2, O_3 are interactive operations in basic SS. The **timeslot** presents a chance for evaluating O_i in non-interactive approach. The evaluation of next operation could start immediately after prior operation finished non-interactively within timeslot.

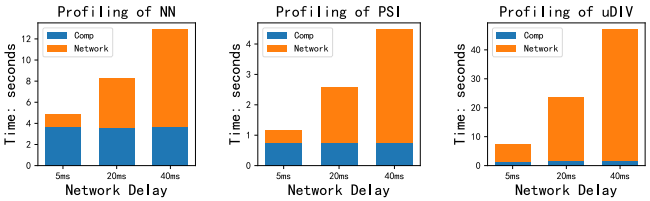


Fig. 2. Profiling of NN, PSI, uDIV under different network conditions. We simulate the propagation delay over parties. The communication cost contains both transmission delay and propagation delay.

interactive operation, there is a timeslot when the parties are communicating with each other, as shown in Figure 1. If the operation is converted successfully and evaluated as a flattened operation within the timeslot, the communication is saved and can be interrupted. Otherwise, the conversion is interrupted and the original operation will act as a fallback. With these measures, 1) SOLAR measures the timeslot for saved communication cost accurately with original operations. 2) Interrupted conversion of operation does not need re-run because the result of original operation is equivalent and acts as an efficient fallback for interrupted conversion.

However, it has higher computation cost to perform both original and converted operations. SOLAR relieves this problem by merging the common part of computation in original and converted operations. As shown in Figure 1, the figure on the right side shows optimized pattern. The orange blocks represent converting original operations, and they reuse the intermediate result of original operations marked by blue. Conversions are conducted in the timeslot when the original operations are transmitting data and do not consume CPU time. Note that converting cost is growing and we will illustrate this in Section II-B.

Moreover, in order to exploit more chances to reduce communication, SOLAR uses speculative conversion strategy that always performs conversion in the timeslot to maximize speedup because accurate estimation of timeslot and low fallback price enable us to exploit all chances for optimization without risk of increasing total cost. If the conversion finishes within the timeslot, i.e., before the original operation finishes interacting with other parties, the interaction in the original operation is interrupted and some communication cost is saved. Otherwise, the conversion will be interrupted and the original operation delivers its result as fallback for the operation.

We implemented SOLAR on top of ABY, a popular SS based MPC framework. In essence, SOLAR is built as an optimizer that watches the finish of two subroutines, namely original operation and the process of converting and evaluate the operation non-interactively. SOLAR accepts the first result produced by the two subroutines and interrupts the other one. We tested SOLAR on basic operators and programs under both semi-honest security and malicious security with authentication. The results show that SOLAR achieves 1.6-8.1× speedup over basic SS on both security settings on tested applications. It also achieves 1.2-8.6× speedup over static conversion.

II. Background

MPC [16], [19] has been studied for decades, and the application of SS based MPC has drawn great attention recently mainly because of arising concern of privacy in the era of big data. SS protocols have been applied in privacy-preserving applications such as Private Set Intersection [10], [27], [28], Biometric Matching [17], [18], Data Mining [21], and Federated Learning [31], [32].

However, SS based MPC still faces large communication overhead because common operations such as multiplication and comparison on secret sharings require at least one round of communication between parties. These interactive operations are very common in MPC programs. As a result, most privacy-preserving applications based on SS protocols are several orders of magnitude slower than non-MPC applications and communication cost dominates the total cost. We tested Private Set Intersection (PSI), Neural Network (NN) inference on MNIST dataset and unsigned division (uDIV) on two parties over the network with 1000 Mbps bandwidth and different network propagation delay (5ms, 20ms, 40ms). The result shows the dominance of communication cost, especially in high-latency environment, where over 70% time spent in communication, as shown in Figure 2.

To illustrate the root cause for communication overhead, in this section, we introduce formal definition and operations in SS, including how SS protects private data, how to perform basic operations such as multiplication on secret sharings, and current strategies to optimize the performance of SS protocol. To simplify narration, we assume there are two parties.

A. Secret Sharing

The general workflow of SS with multiple parties and a joint function is 1) The parties generate secret sharings of their private data and send to other parties so that each party has parts of the secret sharings of private data. 2) The parties describe the joint function as a circuit and evaluate collaboratively, in which the inputs are secret sharings of private data and the outputs are secret sharings of the function output. 3) The parties reconstruct the output.

In the first stage, the parties generate secret sharings to protect the value of private data. For example, in two-party

arithmetic secret sharing, secret value x is split randomly into two sharings $x = x_0 + x_1 \mod 2^k$ by the owner of x. The owner sends one of the secret sharing to the other party, who cannot know the real value of x with only one sharing. In general n-party setting, similarly, every party with private data also generates secret sharings according to the number of parties and send to other parties.

In the second stage, the function to be evaluated jointly is first expressed as a circuit which is made up of gates, which are the basic operations that can be evaluated on secret sharings. For example, if the function is to calculate the product of two vectors with length l, in arithmetic sharing, the circuit will be l MUL gates and the output of l MUL gates will be passed into the input of $l-1$ ADD gates.

Some operations on secret sharings can be done locally without interaction, while some operations require interaction of parties to ensure correctness. For example, in two-party arithmetic secret sharing, to multiply two secret sharings x_i and y_i and ensure the reconstructed result is equal to $x \cdot y$, noticed that $x \cdot y = x_0y_0 + x_0y_1 + x_1y_0 + x_1y_1$ involves the product of sharings in different parties (x_0y_1 and x_1y_0), at least one round of communication is needed to securely calculate $x \cdot y$. Furthermore, it takes only microseconds to compute multiplication locally, while communication takes at least milliseconds over the Internet, so communication becomes the major cost in this operation.

In the last stage, the parties reconstruct the output of circuit as the result of function. This is symmetric to the sharing process in the first stage and is not the major cost of SS protocols. It is the second stage that takes the major part in the total cost of SS protocols.

Interactive Operation in SS Operations that cannot be evaluated locally and requires interaction are called *interactive operations* and they takes most of the total cost, in contrast of the non-interactive operations which can be performed locally. To show how to evaluate interactive operation securely, consider the following operation: secret value x and y is owned by party P_0 and party P_1 respectively. The function is to multiply two secret values. In arithmetic secret sharing, $x_0 + x_1 = x$ and $y_0 + y_1 = y$ and P_i owns $x_i, y_i, i \in \{0, 1\}$. Denote $[[\cdot]]$ as secret sharings, then x_0, x_1 can be noted as $[[x]]$ and y_0, y_1 can be note as $[[y]]$.

The evaluation of interactive operation is adopted from Beaver's circuit randomization [5] and the protocol is as follow:

- The parties generate random secret r_x, r_y and keep $[[r_x]], [[r_y]]$ locally. This can be achieved by the parties generating random numbers locally without telling the other party.
- The parties reconstruct $x - r_x$ and $y - r_y$. And P_i calculates $(x-r_x)(y-r_y)[[1]]+(x-r_x)[[r_y]]+[[r_x]](y-r_y) + [[r_xr_y]]$.

The correctness can be proved by reconstructing the result which equals to $x \cdot y$. Note that $[[r_xr_y]]$ requires extra computation but it can be done before the task since r_x, r_y are irrelevant to the real input.

Since it requires reconstruction to perform interactive operations, the performance of interactive operation is several orders of magnitude slower than non-interactive operations. Note that other SS schemes such as boolean sharing which splits secrets with boolean XOR operation have similar operations, in which AND requires reconstruction of secret sharings.

B. SS Communication Optimizations

To perform reconstruction in interactive operations, at least one round of communication is needed to send secret sharing to other parties. However, the communication round can be reduced when evaluating multiple interactive operations.

Interactive Operation Conversion. Consecutive interactive operations can be converted and evaluated with less rounds of communication. For example, if there are two consecutive interactive operations $O_1 = x \cdot y$ and $O_2 = O_1 \cdot z$ in which x, y, z are secret data and protected by SS. In basic protocol, O_1 requires one round of communication and O_2 requires one more round. However, the communication can be reduced with the following expression:

$$\begin{aligned} O_2 &= O_1 \cdot z \\ &= x \cdot y \cdot z \\ &= (x - r_x + r_x)(y - r_y + r_y)(z - r_z + r_z) \\ &= (x - r_x)(y - r_y)(z - r_z) + (x - r_x)(y - r_y)r_z \\ &+ (x - r_x)r_y(z - r_z) + r_x(y - r_y)(z - r_z) \\ &+ (x - r_x)r_yr_z + r_x(y - r_y)r_z + r_xr_y(z - r_z) + r_xr_yr_z \end{aligned} \tag{1}$$

In Eq. 1, r_x, r_y, r_z is the random number generated collaboratively by the parties. Similar to Protocol II-A, the parties can first reconstruct $(x - r_x), (y - r_y), (z - r_z)$. Denote $[[1]]$ as the secret sharing of 1, $[[O_2]]$ can be set as:

$$\begin{aligned} [[O_2]] &= (x - r_x)(y - r_y)(z - r_z)[[1]] + (x - r_x)(y - r_y)[[r_z]] \\ &+ (x - r_x)(z - r_z)[[r_y]] + [[r_x]](y - r_y)(z - r_z) \\ &+ (x - r_x)[[r_yr_z]] + (y - r_y)[[r_xr_z]] \\ &+ [[r_xr_y]](z - r_z) + [[r_xr_yr_z]] \end{aligned} \tag{2}$$

Again, $[[r_xr_yr_z]], [[r_xr_y]], [[r_xr_z]], [[r_yr_z]]$ can be computed before the task. So in online execution, $[[O_2]]$ can be calculated with only one round of communication, which is to reconstruct $(x-r_x), (y-r_y), (z-r_z)$. We can apply similar conversion on four or even more operations and save more communication. However, it will also introduce extra computation overhead which is exponential to the number of involved secret sharings. As shown in Eq. 2, it involves $2^3 = 8$ multiplications to evaluate two consecutive MUL on secret sharings.

To solve this, current works set fixed number of involved secret sharings to limit computation overhead. But with fixed conversion, we might lose opportunity to optimize when the computation facility is powerful or the communication cost is very large due to high latency network.

Considered that the non-interactive approach usually outperforms the interactive approach and both approach can be taken independently, we are motivated to speculatively convert

every interactive operation during execution and interrupt the conversion when the operation has been finished interactively. Furthermore, the conversion could be scheduled to the timeslot when the CPU is idle for network transmission during interaction, which reduces idle cycles of CPU and avoid interfering the interactive executor.

However, the main difficulty is how to deal with interrupted conversion without disposing generated random secrets. Specifically, the random secrets such as $[[r_x]], [[r_y]], [[r_z]]$ and their multiplications shown in Eq. 2 have to be disposed if O_2 is interrupted because it is slower than original operations. Although it is feasible to prepare as much random data as possible, reloading random secrets is still expensive especially when the size of random secret is large.

To solve this problem, we design new expression of secret sharings which can use the same set of random secrets in both interactive and non-interactive approach. This makes converted operations compatible with original operations and can be interrupted without extra cost. We elaborate the detail in the next section.

III. Design and Implementation

A. Overview of SOLAR

At the core of SOLAR, it exploits original operations as a guidance for adjusting the size of conversion dynamically. Unlike previous works that fix the conversion size to a small value [24], [25] and perform conversion statically, SOLAR performs conversion for the next operation dynamically and checks if the conversion finish within a timeslot which is estimated by the time spent in interacting with the other party in original operations. If the conversion and non-interactive evaluation finishes within the timeslot, it means SOLAR saves more communication time than extra computation and reduces total cost successfully. Otherwise the conversion is interrupted and the original operation commits its result which is equivalent to the interrupted operation.

The foundation of SOLAR is the conversion of interactive operations to non-interactive operations, which is an extension of Beaver's circuit randomization [5]. In Beaver's basic implementation, the parties calculates the circuit with random input before the real execution and correct the result in real execution, as shown in Section II-A. And it performs correction that involves reconstruction of the offset between real input and random input in every interactive operations. In the extended implementation, multiple interactive operations reconstructs all offset in single round of reconstruction, but is also leads to higher computation cost during correction.

SOLAR performs converted operations non-interactively and check if the last converted operation finishes earlier than the corresponding original operation, which is evaluated interactively. However, performing both converted and original operations leads to higher total cost, because both original and converted operations involves reconstruction on the same set of secret sharings and calculating with the offset. Also, interrupted conversion will lead to higher computation cost for the next converted operation because interrupted conversion failed to supply the result needed for the next operation.

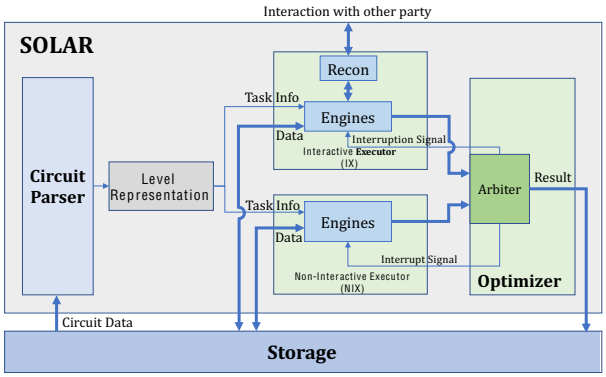


Fig. 3. Structure of SOLAR

We solve this problem by merging the common part of implementation between original and converted operations. By expressing secret sharings in vectorized forms that contain data for both original and converted operations, we successfully merges the computation part of original operations into converted operations. By doing this, SOLAR gets the following benefits:

- There is no risk of increasing total cost since the common part of evaluation is merged and the different part is scheduled strictly within the timeslot of network transmission.
- The efficiency of fallback for converted operations increases because the previous original operations provides its result that can be reused to reduce computation.

Without worrying about computation overhead, SOLAR can achieve best performance with speculative conversion strategy. To achieve this, SOLAR is set up as shown in Figure 3. From the left to the right, it can be roughly divided into four parts: Circuit Parser, two Circuit Executors including Interactive Executor (IX) and Non-Interactive Executor (NIX), Optimizer, and Storage. The general workflow of SOLAR is as follow:

1) The MPC task described with a circuit and the gates are sorted in topology order by Circuit Parser. The gates with the same topology order are grouped as a *Level Representation* and fed into Circuit Executor. Each gate is expressed as an operation with input and operation type. This step is common practice of many circuit based MPC so we will not dive into detail.
2) The subexecutor IX performs the original operations interactively. During sending data, the NIX performs conversion on the operation and evaluate non-interactively.
3) Optimizer accepts the result from the subexecutors. It interrupts the other subexecutor when one subexecutor finishes first. The result will be written into storage which represents a map from gate to its inputs and outputs.

Next we describe the detail of vectorized secret sharing and how it enables efficient evaluation of two operations and fallback.

B. Vectorized Secret Sharing

Vectorized secret sharing is an extended expression of secret sharing that maintains all prior secret sharings and can be evaluated in both IX and NIX. To simplify narration, we first define some symbols used in the following contents, as shown in Table I.

TABLE I
NOTATIONS AND OPERATORS USED THROUGHOUT THIS SECTION. NOTE THAT r_x AND r_y IS THE CORRESPONDING RANDOM INPUT GENERATED WITH BEAVER CIRCUIT RANDOMIZATION.

Notation	Definition
$[[a]]$	secret sharing of secret value a
$ovec(x)$	$(1, x - r_x)$
$ovec(x + y)$	$(1, x - r_x, 1, y - r_y)$
$ovec(x \times y)$	$(1, x - r_x, y - r_y, (x - r_x)(y - r_y))$
$rvec(r_x)$	$([[r_x]], [[1]])$
$rvec(r_x + r_y)$	$([[r_x]], [[1]], [[r_y]], [[1]])$
$rvec(r_x \times r_y)$	$([[r_x r_y]], [[r_y]], [[r_x]], [[1]])$

Then we express both original operations and converted operations using the same notation.

Original Operations For original operations such as MUL with two random inputs a, b and real inputs x, y in arithmetic secret sharing, the parties first reconstruct the offset $x - a$ and $y - b$ with one round of communication. Then the online calculation proceeds as:

$$\begin{aligned}[[xy]] &= [[r_x r_y]] + [[r_y]](x - r_x) + [[r_x]](y - r_y) \\ &\quad + (x - r_x)(y - r_y) \\ &= (1, x - r_x, y - r_y, (x - r_x)(y - r_y)) \cdot \\ &([[r_x r_y]], [[r_y]], [[r_x]], [[1]])\end{aligned}$$

The expression above can be regarded as an inner product of $ovec(x \times y)$ and $rvec(r_x \times r_y)$ as defined in Table I.

Moreover, the evaluation for operations such as ADD in arithmetic sharing that does not need interaction can also be expressed with $rvec$ and $ovec$. Specifically, for ADD operation:

$$\begin{aligned}[[x + y]] &= [[r_x]] + [[x - r_x]] + [[r_y]] + [[y - r_y]] \\ &= (1, x - r_x, 1, y - r_y) \cdot ([[r_x]], [[1]], [[r_y]], [[1]])\end{aligned}$$

As shown above, the evaluation of all original operations can be expressed with the product of corresponding $ovec$ and $rvec$.

Converted Operations For converted operations such as MUL3, it can also be expressed with vectorized form. As shown in Expression 2, $[[O_2]]$ can be expressed as inner product of $ovec(x \times y \times z)$ and $rvec(r_x \times r_y \times r_z)$.

Similarly, when encountered operations that does not require interaction such as ADD in arithmetic sharing, the calculation of next $ovec$ proceed with $+$ operator as in Table I. Specifically, consider the following MPC task as an example:

In secure machine learning task, two parties train a linear model collaboratively with data x, *inverted* label y and parameter w all secret shared and unknown to both parties. Then the gradient will be:

$$(wx + y)x$$

With converted operation and vectorized secret sharing, the calculation of gradient can be conducted as follow: 1) Calculate $ovec(w \times x)$ and $rvec(r_w \times r_b)$. 2) Calculate $ovec(w \times x + y)$ and $rvec(r_w \times r_b + r_y)$. 3) Calculate $ovec((w \times x + y) \times x)$ and $rvec((r_w \times r_b + r_y) \times r_x)$. 4) Perform inner product of the last $rvec$ and $ovec$.

Each step in the above is defined in Table I and they do not involve communication with the other party. Note that the calculation of $rvec$ can be scheduled before the task because they only contains correlated random numbers and are irrelevant to the real values.

To summarize, vectorized secret sharing is capable of expressing both original and converted operations. The communication cost lies in getting $ovec$ by reconstructing the offset as plaintext.

C. Circuit Executor

Circuit Executor consists of two subexecutors, IX and NIX, to perform original interactive operations and converted non-interactive operations respectively. With vectorized secret sharing, it merges the calculation of original operations into converted operations so that the total computation cost is reduced.

To elaborate the detail, we divide the evaluation into three phases according to the pattern of original operation: pre-communication, communication, and post-communication.

Pre-communication IX prepare new $ovec$ for reconstruction. It fetch the secret sharing from previous operation and the random secrets of the input. For example, in a MUL operation, IX fetches input secret sharing and the random secrets that generated before the task.

Communication IX reconstructs the $ovec$ which involves communication with the other party. During the transmission, NIX evaluates the same operation non-interactively with the $ovec$ and $rvec$. Unlike IX, NIX does not need reconstruction to get $ovec$, instead, it fetches $ovec$ either from previous converted operation or original operation.

Post-communication If NIX finishes the converted operation before the IX finishes transmission, the IX is interrupted and NIX commits its result. Otherwise NIX is interrupted and IX commits its result. This is implemented by a optimizer watching the first subexecutor to finish the operation and interrupting the other subexecutor. The result is written to Storage as a $ovec$ for the next operation.

D. Security Definition

SOLAR can support both semi-honest and malicious security model. For semi-honest model, it's assumed that the parties will not deviate from the protocol but will try to extract information from the messages. For malicious security, the malicious party will deviate from the protocol and attempt to

cheat. For example, the malicious party may send fake secret sharing values.

To detect the malicious behavior during execution, current secret sharing protocols such as SPDZ2k [14] adopt Message Authentication Checking (MAC) mechanism which detects the malicious behavior by checking the MAC values with expected values and aborting if they are not equal.

SOLAR supports both semi-honest and malicious security because the essence of authentication is the calculation of MAC, which is also a secret sharing. So the generation of MAC values can also be implemented by Beaver Circuit Randomization.

E. Implementation

We build the Circuit Executor on top of ABY, a mixed protocol framework to conduct two-party secure computation. All parties in SOLAR are the same and act as both client and server. The parties reads the circuit description file formulated in Bristol Circuit Format [1] and describe the circuit with a DAG, in which every gate is associated with two incoming edges indicating input signals and one outgoing edge indicating output signal.

Before the MPC task, the parties prepare $rvec$ collaboratively. Since $rvec$ is irrelevant to the input of MPC task, the parties are able to prepare enough $rvec$ for upcoming task. Our implementation in this part is based on correlated oblivious transfer that generates correlated random numbers without revealing to the other party.

During the MPC task, the parties load $rvec$ and reconstruct $ovec$ with the input of MPC task. Then each gate in the circuit is evaluated as described in Section III-C.

Moreover, observed that most evaluation process is the calculation of vector, we implement the Circuit Executor in parallel. The total work is about 900 lines of code in C++.

IV. Evaluation

In this section, we give a detailed evaluation of SOLAR for commonly used MPC operations and applications. We first provide the evaluation settings, including experiment environment and test cases, then introduce the evaluation metrics and finally show the result of our experiments.

A. Setup

In our experiment, we set up two parties, and each party has multiple computing nodes. Each node is equipped with Intel(R) Xeon(R) Gold 5115 CPU with 20 cores and 128GB memory. The number of threads in each computing node is limited to 16. We tested one, two, four nodes in each party, namely 16, 32, 64 threads. Single threading is also tested to show the effectiveness of the optimizer when computing performance is low.

We simulate different propagation delay over network between parties, including 5ms, 20ms and 40ms, to show the effectiveness of SOLAR in different network delays. The network bandwidth between two clusters is set as 1000 Mbps, and the network bandwidth within each cluster is set as 40 Gbps.

TABLE II
THE DEPTH AND WIRES OF TEST CASES EVALUATING SOLAR

Operator	EQ	LT	uDIV	SHA256	PSI	NN
Depth	7	22	2205	1607	22528	382
Inputs	128	64	128	768	4096	784
Outputs	1	1	64	256	4096	10

In all experiments, the input data from both parties is protected using secret sharing, and only the result is revealed in the end. We describe the circuit file using the Bristol circuit fashion [1], in which the circuit is represented as a DAG and each vertex is a gate with two input wires and one output wire. Then they prepare random input as specified in Beaver's random circuit specified in Section II-A. In the online phase, the parties first generate secret sharings of their input data and evaluate each gate in topology order with the random data generated in the first phase.

SOLAR is tested on roughly two kinds of MPC programs.

- Small basic operations: 64-bit equality testing(EQ), 64-bit comparator(LT), and 64-bit unsigned division(uDIV).
- Large applications: SHA256, Circuit based PSI and Privacy-preserving inference of neural network models (NN).

Basic Operations. 64-bit EQ is performed by two boolean-shared values conducting XORs and reductive ANDs. LT operator is conducted on two 32-bit unsigned values. 64-bit uDIV circuit has two inputs and one output whose bit lengths are 64. The depths of these circuits are listed in Table II.

SHA256 is widely used in data integrity checking and SHA256 in MPC enables collaborative data integrity checking without leaking private data. Previous works have applied SHA256 in MPC to check the integrity of private data [4], [13]. But SHA256 is a deep boolean circuit involving many communication rounds in previous MPC frameworks.

PSI defines the task to calculate the intersection of two private sets that belongs to two parties respectively. PSI can be applied in privacy-preserving SQL query, in which a JOIN operation is transformed into PSI between two columns in database. [2], [7], [20]. Circuit based PSI is a communication-intensive application as it adopts equality checking as one of the basic operations. In our experiment, we first securely sort the parties' data and then perform equality testing on adjacent data.

Privacy Preserving Machine Learning (PPML) Previous optimization methods on PPML mostly focus on improving the computational throughput or communication data size [3], [11], [23], [26] because machine learning training usually process large volumes of training data. However, reducing the communication rounds is also important because it involve multiple interactive operations, leading to many communication rounds. In our experiment, we tested inference on NN with MNIST dataset. The NN architecture is shown in Figure 4.

Some operations like sigmoid activation are not directly supported in secret sharing, so we adopt the idea of Se-

TABLE III
TIME(MS) AND SPEEDUP OVER NAIVE SS ON BASIC OPERATIONS UNDER DIFFERENT NUMBER OF THREADS AND DIFFERENT NETWORK LATENCY. **L.** STANDS FOR NETWORK LATENCY BETWEEN PARTIES. **F.** STANDS FOR DIFFERENT FRAMEWORKS.

Eval	Threads	1			16			32			64		
	F. / L.	SS	SOLAR	Speedup	SS	SOLAR	Speedup	SS	SOLAR	Speedup	SS	SOLAR	Speedup
EQ	5ms	42.4	10.8	3.89	41.3	10.3	4.01	41.1	10.2	4.03	41.2	9.5	4.3
	20ms	163.1	40.4	4.03	159.7	40.2	3.97	160.4	39.2	4.09	160.3	34.2	4.68
	40ms	324.4	80.5	4.03	321.4	80.2	4.01	318.5	78.4	4.05	318.9	64.3	4.95
LT	5ms	125.1	43.1	5.40	122.3	21.2	5.76	121.7	19.3	6.31	122.1	15.4	7.89
	20ms	488.8	83.0	5.89	483.9	78.5	6.16	482.2	79.3	6.07	483.1	61.3	7.87
	40ms	968.6	163.1	5.93	964.6	155.2	6.21	963.9	158.3	6.08	962.8	119.2	8.07
uDIV	5ms	11663	2178	5.35	11548	2098	5.50	11472	2019	5.68	11335	2094	5.41
	20ms	42940	7689	5.58	42814	7593	5.63	42644	7501	5.68	88543	14225	6.22
	40ms	89096	15910	5.60	88727	15494	5.72	88543	14225	6.22	88122	14023	6.28
SHA256	5ms	9617	3875	2.48	9433	3638	2.59	9225	3442	2.68	9133	3305	2.76
	20ms	33880	12047	2.81	33593	11857	2.83	33054	11525	2.87	32584	11047	2.95
	40ms	66074	22827	2.89	65732	22591	2.91	65034	22487	2.89	64497	22028	2.92
PSI	5ms	1045	757	1.38	1013	612	1.66	992	468	2.11	983	414	2.37
	20ms	3505	2079	1.69	3464	1553	2.23	3436	1217	2.82	3451	957	3.60
	40ms	6771	3966	1.70	6714	3048	2.20	6683	2076	3.22	6690	1623	4.12
NN	5ms	4813	4264	1.13	2212	1398	1.58	1958	1162	1.69	1859	1116	1.67
	20ms	8309	5911	1.40	5628	2210	2.54	5404	1976	2.73	5279	1746	3.02
	40ms	12923	6775	1.90	10127	2546	3.98	9949	2245	4.43	9816	2007	2.89

1. *Input*: $R^{28\times28}$
2. *Convolution*: window size = (5,5), stride =(1,1), channels = 16, no padding.
3. *ReLU*
4. *Max Pooling*: window size = (2,2), stride = (2,2)
5. *Convolution*: window size = (5,5), stride = (1,1), channels = 16, no padding.
6. *ReLU*
7. *Max Pooling*: window size = (2,2), stride= (2,2)
8. *Fully Connected*: Output = (100, 1)
9. *ReLU*
10. *Fully Connected*: Output = (10, 1)

Fig. 4. Neural Network architecture used in evaluation.

cureML [23] and simulate with other operations. ReLU layer is implemented by a boolean circuit which produces $max(x, 0)$. Max Pooling layer is implemented by applying comparing circuit on every four elements in the tensor and produces the maximum. These layers involve non-linear gates such as AND so they require interaction of parties.

The input size, output size and depth of the above applications are shown in Table II. Note that this may vary due to different implementations, but it does not affect the conclusion of our experiment since we use the same implementation throughout the evaluation.

In each test case, we compare SOLAR with basic SS to show the performance of SOLAR. Note that we only measure online phase time, which is widely accepted in MPC frameworks [15], [22], [25] because the offline phase is irrelevant to the input of the MPC program and can be conducted in advance.

B. Evaluation Metrics

Four primary metrics are used to evaluate the performance of SOLAR: Speedup over basic SS on semi-honest and malicious security, speedup over fixed conversion, communication cost.

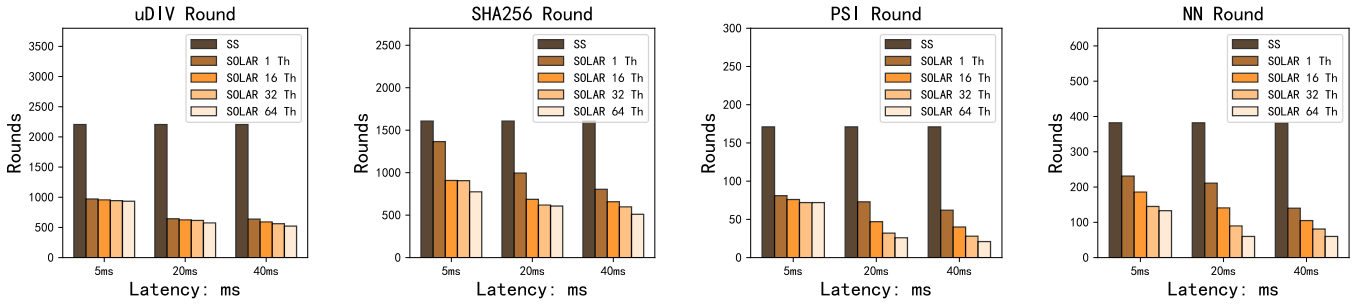
Speedup is measured by simply running the same application with SOLAR and basic SS framework which is implemented in ABY, respectively. We compare the online time in different MPC frameworks and calculate the speedup. We tested on both semi-honest security and malicious security with authentication.

Communication Cost measures the time spent in communication in each test case including rounds and time. In basic SS, the communication rounds stays the same with any number of threads. With SOLAR, the communication may be interrupted as NIX has finished earlier. Thus, we show the difference in online communication cost between SOLAR and the basic SS.

C. Results

TABLE IV
TIME COST (MS) AND SPEEDUP OVER NAIVE SECRET SHARING UNDER **MALICIOUS SECURITY** SETTING UNDER DIFFERENT NUMBER OF THREADS AND DIFFERENT NETWORK LATENCY. **L.** STANDS FOR NETWORK LATENCY BETWEEN PARTIES. **F.** STANDS FOR DIFFERENT FRAMEWORKS.

Eval	Threads	1			16		
	F. / L.	SS	SOLAR	S.	SS	SOLAR	S.
EQ	5ms	51.3	15.2	3.37	49.3	14.1	3.49
	20ms	186.3	52.4	3.55	182.5	49.7	3.67
	40ms	378.6	103.5	3.65	370.2	98.48	3.75
LT	5ms	134.7	28.4	4.73	131.2	25.3	5.17
	20ms	519.2	89.4	5.80	509.3	86.1	5.91
	40ms	1013.4	174.3	5.81	1002.3	169.3	5.92
uDIV	5ms	12032	2534	4.74	11489	2289	5.01
	20ms	43394	8448	5.13	42645	8344	5.11
	40ms	90034	17423	5.16	89143	17432	5.11
SHA256	5ms	10034	4525	2.21	9869	3877	2.54
	20ms	34940	13015	2.68	34256	12397	2.76
	40ms	67239	25159	2.67	66354	23774	2.79
PSI	5ms	1287	917	1.40	1105	766	1.44
	20ms	4694	2853	1.64	4487	2684	1.67
	40ms	7132	4345	1.64	6955	4038	1.72
NN	5ms	5852	5133	1.14	3043	1977	1.53
	20ms	10335	7437	1.38	5856	2710	2.16
	40ms	14829	8748	1.69	11327	3841	2.94



(a) (b) (c) (d)

Fig. 5. Communication Rounds in uDIV, SHA256, PSI, NN under different latency and threads.

Speedup: We evaluated the basic operations and applications mentioned in Section IV-A and the results are shown in Table III.

In the table, we can observe that all MPC programs benefit from SOLAR under different network latency and different number of threads. For all test cases with the same network latency, the speedup tends to increase with more number of threads. This is because NIX is faster with more threads and is more capable to finish earlier than IX. For all test cases with the same number of threads, the speedup tends to increase with larger network latency, which can be explained by longer network timeslot in IX and it will provide more chances for NIX to finish its task.

The results of tests on malicious security are shown in Table IV. Every secret sharing is associated with a MAC value which also needs to be generated and transmitted when reconstruction. It will double the amount of computation and data transmission. At the end of protocol, the MAC value is checked to detect potential malicious behavior, which leads to an extra round of communication.

TABLE V
TIME COST (MS) AND SPEEDUP OVER **FIXED CONVERSION**, WHICH TRIES TO CONVERT EVERY FOUR INTERACTIVE OPERATIONS. **L.** STANDS FOR NETWORK LATENCY BETWEEN PARTIES. **F.** STANDS FOR DIFFERENT FRAMEWORKS.

Eval	Threads	1			16		
	F. / L.	Fixed	SOLAR	S.	Fixed	SOLAR	S.
EQ	5ms	14.5	10.4	1.4	13.1	11.1	1.18
	20ms	54.3	41.2	1.31	51.3	40.8	1.25
	40ms	118.4	81.1	1.45	115.4	80.6	1.43
LT	5ms	45.7	22.9	1.99	42.7	20.8	2.05
	20ms	173.5	83.0	2.09	168.4	77.4	2.17
	40ms	371.8	167.4	2.22	358.3	151.2	2.37
uDIV	5ms	3348	2091	1.60	3244	2089	1.55
	20ms	15767	7714	2.04	14338	7604	1.89
	40ms	32677	16134	2.02	31767	15503	2.05
SHA256	5ms	8356	3874	2.15	7134	3712	1.92
	20ms	18447	12047	1.53	14558	11426	1.27
	40ms	27485	22827	1.20	26343	22894	1.15
PSI	5ms	6174	765	8.07	5375	623	8.62
	20ms	7163	2061	3.47	8157	1560	5.22
	40ms	8174	3897	2.05	7369	3056	2.41
NN	5ms	6061	4284	1.41	5307	1390	3.82
	20ms	7268	5809	1.25	6181	2178	2.83
	40ms	8773	6885	1.27	7492	2521	2.97

From the table we may conclude that under malicious secure setting, SOLAR can still achieve speedup over basic secret sharing protocol. However, compared to the same circuit under semi-honest security, the speedup drops. The reason for lower speedup under malicious security is the increase of computation overhead. During evaluation, both secret sharing and the MAC value need to be calculated, which doubles the amount of computation task. So the NIX has more workload in the time slot when the IX is blocked.

We notice that SS benefit little from increasing the number of threads in most applications except NN. That's because the computation cost in basic SS only takes a minor part in total cost in EQ, LT, and SHA256. Since NN is massively parallel and computation cost takes larger part as shown in Figure 6.

Moreover, to show the speedup over fixed conversion, we set a fixed conversion size in SOLAR as baseline. The result is shown in Table V. From the table, we see that fixed conversion is usually slower than SOLAR. And for some cases like PSI, it is even slower than basic SS, because fixed conversion lacks consideration of network latency and computation performance so it may miss some optimization opportunity or bring too much computation overhead.

Communication: We recorded the communication rounds of SOLAR under different number of threads and different network latency. The result is shown in Figure IV-C. Note that we only count the round in which the Comm. Executor is not interrupted.

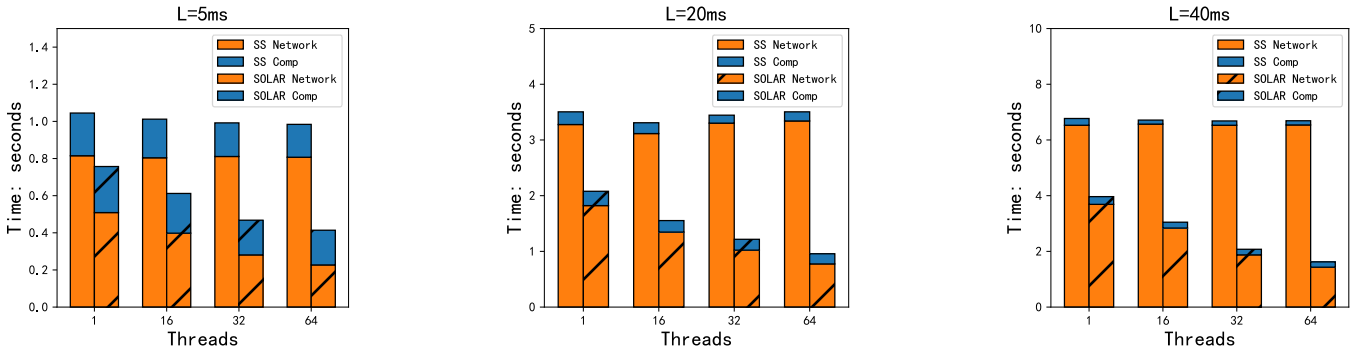
From the figure, we know that SOLAR reduces communication rounds, and as the network latency becomes larger, more communication rounds are saved. Also, more communication rounds are saved as the number of threads becomes larger. The reason is that when network latency becomes larger, NIX will have larger timeslot for conversion. If more threads are used to accelerate computation, NIX will finish in less time. Both conditions enable more communication-computation conversion and reduce more rounds.

Profiling: To give a detailed analysis of the impact of SOLAR, we give profiling on PSI. We show the time spent in communication and computation, as shown in Figure 6.

From the profiling result, it is observed that:

First, based on SS profiling result from Figure 6(a), Figure 6(b), and Figure 6(c), we know that as the network latency increases, the communication overhead also increases. In SS, the increase is proportional to the network latency. Also, as the number of threads increases, only computation time is reduced, but communication time stays barley the same because SS has a fixed number of communication rounds.

Second, SOLAR reduced the time spent in communication



(a) (b) (c)

Fig. 6. Profiling of SS and SOLAR on PSI under different network latency and number of threads. L stands for latency.

effectively, compared with SS under the same network latency and the same number of threads. This is because NIX performs conversion and interrupt the communication of IX in the middle. In many cases, NIX is able to finish within timeslot because of the high-performance computation facilities.

Third, based on the bar in all figures where the number of threads increases, we know SOLAR benefits from the increase of computation power. Compared to basic SS that cannot reduce communication cost even with more threads. The reason is that during execution, NIX tries to convert more operations until exceeds the timeslot. Moreover, when computation performance improves, the time spent in NIX is reduced, leading to less total cost.

V. Discussion

Extra CPU usage in SOLAR: NIX performs its task during the time slot when IX is blocked by network transmission. NIX increases the CPU usage during this timeslot which might cause concern of slowing down other tasks.

For this concern, we think extra CPU usage will not harm the performance of the system. Because 1) The performance of most of the current MPC task is bounded by the network transmission instead of CPU performance. For example, SecureML [23] transmit hundreds of GBs of data between parties but only calculates simple matrix multiplications. 2) Even if extra CPU usage harms the performance, it will only lead to possible interruption of conversion and the performance will not be worse than current SS system.

Extending to n Parties: SOLAR can be extended to an arbitrary number of parties that is based on Beaver Circuit Randomization [5]. In 1-out-of-n secret sharing where one secret is randomly partitioned into n parts and each party holds one part, the evaluation of interactive operation also follows the same pattern as specified in Section II-A. And the conversion of operation stays the same.

VI. Related Work

The communication cost has already been identified as the major cost in many SS-based MPC protocols. And many recent works tried to reduce communication rounds or the amount of data sent via network.

ABY [15] optimizes SS by mixed protocol. It profiles the time cost of all kinds of gates in different SS protocol, which enables empirical optimization on MPC applications. Specifically, ABY implements Arithmetic Sharing, Boolean Sharing, Yao's Garbled Circuit, and the conversion between these protocols. To optimize the performance of MPC program, the programmer divide the application code manually and choose the optimal protocol in each part. Compared to ABY, SOLAR does not require any prior knowledge for optimization.

Recent works such as ABY2.0 [25] and [24] reduce communication rounds by implementing multi-fan-in gates so that multiple operations could be evaluated in one communication round. SOLAR distinguishes itself with automatic generation of flattening and conversion schemes that is optimal in any network conditions.

Some works [8], [9], [12], [23], [30], [34] focus on domain specific optimization in PPML. SecureML [23] designed secret sharing schemes for matrix multiplication to reduce transmitted data size during communication. Cerebro [34] designed domain-specific language and automatically optimized the layout of the circuit in multiple parties. Compared to these works on the specific application, our work focuses on all applications constructed with secret sharing.

To the best of our knowledge, our work is the first secret sharing based MPC framework that optimizes the communication automatically without modification of MPC circuits or assumptions on application. Also, SOLAR automatically adjust itself to the network condition between parties and choose the optimal evaluation strategy in runtime.

VII. Summary

In this paper, we present SOLAR, an MPC framework with speculative dynamic communication-computation conversion strategy that converts interactive operations to non-interactive operations. The experiments' results show that SOLAR achieves 1.6-8.1× speedup over basic SS and 1.2-8.6× over static conversion. Compared to previous works, SOLAR successfully exploits more chances for optimization with conversion in runtime without concern of increasing total cost.

Acknowledgement

This paper is supported in part by Hong Kong RGC TRS T41-603/20-R, and the Turing AI Computing Cloud (TACC) [33]. We thank reviewers for their valuable comments.

REFERENCES

[1] 'bristol fashion' mpc circuits. https://homes.esat.kuleuven.be/~nsmart/MPC/, Last accessed on 28/07/2022.

[2] Divyakant Agrawal, Amr El Abbadi, Fatih Emekci, and Ahmed Metwally. Database management as a service: Challenges and opportunities. In *2009 IEEE 25th International Conference on Data Engineering*, pages 1709–1716. IEEE, 2009.

[3] Mohammad Al-Rubaie and J Morris Chang. Privacy-preserving machine learning: Threats and solutions. *IEEE Security & Privacy*, 17(2):49–58, 2019.

[4] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458, 2014.

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

[6] Amos Beimel. Secret-sharing schemes: A survey. In Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, and Chaoping Xing, editors, *Coding and Cryptology*, pages 11–46, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[7] Richard Brinkman, Jeroen Doumen, and Willem Jonker. Using secret sharing for searching in encrypted data. In *Workshop on Secure Data Management*, pages 18–27. Springer, 2004.

[8] Di Chai, Leye Wang, Kai Chen, and Qiang Yang. Efficient federated matrix factorization against inference attacks. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2022.

[9] Di Chai, Leye Wang, Junxue Zhang, Liu Yang, Shuowei Cai, Kai Chen, and Qiang Yang. Practical lossless federated singular vector decomposition over billion-scale data. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 46–55, 2022.

[10] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-psi with linear complexity via relaxed batch opprf. *Proceedings on Privacy Enhancing Technologies*, 2022(1):353–372, 2022.

[11] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. *arXiv preprint arXiv:1912.02631*, 2019.

[12] Xiaodian Cheng, Wanhang Lu, Xinyang Huang, Shuihai Hu, and Kai Chen. Haflo: Gpu-based acceleration for federated logistic regression. *arXiv preprint arXiv:2107.13797*, 2021.

[13] Seung Geol Choi, Kyung-Wook Hwang, Jonathan Katz, Tal Malkin, and Dan Rubenstein. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In Orr Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, pages 416–432, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[14] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. Spdz2k: Efficient mpc mod 2^k for dishonest majority. Cryptology ePrint Archive, Paper 2018/482, 2018.

[15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - a framework for efficient mixed-protocol secure two-party computation. In *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society, 2015.

[16] Wenliang Du and Mikhail J Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *Proceedings of the 2001 workshop on New security paradigms*, pages 13–22, 2001.

[17] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *International symposium on privacy enhancing technologies symposium*, pages 235–253. Springer, 2009.

[18] David Evans, Yan Huang, Jonathan Katz, and Lior Malka. Efficient privacy-preserving biometric identification. In *Proceedings of the 17th conference Network and Distributed System Security Symposium, NDSS*, volume 68, pages 90–98, 2011.

[19] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78, 1998.

[20] Mohammad Ali Hadavi, Ernesto Damiani, Rasool Jalili, Stelvio Cimato, and Zeinab Ganjei. As5: A secure searchable secret sharing scheme for privacy preserving database outsourcing. In Roberto Di Pietro, Javier Herranz, Ernesto Damiani, and Radu State, editors, *Data Privacy Management and Autonomous Spontaneous Security*, pages 201–216, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[21] Yehida Lindell. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*, pages 1005–1009. IGI global, 2005.

[22] Payman Mohassel and Peter Rindal. ABY3: A mixed protocol framework for machine learning. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 35–52, 2018.

[23] Payman Mohassel and Yupeng Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. *Proceedings - IEEE Symposium on Security and Privacy*, pages 19–38, 2017.

[24] Satsuya Ohata and Koji Nuida. Communication-efficient (client-aided) secure two-party protocols and its application. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 369–385, Cham, 2020. Springer International Publishing.

[25] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. {ABY2. 0}: Improved {Mixed-Protocol} secure {Two-Party} computation. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2165–2182, 2021.

[26] Arpita Patra and Ajith Suresh. Blaze: blazing fast privacy-preserving machine learning. *arXiv preprint arXiv:2005.09042*, 2020.

[27] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11478 LNCS:122–153, 2019.

[28] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based psi via cuckoo hashing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 125–157. Springer, 2018.

[29] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. Cryptology ePrint Archive, Paper 2020/1002, 2020. https://eprint.iacr.org/2020/1002.

[30] Han Tian, Chaoliang Zeng, Zhenghang Ren, Di Chai, Junxue Zhang, Kai Chen, and Qiang Yang. Sphinx: Enabling privacy-preserving online learning over the cloud. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2487–2501. IEEE, 2022.

[31] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, AISec'19, page 1–11, New York, NY, USA, 2019. Association for Computing Machinery.

[32] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. Privacy preserving vertical federated learning for tree-based models. *Proc. VLDB Endow.*, 13(12):2090–2103, jul 2020.

[33] Kaiqiang Xu, Xinchen Wan, Hao Wang, Zhenghang Ren, Xudong Liao, Decang Sun, Chaoliang Zeng, and Kai Chen. Tacc: A full-stack cloud computing infrastructure for machine learning tasks. *arXiv preprint arXiv:2110.01556*, 2021.

[34] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. Cerebro: A Platform for Multi-Party Cryptographic Collaborative Learning. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, August 2021.