The ISC Int'l Journal of Information Security
ISeCure

July 2023, Volume 15, Number 2 (pp. 163–177)
http://www.isecure-journal.org

MISC: Multi-Input Secure Two-Party Computation

Farhad Taheri [1], and Siavash Bayat-Sarmadi [1,*]
[1] *Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*

ARTICLE INFO.

Article history:
Received: January 4, 2022
Revised: August 29, 2022
Accepted: May 6, 2023
Published Online: July 1, 2023

Keywords:
Garbled Circuits, GMW Protocol, Oblivious Transfer, Privacy-Preserving Applications, Secure Function Evaluation

Type: Research Article

doi: 10.22042/isecure.2023.323624.745

dor: 20.1001.1.20082045.2023.15.2.3.2

ABSTRACT

Secure multi-party computation (MPC) allows a group of parties to compute a function on their private inputs securely. Classic MPC protocols for two parties use Yao's garbled circuit (GC) or the Goldreich-Micali-Wigderson (GMW) protocol. In this paper, we propose MISC, a multi-input secure computation protocol, by combining GC and GMW in a novel way. MISC can evaluate multi-input AND gates, which can reduce the round complexity. Moreover, MISC reduces the communication overhead by 1.7× and 2.4× for 2-input and by 2× and 2.8× for 4-input AND gates compared to the state-of-the-art GMW-style and GC-style protocols, respectively. In order to use the MISC efficiently in different applications, we redesign common building blocks with multi-input AND gates such as Equality checking, Maxpool, Comparison, and Argmax/Argmin. Results on privacy-preserving applications, e.g., circuit-based private set intersection (PSI) and private machine learning (CNN inference), show that compared to GMW, MISC improves the total communication overhead by 3× and the total run time by 1.5×.

1 Introduction

Secure computation allows two or more parties to securely evaluate a function on their private inputs while revealing nothing but the result. During the past decade, it has received significant improvements and was used in various applications such as privacy-preserving auctions [1], secure analysis of genomes [2, 3], post-quantum signature [4], and recently in the domain of privacy-preserving machine learning (PPML) [5–7]. Secure Two-party computation (STPC) protocols for Boolean circuits use either: 1) the constant round Yao's garbled circuits (GC) protocol [8], or 2) the multi-round Goldreich-Micali-Wigderson (GMW) protocol [9].

* Corresponding author.
Email addresses: `farhadtaheri@ce.sharif.edu`, `sbayat@sharif.edu`
ISSN: 2008-2045 © 2023 ISC. All rights reserved.

In the GC approach, the inputs and outputs of the circuit's gates are assigned by randomly chosen κ-bit labels, where κ is the symmetric security parameter (e.g., $\kappa = 128$). Thereby, parties cannot determine the intermediate values and input values of the other party (actual value) during the function evaluation. In GC, exclusive OR (XOR) gates can be evaluated for free [10]. However, for each AND gate, a garbled table of size 2κ-bit [11] must be sent in the setup phase using function-dependent preprocessing. Rosulek and Roy bypass this lower bound with a novel slicing and dicing technique to 1.5κ+5-bit [12]. Generating the *garbled table* requires symmetric encryption in the form of fixed-key AES operations [13]. GC-based protocols perform in the online phase symmetric-key operations for each AND gate and need substantial memory to store the garbled tables [11]. As a concrete example, to evaluate the *private set intersection* (PSI) circuit of [14] for sets of 16,384 elements consisting of 6,724,062 2-input AND gates, which requires sending a GC of size 215 MBytes. The main advantage of the GC protocol is its constant round complexity, and most of the communication can be performed in the setup phase with function-dependent preprocessing. In GMW, each party secret-shares its private inputs using an XOR-based sharing scheme. Both parties then evaluate the function with their secret shares to obtain the shares of the outputs of the circuit. The final output is obtained by combining these output shares. GMW has a cheap online phase that requires not even symmetric cryptography, but requires several communication rounds linear in the multiplicative depth of the circuit. In order to reduce the round complexity of GMW, previous works [15–17] use circuits composed of multi-input gates/tables, and lookup tables (LUTs) instead 2-input gates.

In [15], the authors focus on Beaver triple extension [18] to achieve round-efficient STPC using a semi-trusted third party. They assume that a third party can compute all pre-computations without using expensive cryptographic primitives such as homomorphic encryption (HE) or oblivious transfer (OT). The main drawback of [15] is the use of this semi-trusted party and the assumption that it does not collude with any of the other parties. In [16], the authors modify the 2-input Boolean gates in GMW to n-input lookup tables and propose two efficient protocols to evaluate lookup tables securely. They have two major bottlenecks in their work. First, extracting XOR gates from lookup tables requires post-processing efforts, and extracting all XOR gates is still challenging for real-world applications. Second, there is no conversion between the protocols proposed in [16] and other 2PC protocols, which make it inflexible in practice. ABY 2.0 [17] is an efficient mixed-protocol STPC framework that improves over GMW and contains the previously most efficient construction for multi-input AND gates. However, this protocol increases the total communication overhead. Moreover, in previous works [19, 20], the authors propose a mixed-protocol framework to achieve higher performance. For instance, they employ an arithmetic sharing protocol for linear functions and GMW or GC for non-linear functions for a single application. In this work, to reduce the communication complexity, we combine ideas of GC and GMW to design an STPC protocol called MISC. Moreover, this protocol can evaluate multi-input AND gates to reduce the round complexity. To improve flexibility, we propose a conversion protocol between MISC and other STPCs.

1.1 Our Contributions

We propose a multi-input secure two-party computation protocol called MISC. Our work combines both STPC protocols, Yao's GC and GMW similar to [21], on the gate level, which reduces communication and round complexity. Moreover, MISC can convert to other STPC protocols such as arithmetic sharing, GMW, and GC. Our proposed protocol reduces the communication overhead for 2-input AND gate by 1.7×. Furthermore, by evaluating one N-input AND gate instead of (N-1) 2-input AND gates, the proposed protocol can reduce the number of rounds. We evaluate our protocol on several privacy-preserving applications, such as private machine learning and private set intersection. In summary, our main contributions are as follows:

- We propose MISC, a secure two-party computation protocol that reduces round complexity and OT overhead for multi-input AND gates without adding computation complexity in the setup and online phases. MISC can reduce the communication overhead by 1.7× for each 2-input AND gate and 2× for each 4-input AND gate compared to ABY2.0 [17].
- We introduce conversions between MISC, arithmetic sharing, and GC with low overhead. This enables using MISC more efficiently in privacy-preserving applications.
- We redesign standard building blocks such as Equality Check, Maxpool/Minpool, and Argmax/Argmin using multi-input AND gates.
- We also improve the total communication overhead to 3× and the total time up to 1.7× in these applications.

We explain preliminaries and related works in Section 2. In Section 3, our MISC protocol and also the underlying OT protocol are described. Conversions between MISC and other STPC protocols are provided in Section 4. In Section 5 and Section 6, we present extensive evaluations of MISC. Finally, we conclude this work in Section 7.

2 Preliminaries and Previous Works

In this section, we provide preliminaries and previous works for STPC and define our security model.

2.1 Semi-Honest Security Model

In this work, we work in the semi-honest (passive or honest-but-curious) security model. In this model, each party follows the protocol but wants to learn information about the other party's inputs or intermediate values that cannot be deduced from its inputs or the output. Although there is no guarantee for each party to not deviate from the protocol [22], it allows constructing of highly efficient protocols [9–12, 16, 17, 23, 24]. Moreover, such protocols can be used as stepping towards protocols with stronger security [25].

2.2 Oblivious Transfer

Oblivious Transfer (OT) is one of the main building blocks for secure computation [26]. In OT_1^n the sender S wants to send a secret to the receiver R. S provides n inputs $(s_1, s_2, \ldots, s_n)$ to OT and R inputs a number r $(1 \leq r \leq n)$. As output, R obliviously receives s_r and does not learn any information about the other inputs of S, whereas S does not learn which input was selected by R.

OT is a fundamental building block for STPC and requires expensive public-key cryptography [27]. In [28], the OT Extension technique was introduced, which allows to generate many OTs from a small number (equal to the security parameter) of base OTs using only fast symmetric-key cryptography.

Concretely, the OT Extension implementation of [28] generates around 1 million OT_1^2 per second with semi-honest security. Random OT (R-OT) is a special-purpose OT functionality, tailored for more efficient secure computation [28]. In an R-OT_1^2, the sender inputs no messages to the OT protocol, but receives two messages m_0, m_1 as a random protocol output. In contrast, the receiver still inputs its selection bit b and obtains the chosen message m_b as output. OT precomputations [29] allows to move all cryptographic operations of OT to a setup phase, resulting in a very efficient online phase.

2.3 Related Work on STPC

In this section, we discuss the main secure two-party computation protocols.

2.3.1 Yao's Garbled Circuit

Yao introduced a secure two-party computation protocol in the 1980s [8] called Garbled Circuit (GC). The GC protocol has a constant number of rounds and no interaction in the online phase using function-dependent preprocessing. However, the input-dependant online phase requires symmetric encryption. Thereby, the GC protocol is suitable for WAN networks [19]. Today's most efficient instantiation of the GC protocol allows free XOR gates [10], and each AND gate require symmetric cryptographic operations using fixed-key AES [13] and sending 2κ-bits [11] which was recently improved to 1.5κ+5-bits [12].

2.3.2 Secret Sharing (GMW)

The GMW protocol [9] uses secret sharing to compute the function by both parties interactively. Like GC, functions in the GMW protocol are represented as the Boolean circuits. In this protocol, each party has a share for each wire. For instance, for a wire with value v, one party has share v_1, and the other has share v_2 s.t. $v = v_1 \oplus v_2$. Each XOR gate can be computed without interaction by each party locally XORing its shares, and thus evaluation of XOR gates is free. Unlike XOR gates, AND gates need multiplication triples for evaluation [30], and these can be precomputed efficiently using OT extension [28]. This makes GMW a non-constant round in the online phase. Note that each AND gate of the same layer in the circuit can be evaluated in parallel. However, the evaluation time increases with increasing circuit depth. Nevertheless, the GMW protocol has low online computation and communication complexity due to the lack of symmetric encryption in the online phase, and labels are only 1-bit. Hence, the GMW protocol is appropriate for LAN networks [19].

The GMW protocol, despite little communication and computation overhead, has a multi-round online phase. This round complexity makes it inefficient in many real-world applications [15–17, 23]. [15] shows that 99% of the online time is caused by communication latency.

Several works improved over the basic GMW protocol: [15] efficiently evaluated multi-input AND gates using a trusted third party. Besides using the trusted party assumption, this protocol has significant computation overhead for gates with many inputs. [16] proposed a new protocol to evaluate functions with Lookup Tables (LUT) rather than Boolean gates to reduce round complexity. To this end, the authors designed two protocols for the LUT-based circuit called OP-LUT for optimized online communication, and SP-LUT for optimized setup/total communication. ABY 2.0 [17] is an efficient mixed-protocol STPC protocol. It uses a different perspective on Beaver's circuit randomization [30] and considers multi-input AND gate evaluation.

2.3.3 Combination of GC + GMW for 2-Input Gates

Yalame *et al.* [21] proposed a new protocol based on combining GC and GMW to reduce the communication overhead of 2-input AND gates. In this protocol, they considered two parties named the garbler and the evaluator similar to the GC protocol. However, evaluating a 2-input AND gate, they used OT, similar to the GMW protocol. The garbler generates the garbled table with a random 2-bit label without encryption and sends the labels for the circuit's inputs to the evaluator, similar to GC. However, for evaluating AND gates, the parties use OT_1^2 based on the gate's input labels. This protocol reduces the communication complexity from OT_1^4 (or $2\times$ R-OT_1^2 [28]) in

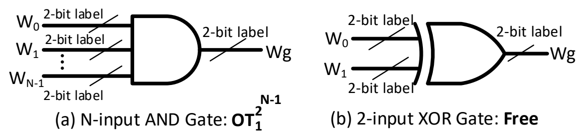


Figure 1. Gates and costs in MISC

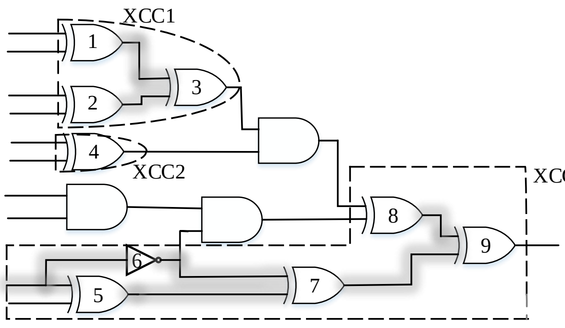


Figure 2. Partitioning XOR gates in a circuit into XOR Connected Components (XCCs) (We highlight the XCC connection with gray color)

GMW to $1\times\mathrm{OT}^{2}_{1}$. In our work, we extend the protocol of [21] from 2-input to multi-input gates and provide conversion to other secure computation protocols.

3 Our Proposed Scheme: MISC

In this work, we design the multi-input secure two-party computation protocol MISC. To this end, we propose a new protocol based on GC and GMW features similar to [21]. As mentioned earlier, round complexity is one of the primary disadvantages of GMW. Our protocol reduces the round complexity and the communication cost of GMW-style protocols. In MISC, XOR gates are evaluated for free, and we use OT for each AND gate in the circuit, similar to the GMW-style protocols. Furthermore, the label size in MISC is 2-bit. The online computation cost for MISC is much less than GC with 128-bit labels. We also need no symmetric cryptography, such as AES, to evaluate each AND gate in the online phase. Figure 1 shows the gates used in MISC.

MISC reduces R-OT^{4}_{1} or 2× R-OT^{2}_{1} in [16] to R-OT^{2}_{1} for 2-input AND gates. For multi-input AND gates, MISC reduces the OT overhead in the most recent GMW-style protocol [17]. MISC is an improved version of the protocol of [21], and we make it compatible with multi-input AND gates to reduce the round complexity. Also, MISC conversions are designed and optimized to be used efficiently in applications.

3.1 Notations

We write $x \oplus y$ for bitwise XOR between x and y with the same length and $a|b$ for concatenation of a and b. We denote $\in_R$ for uniform random choice, and $\{0,1\}^n$ denotes a binary string with length n. κ denotes the symmetric security parameter. We denote $\langle x\rangle^t$ as a shared variable, and $t \in \{R, A, Y\}$ indicates the type of sharing, where R denotes the MISC protocol, A denotes Arithmetic sharing, and Y denotes Yao.

W^{α}_{β} denotes the wire label with logical value α on the wire β. In the STPC, if one party knows the value W^{α}_{β}, he cannot find α. Also, given W^{α}_{β}, one cannot obtain $W^{1-\alpha}_{\beta}$ for 0 and 1 values.

As in [21], we also make use of XOR connected components (*XCCs*) in MISC. In an *XCC*, the XOR gates are connected without any AND gate. We define xcc() as a function to identify the *XCC* of each XOR gate. n_{XCC} represents the number of *XCCs* in the circuit. Figure 2 shows a circuit with three *XCCs*. For instance, in this figure, gates 1, 2, and 3 create the XCC condition. The limitation of protocol in [21] and MISC is the *XCC* problem, which means that to preserve security, the inputs of an AND gate must not come from one XOR group. In the *XCC*, the number of states is reduced, which may lead to a security problem. More information about the *XCC* condition is explained in [21].

3.2 MISC Approach

The main goal of our protocol is to reduce the communication and round complexity of AND gates in GMW-style protocols. Similar to the GC protocol [8], MISC uses a garbled table for each gate, and evaluating XOR gates is free. For the sake of simplicity, we described the protocol for a 3-input AND gate. The routine for AND gates with more inputs is similar.

To design our round-efficient protocol, we use a 2-bit *label* for each wire. Following the point-and-permute technique for GC [24], the most significant bit (MSB) of the wire label is different for the 0 and 1 values. The relation between the MSB and the actual value of the wire is random and secret. Table 1 shows a sample garbled table for a 3-input AND gate. For each row in the garbled table, we create a 2-bit index, the concatenation of the MSBs of W_b and W_c (i.e., *index* $= MSB(W_b)|MSB(W_c)$). For instance, as shown in the first row of Table 1, if the actual value is 0 for wire b and 0 for wire c, the random labels of these wires are 11 and 01, so the *index* is 10. The label has another randomly selected bit, the least significant bit (LSB), in addition to the MSB.

Each 3-input AND gate has six input labels of 2-bit: W^0_a, W^1_a, W^0_b, W^1_b, W^0_c, W^1_c. Although these have 12 bits, there are only $2^{12}/2^3 = 512$ different possibilities because the MSBs of the two labels per wire are opposite. We construct $2^6 = 64$ different *garbling groups*

Table 1. Sample garbling for 3-input AND gate (The first four columns are the actual values of input and output wires)

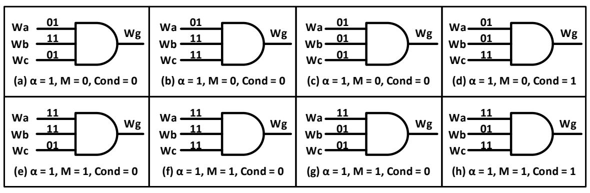
Input			Output	Garbled Input			Garbled Output	Index
a	b	c	g	Wa	Wb	Wc	Wg	
0	0	0	0	01	11	01	W_g^0	10
0	0	1	0	01	11	11	W_g^0	11
0	1	0	0	01	01	01	W_g^0	00
0	1	1	0	01	01	11	W_g^0	01
1	0	0	0	11	11	01	W_g^0	10
1	0	1	0	11	11	11	W_g^0	11
1	1	0	0	11	01	01	W_g^0	00
1	1	1	1	11	01	11	W_g^1	01

based on the six LSBs of the input labels. The 6-bit number of each group is obtained from the concatenation of the six LSBs of the input labels. Each garbling group has eight values are determined based on all possible combinations of the three input labels of the AND gate. The evaluator who has the input's labels computes $\alpha = LSB(W_a \oplus W_b \oplus W_c)$. Also, the garbler who knows the garbling group calculates the following parameters for each garbling group as follows:

$$\begin{aligned} \alpha_b &= LSB(W_b^0 \oplus W_b^1), \\ \alpha_c &= LSB(W_c^0 \oplus W_c^1), \\ \alpha_d &= LSB(W_a^1 \oplus W_b^1 \oplus W_c^1). \end{aligned} \tag{1}$$

Parameters α_b, α_c, α_d are the same for each garbling group. For a better explanation, we show the garbling groups (*G0* to *G63*) of a 3-input AND gate in Table 2. Each AND gate corresponds to one cell in Table 2. The garbler determines the table's row and has no information about the column and the evaluator's input label. The evaluator only knows the input's labels (columns in Table 2) and has no information about the number of the garbling group (row in Table 2). It would be ideal if the evaluator could use α as input for OT_1^2 and receive the output label from the garbler. However, as shown in Table 2, for example, in the group G0, $\alpha = 0$ for all input values. Therefore, two other parameters, called *Cond* and M, are used.

The *Cond* value is 1 for each cell with an asterisk in Table 2. The evaluator calculates the *Cond* parameter by comparing the AND gate index bit with the 2-bit number ρ that the garbler sends to the evaluator. The garbler computes ρ by concatenating the MSBs of $W_b^{\bar{\alpha}_b}$ and $W_c^{\bar{\alpha}_c}$: $\rho = MSB(W_b^{\bar{\alpha}_b})|MSB(W_c^{\bar{\alpha}_c})$. If ρ equals the index, the evaluator sets *Cond* to 1, and 0 otherwise. For instance, in Figure 3, α and *Cond* are calculated for eight states of an AND gate. This gate belongs to the garbling group G63 ($LSB(W_a^0)|LSB(W_a^1)|LSB(W_b^0)|LSB(W_b^1)|LSB(W_c^0)|LSB(W_c^1) = (111111)_2 = 63$) and *Cond* is 1 when the inputs are equal to 011 or 111 (cf. Table 2). In the 2-input AND gate, the evaluator can use $\alpha \oplus Cond$ as input for OT_1^2 to receive the output label [21]. The result of $\alpha \oplus Cond$ for selecting the output label between W^0 and W^1 is always different. The relation between this bit and the output value of the gate is uniformly random [21]. The highlighted part of Table 2 indicates the corresponding garbling groups for a 2-input AND gate. However, for an AND gate with more than two inputs, we cannot use $\alpha \oplus Cond$ because the value of this equation is not different for the 0 and 1 output labels. For example, as shown in Table 2, the value of $\alpha \oplus Cond$ is the same in columns 3 and 7 while we want the value of $\alpha \oplus Cond$ for column 7 to be different from the other columns. So we need one more bit to use in OT_1^4 for a 3-input AND gate. We use this bit to divide the table into two parts. To this end, we use a bit referred to as M. This bit is the MSB of the label W_a: $M = \mathrm{MSB}(W_a)$.



Wa 01 Wb 11 Wc 01 Wg
(a) α = 1, M = 0, Cond = 0
Wa 01 Wb 11 Wc 11 Wg
(b) α = 1, M = 0, Cond = 0
Wa 01 Wb 01 Wc 01 Wg
(c) α = 1, M = 0, Cond = 0
Wa 01 Wb 01 Wc 11 Wg
(d) α = 1, M = 0, Cond = 1
Wa 11 Wb 11 Wc 01 Wg
(e) α = 1, M = 1, Cond = 0
Wa 11 Wb 11 Wc 11 Wg
(f) α = 1, M = 1, Cond = 0
Wa 11 Wb 01 Wc 01 Wg
(g) α = 1, M = 1, Cond = 0
Wa 11 Wb 01 Wc 11 Wg
(h) α = 1, M = 1, Cond = 1

Figure 3. Determining α, M, and *Cond* for eight 3-input AND gates as an example

The evaluator calculates the index bit by concatenating the MSB of the labels W_b and W_c; then compares it with ρ to compute *Cond*. In the next step, the evaluator concatenates $M = MSB(W_a)$, and $\alpha \oplus Cond$; into a 2-bit number as input for OT_1^4. On the other hand, the garbler provides the inputs for OT_1^4 according to the value of $(\alpha_b, \alpha_c, \alpha_d)$. The evaluator then obtains the corresponding AND gate output label from OT_1^4. The details for OT inputs are given in Section 3.3 and Section 3.4, and the overhead of the OT is investigated in Section 3.6.

A similar approach can be used for an AND gate with more than three inputs. For example, in a 4-input AND gate, the garbler sends ρ, the concatenation of the MSBs of W_c and W_d, to the evaluator. The evaluator calculates *Cond* and provides *Cond* with two other MSBs of W_a and W_b as the input for OT_1^8 till the output label is determined. Thus, this protocol can evaluate an N-input AND gate with $\mathrm{OT}_1^{2^{N-1}}$ in two rounds.

The only problem of this scheme is the evaluation of the AND gates when inputs come from one *XCC*. For these gates, we cannot run this protocol and must use typical STPC protocols such as GMW without improvement. However, this is a rarely-happening condition in privacy-preserving applications. It never happens in the building blocks we investigate in Section 5:

Table 2. All possible Garbling Groups for 3-input AND gates (An asterisk shows the cell when $Cond = 1$ and the highlighted part of the table indicates the garbling groups for 2-input AND gates)

		Garbled input							
$LSB(W_a^0) \| LSB(W_a^1) \|$ $LSB(W_b^0) \| LSB(W_b^1) \|$ $LSB(W_c^0) \| LSB(W_c^1)$	**Garbling Group**	W_a^0 W_b^0 W_c^0	W_a^0 W_b^0 W_c^1	W_a^0 W_b^1 W_c^0	W_a^0 W_b^1 W_c^1	W_a^1 W_b^0 W_c^0	W_a^1 W_b^0 W_c^1	W_a^1 W_b^1 W_c^0	W_a^1 W_b^1 W_c^1
000000	**G0**	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$
000001	**G1**	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$
000010	**G2**	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$
000011	**G3**	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$
000100	**G4**	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$
000101	**G5**	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$
000110	**G6**	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 1$
000111	**G7**	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$
001000	**G8**	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0$
001001	**G9**	$\alpha = 1$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$
001010	**G10**	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$
001011	**G11**	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$
001100	**G12**	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$
001101	**G13**	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$
001110	**G14**	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$
001111	**G15**	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$
010000	**G16**	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$
010001	**G17**	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$
010010	**G18**	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$
010011	**G19**	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$
010100	**G20**	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$
010101	**G21**	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$
010110	**G22**	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$
010111	**G23**	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$
011000	**G24**	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$
011001	**G25**	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$
011010	**G26**	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$
011011	**G27**	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$
011100	**G28**	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$
011101	**G29**	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$
011110	**G30**	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$
011111	**G31**	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$
100000	**G32**	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$
100001	**G33**	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$
100010	**G34**	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$
100011	**G35**	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$
100100	**G36**	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$
100101	**G37**	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$
100110	**G38**	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$
100111	**G39**	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$
101000	**G40**	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$
101001	**G41**	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$
101010	**G42**	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$
101011	**G43**	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$
101100	**G44**	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$
101101	**G45**	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$
101110	**G46**	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$
101111	**G47**	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$
110000	**G48**	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$
110001	**G49**	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$
110010	**G50**	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$
110011	**G51**	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$
110100	**G52**	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$
110101	**G53**	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$
110110	**G54**	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$
110111	**G55**	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$
111000	**G56**	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 1$
111001	**G57**	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0$
111010	**G58**	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 1$
111011	**G59**	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 0$
111100	**G60**	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0$	$\alpha = 0^*$
111101	**G61**	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1^*$	$\alpha = 0$	$\alpha = 1$
111110	**G62**	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1$	$\alpha = 0^*$	$\alpha = 1$	$\alpha = 0$
111111	**G63**	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1$	$\alpha = 1^*$
actual output		**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**

Equality check, Maxpool, and Argmin.

If the circuit face with the XCC condition, which means the input of the AND gate comes from one XCC (cf. Figure 2), our proposed scheme uses GMW: The garbler uses OT_1^8 to send the output label to the evaluator for a 3-input AND gate.

3.3 MISC Garbling

The MISC garbling is given in Algorithm 1 and summarized as follows:

- **Step 1**: The scheme needs a random bit for each XCC to work. The algorithm starts by generating an n_{XCC}-bit random number, namely S. Each bit in S corresponds to one XCC in the circuit.
- **Step 2**: Labels of all wires connected to XOR gates are assigned as follows:
- **Step 2.1**: XOR gates are processed in topological order, and their XCC are determined.
- **Step 2.2**: For each XOR gate, if its input wires are unlabeled, label values are assigned. For an input wire, the label value W^0 (corresponding to the actual value zero) is a random number. Its label W^1 (corresponding to one) is calculated by XORing W^0 and R, where R is a 2-bit number with its MSB equal to one and its LSB equal to the corresponding bit of the current XCC. The label of the output wire is obtained by XORing the corresponding input garbled values (this is similar to the free XOR technique in GC [10]).
- **Step 3**: Labels are assigned to the remaining unlabeled wires (AND gates without $XCCs$).
- **Step 4**: As in Yao's GC, the garbler sends the input labels to evaluator using OT_1^2.
- **Step 5**: AND gates are evaluated interactively in this step.
- **Step 5.1**: α_b, α_c, and α_d are computed for the current AND gate.
- **Step 5.2**: A 2-bit value ρ is calculated according to the input labels of the current AND gate and its values α_b and α_c. The parameter ρ is then sent to the evaluator.
- **Step 5.3**: The garbler acts as the sender of OT_1^4 with inputs according to $(\alpha_b, \alpha_c, \alpha_d)$.
- **Step 6**: In this step, the garbler provides the evaluator with the required information to decrypt the output wires by sending the MSB of the zero output labels (this is similar to Yao's GC [12]).

3.4 MISC Evaluation

The MISC evaluation is given in Algorithm 2 and summarized as follows:

Algorithm 1 MISC Garbling

```
    Procedure GARBLE
1:  $S \in_R \{0,1\}^{n_{XCC}}$
2:  for each XOR gate g with input a, b (in topological order)
2.1:    $R \leftarrow 1|S[xcc(g)]$
2.2:    if (label value of $a$ has not been assigned)
            $W_a^0 \in_R \{0,1\}^2$, $W_a^1 \leftarrow W_a^0 \oplus R$
        if (label value of $b$ has not been assigned)
            $W_b^0 \in_R \{0,1\}^2$, $W_b^1 \leftarrow W_b^0 \oplus R$
        $W_g^0 \leftarrow W_a^0 \oplus W_b^0$, $W_g^1 \leftarrow W_g^0 \oplus R$
    end for
3:  for each wire $i$ that label values have not been assigned
      $W_i^0 \in_R \{0,1\}^2$, $tmp \in_R \{0,1\}$
      $W_i^1 \leftarrow \overline{MSB(W_i^0)}|tmp$
    end for
4:  for each input wire $i$ of the circuit
        Send corresponding label $W_i$ to evaluator (using $\text{OT}_1^2$)
5:  for each AND gate g with inputs a, b, c
5.1:    $\alpha_b \leftarrow LSB(W_b^1 \oplus W_b^0)$
        $\alpha_c \leftarrow LSB(W_c^1 \oplus W_c^1)$
        $\alpha_d \leftarrow LSB(W_a^1 \oplus W_b^1 \oplus W_c^1)$
5.2:    $\rho \leftarrow MSB(W_b^{\bar{\alpha}_b})|MSB(W_c^{\bar{\alpha}_c})$
        send $\rho$ to the evaluator
5.3:    provide the inputs of $\text{OT}_1^4$:
        $W_g^{[LSB(W_a^0)\oplus\alpha_d\oplus(\alpha_b+\alpha_c)]\cdot\overline{MSB(W_a^1)}}$.
        $W_g^{\overline{[LSB(W_a^0)\oplus\alpha_d\oplus(\alpha_b+\alpha_c)]}\cdot\overline{MSB(W_a^1)}}$.
        $W_g^{[LSB(W_a^0)\oplus\alpha_d\oplus(\alpha_b+\alpha_c)]\cdot MSB(W_a^1)}$.
        $W_g^{\overline{[LSB(W_a^0)\oplus\alpha_d\oplus(\alpha_b+\alpha_c)]}\cdot MSB(W_a^1)}$
    end for
6:  for each output wire $i$ of the circuit
        $d_i \leftarrow MSB(W_i^0)$
        send $d_i$ to the evaluator
    end for
```

- **Step 1**: As in Yao's GC, the evaluator obtains the labels corresponding to all input wires of the circuit. The ones corresponding to the garbler's inputs are sent directly, and the ones corresponding to the evaluator's input wires are obtained via OT_1^2.
- **Step 2**: In this step, the evaluator processes all gates in topological order.
- **Step 2.1**: If the corresponding gate is an XOR, he simply XORs both labels of the inputs of the gate (similar to free XOR in GC [10]).
- **Step 2.2**: If the corresponding gate is an AND, he XORs the LSBs of the input labels to obtain α. Now, the evaluator performs the following actions:
- **Step 3**: Receives ρ_g from garbler.
- **Step 3.1**: It compares the concatenation of W_b

Algorithm 2 MISC evaluation

```
   Procedure EVALUATE
1: for each input wire $i$ of the circuit
       Receive corresponding label $W_i$ from
       garbler (using $\text{OT}_1^2$ for ealuator's inputs)
2: for each gate $g$ (in topological order)
   2.1: if ($g$ == XOR with inputs a, b)
            $W_g \leftarrow W_a \oplus W_b$
   2.2: if ($g$ == AND with inputs a, b, c)
            $\alpha \leftarrow LSB(W_a \oplus W_b \oplus W_c)$
            $M \leftarrow MSB(W_a)$
3: Receive $\rho_g$ from garbler
   3.1: if ($MSB(W_b)|MSB(W_c) == \rho_g$)
                 $OT_{in} \leftarrow M|\bar{\alpha}$
             else
                 $OT_{in} \leftarrow M|\alpha$
   3.2:      provide $OT_{in}$ as select input of $\text{OT}_1^4$
             and receive its output $W_g$.
   end for
4: for each circuit output wire $i$
       Receive $d_i$ from garbler
       $y_i \leftarrow d_i \oplus MSB(W_i)$
   end for
```

and W_c against ρ_g received from the garbler. Based on these comparisons and MSB(W_a), the evaluator determines OT$_{in}$, the selection input to OT_1^4.

- **Step 3.2**: The garbler and the evaluator execute OT_1^4, where the evaluator acts as the receiver with input OT_{in} and the garbler acts as the sender. As output, the evaluator obtains the label value W_g of the AND gate output.
- **Step 4**: In this step, similar to Yao's GC [12], the evaluator decodes his circuit output wires y_i.

3.5 Security Proof

As described in Section 2.1 we work in the semi-honest adversary model in which the adversary is not allowed to deviate from the protocol. The structure of our scheme is based on the GC and GMW protocols [9, 31]. The proposed scheme follows these protocols with some additional steps. On the garbler side, all the performed actions can be summarized as follows.

(1) The garbler sets a random label for each wire in the circuit. Then, for each AND gate, based on the garbling group in Table 2 computes α_b, α_c, α_d, and ρ. The garbler sends ρ to the evaluator.
(2) The garbler sends the evaluator's input labels with OT. Also, it sends its input labels to the evaluator as in GC [12].
(3) The garbler sends the output label with OT_1^4 for each 3-input AND gate in the online phase. Due to the security of OT, the garbler learns nothing about the output label selected by the evaluator, who learns nothing about the other output label.
(4) The garbler provides the evaluator with the required information to decrypt the output wires by sending the MSB of the zero output labels as in GC [12].

The actions performed on the evaluator side are as follows.

(1) Receives the input label from the garbler as in GC [12].
(2) Calculates the garbled output of XOR gates by XORing its garbled inputs same as the freeXOR technique in GC [10].
(3) Receives ρ for each AND gate.
(4) Receives the AND gate output label with OT as in GMW [9]. In this step, the evaluator calculates α, M and, *cond* parameters and determines the selection input of OT_1^4. The evaluator has no information about the garbling group (which corresponds to the rows in Table 2), and the garbling table is symmetric and leaks no information to the evaluator.
(5) Calculates and sends the circuit's output value as in GC [12].

The actions mentioned above are similar to previous work on GC and GMW [9, 31]. The garbler in MISC builds garbling groups for each AND gate. Then sends the input labels of the circuit to the evaluator as in GC. Unlike GC, the evaluator uses OT based on the input labels to determine the output label. The evaluator receives only the output label but nothing about the other output labels. So he cannot learn anything from guessing the other input labels to reach the other output labels. The evaluator knows nothing about the garbling group, so it can not reproduce the values on the garbler side. The ρ parameter used to determine *Cond* on the evaluator side leaks no information to the evaluator because the *Cond* is uniformly random in the garbling groups (cf. Table 2). In MISC, similar to GC, the garbler learns nothing about the evaluator's input and output label selected by the evaluator.

3.6 Using Random OT

We use Random-OT (R-OT) [16, 28] for evaluating AND gates in the setup phase. In an R-OT, both sender and receiver obtain their input as a random output of the protocol. In MISC we need R-$\text{OT}_1^{2^{N-1}}$ instead of (N-1) R-OT_1^4 for an N-input AND gate [16]. To improve communication, we use the R-OT_1^n protocol to reduce $\binom{n}{1}$ $\text{OT}^1_{\log_m n}$ to $\binom{m}{1}$ $\text{OT}_1^{\log_m n}$ as in [16]. We vary possible choices for n and m and observe that the highest improvement is obtained when we choose $n = 16$ for $m = 2$ and $m = 4$, and $n = 64$ for $m = 8$.

Table 3. Evaluation of an N-input AND gate with MISC in comparison to previous works (Best results are marked in bold)

	N-Input AND	**Setup [bits]**	**Online [bits]**	**Total [bits]**	**Online Rounds**
2-input AND	2-input AND GC [12]	197	-	197	-
	2-input AND [16]	134	4	138	**1**
	2-input AND gate [21]	**77**	5	**82**	**1**
	2-input AND [28]	256	4	300	1
	2-input AND (MISC)	**77**	5	**82**	2
3-input AND	2 × 2-input AND GC [12]	394	-	394	-
	2 × 2-input AND [16]	268	8	276	2
	2 × 2-input AND gate [21]	154	10	164	2
	2 × 2-input AND (MISC)	154	10	164	2
	3-input AND (MISC)	**136**	10	**146**	2
4-input AND	3 × 2-input AND GC [12]	591	-	591	-
	3 × 2-input AND [16]	402	12	414	2
	3 × 2-input AND [21]	231	15	246	2
	3 × 2-input AND (MISC)	231	15	246	2
	4-input AND (MISC)	**189**	19	**208**	2
5-input AND	4 × 2-input AND GC [12]	788	-	788	-
	4× 2-input AND [16]	536	16	552	3
	4 × 2-input AND [21]	308	20	328	3
	4× 2-input AND (MISC)	308	20	328	3
	5-input AND (MISC)	**238**	36	274	**2**

By doing so, we have 75 bits of communication for R-OT$^2_1$, 134 bits for R-OT$^4_1$, 187 bits for R-OT$^8_1$, and 236 bits R-OT$^{16}_1$. The online communication per N-Input AND gate is $(N-1)+2^N$, which is the same as in the SP-LUT protocol in [16].

Recently, [32] proposed a very efficient silent OT protocol named Silent OT which claims to outperform state-of-the-art solutions for performing R-OT. Also, a semi-honest OT protocol with sublinear communication and linear computation was proposed in [33]. Since MISC makes black-box calls to R-OT, it can directly benefit from the performance improvements in [32] and [33].

3.7 Complexity of Multi-Input ANDs

We evaluate the complexity of N-input AND gates in our MISC protocol. In Table 3, we show the communication and round complexity for an N-input AND gate and compare MISC with GMW in [16], GMW in [28], and 2-input GC+GMW [21]. We also compare MISC with the state-of-the-art GC construction of [12]. We set $N = \{2, \dots, 5\}$ in our evaluation. As MISC employ the OT in the online phase, it requires two round in the online phase for each AND gate. However, it can reduce the round complexity for more than 4-input AND gates, compared to GMW.

Table 3 shows: (1) MISC reduces the overall communication, by 1.99× compared to the best GMW, by 1.18× compared to [21] for 4-input AND gates; (2) MISC reduces the round complexity by 1.5× compared to GMW and [21] in 5-input AND gates; (3) Our protocol needs a few more bits of online communication; however, this has a negligible effect on the total time; (4) MISC has 2.84× lower communication than the state-of-the-art GC of [12] for 4-input AND gates.

Table 4. Conversions complexity between MISC, Arithmetic, and Yao sharing (The values are reported for l-bit values and symmetric security parameter κ)

Conv	Setup	Online	
	[bits]	[bits]	Rounds
M2Y	$2l\kappa$	$l\kappa + l$	2
Y2M	0	2	0
A2M	$4l\kappa$	$2l\kappa + l + 2$	2
M2A	$2l\kappa$	$(l^2 + 3l)/2$	2

4 Mixed Protocol Conversions

This section describes the conversion method between our MISC protocol and Yao and Arithmetic sharing in ABY [19]. With these conversions, we can use our protocol efficiently in various application. For instance, in private ML applications such as convolutional neural networks (CNN), Arithmetic sharing is very efficient for convolutional and fully connected layers [37, 38]. MISC is the best candidate to use in pooling layers. The round and communication complexity for converting between MISC, Yao, and Arithmetic sharing are summarized in Table 4.

4.1 M2Y

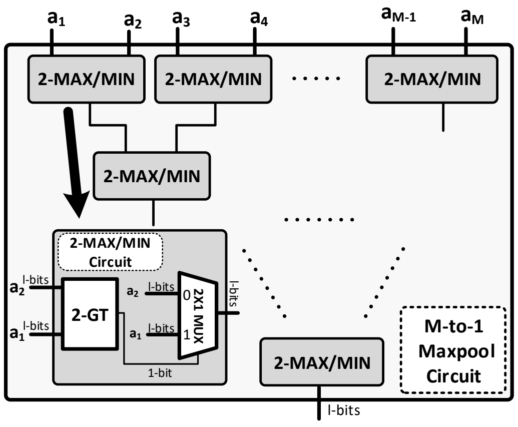
Converting a MISC sharing $\langle x\rangle^R$ to a Yao sharing $\langle x\rangle^Y$ is similar to converting a GMW sharing to Yao [19]. To do so P_i locally sets $x_i = MSB(\langle x\rangle^R_i)$. Then, P_0 samples $\langle x\rangle^Y_0 = k_0 \in_R \{0,1\}^\kappa$ and acts as sender in OT$^2_1$ with inputs $(k_0 \oplus x_0 \cdot \delta; k_0 \oplus (1-x_0)\cdot\delta)$ where δ is the key offset of the free XOR technique [10]. P_1 inputs x_1 to OT and receives the label $\langle x\rangle^Y_1 = k_0 \oplus (x_0 \oplus x_1)\cdot\delta$ as output.

4.2 Y2M

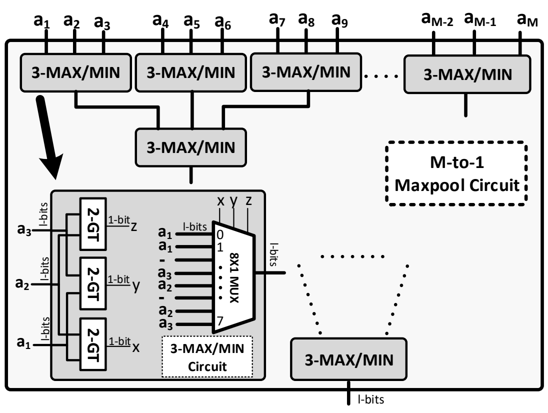
Converting a Yao share $\langle x\rangle^Y$ to a MISC share $\langle x\rangle^R$ is the easiest conversion and is almost for free. Note that the LSB of the Yao share is the permutation bit of the point-and-permute technique [24]. Therefore, the labels $\langle x\rangle^Y_0$ and $\langle x\rangle^Y_1$ already form a valid MISC share of κ-bits. Thus, P_0 locally sets $\langle x\rangle^R_0[1|0] = \langle x\rangle^Y_0[0|1]$ and $\langle x\rangle^R_1[1|0] = \langle x\rangle^Y_1[0|1]$ in the setup phase. P_1 locally sets $\langle x\rangle^R_x[1|0] = \langle x\rangle^Y_x[0|1]$ in the online phase. P_0 sends to P_1 the 2-bit ρ according to Table 2.

4.3 A2M

Converting an Arithmetic share $\langle x\rangle^A$ to a MISC share $\langle x\rangle^R$ can be done by securely evaluating a Boolean addition circuit as for the A2Y conversion in [19]. As the Y2M conversion comes for free, we simply have $\langle x\rangle^R = A2M(\langle x\rangle^A) = Y2M(A2Y(\langle x\rangle^A))$. More precisely, the parties secret share their Arithmetic shares $x_0 = \langle x\rangle^A_0$ and $x_1 = \langle x\rangle^A_1$ as $\langle x\rangle^Y_0$ and $\langle x\rangle^Y_1$, as

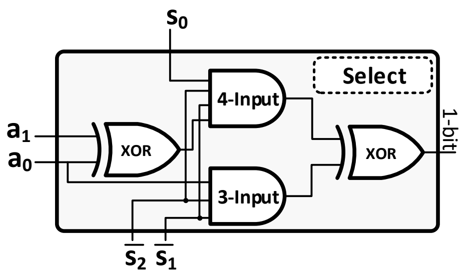


(a) Using 2-MAX circuit [34]

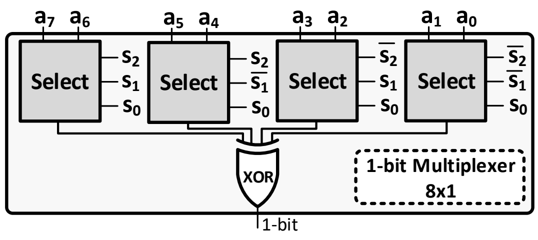


(b) Using 3-MAX circuit [17]

Figure 4. M-to-1 Max/Min circuit



(a) Select component (similar to Y gate in [10])



(b) 8×1 Multiplexer [35]

Figure 5. The new 1-bit 8×1 Multiplexer is compatible with multi-input AND gates

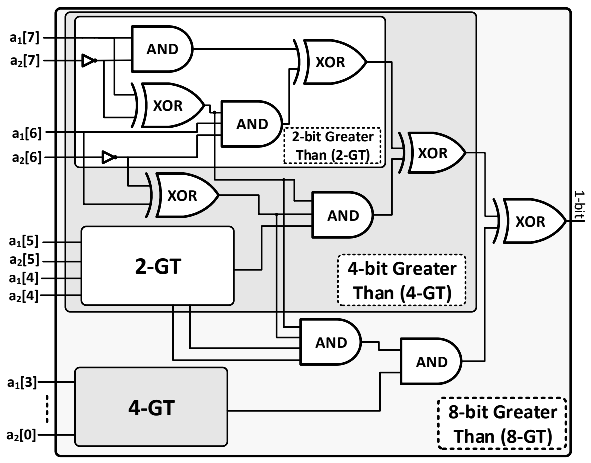


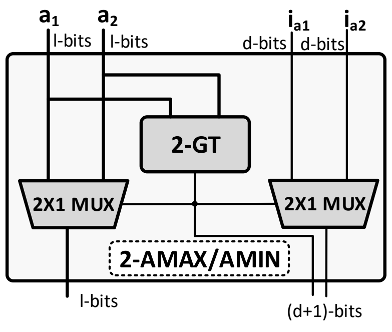
Figure 6. Our new 8-bit Greater Than circuit with multi-input AND gates based on [36]

described in [19], and compute $\langle x\rangle^Y = \langle x\rangle_0^Y + \langle x\rangle_1^Y$. As Yao's protocol has constant round complexity, we use a size-optimized Ripple-Carry l-bit adder with l AND gates [34].

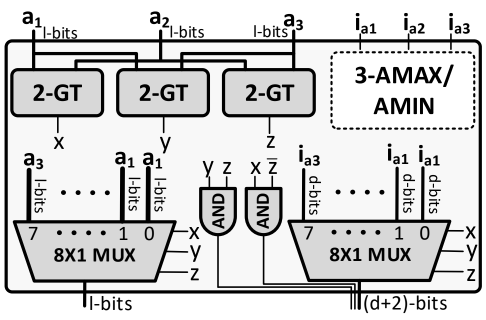
4.4 M2A

To convert from a MISC sharing $\langle x\rangle^R$ to Arithmetic sharing $\langle x\rangle^A$, one simple solution is to follow steps similar to the A2M conversion. Here, the parties evaluate a Boolean subtraction circuit with $\langle x\rangle^R$ and $\langle r\rangle^R$ as the inputs, where r is a random value chosen by P_0. In addition, P_0 generates $\langle r\rangle^A$. After the evaluation, the value $(x - r)$ is reconstructed to P_1, which further generates $\langle x - r\rangle^A$. The parties then locally compute $\langle x\rangle^A = \langle x + r\rangle^A$ - $\langle r\rangle^A$. In this solution, the subtraction circuit would either have size $O(l)$ and depth $O(l)$ [34] or size $O(l \cdot \log_2 l)$ and depth $O(\log_2 l)$ which results in a non-constant round protocol in the online phase [23]. Instead, we use a *novel* round efficient variant inspired by [19].

The general idea for converting a MISC share $\langle x\rangle^R$ to an Arithmetic share $\langle x\rangle^A$ is to use one OT for each bit. P_i locally sets $x_i = MSB(\langle x\rangle_i^R)$. P_0 chooses $r_i \in_R \{0,1\}^l$ and obtains $s_{i,0} = (1 - x_0[i]) \cdot 2^i - r_i$ and $s_{i,1} = x_0[i] \cdot 2^i - r_i$ as inputs of the i-th OT, whereas P_1 inputs choice bit $x_1[i]$ and receives $s_{x_1[i]} = (x_0[i] \oplus x_1[i]) \cdot 2^i - r_i$. After that, P_0 computes $\langle x\rangle_0^A =$



(a) using 2-AMAX circuit [34]



(b) using 3-AMAX circuit

Figure 7. AMAX/AMIN circuits [34]

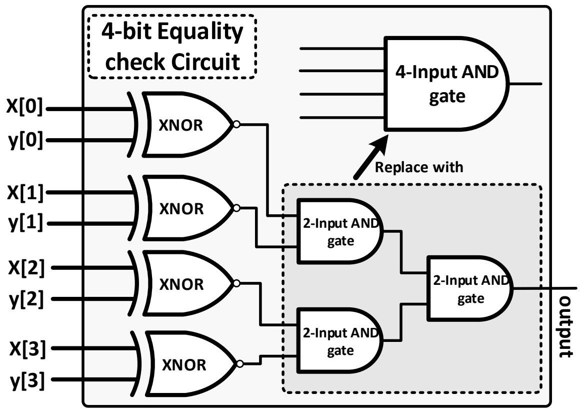


Figure 8. Equality circuit [23, 34]

$\sum_{i=1}^{l} r_i$ and P_1 computes $\langle x\rangle_1^A = \sum_{i=1}^{l}(x_0[i] \oplus x_1[i]) \cdot 2^i - \sum_{i=1}^{l} r_i = \sum_{i=1}^{l} x[i] \cdot 2^i - \sum_{i=1}^{l} r_i = x - \langle x\rangle_0^A$. Security and correctness are the same as in [19, 39].

5 Circuit Building Blocks

This section redesigns common building blocks used in several applications such as machine learning, and PSI. As our MISC protocol is very efficient for multi-input AND gates, we use circuits with such gates instead of the 2-input AND gates.

5.1 Maxpool

Maxpool is used in almost all Convolutional Neural Networks (CNN) as a pooling layer. Pooling layers are used to reduce complexity and extract low-level features from the neighborhood in neural networks. Maxpool selects the maximum number of M numbers. As described in [34] and shown in Figure 4a, the M-to-1 Maxpool circuit can be built from a binary tree of $(M-1)$ 2-to-1 (2-MAX) circuits that each consists of a Grater Than (2-GT) and a 2×1 multiplexer (MUX). We redesign the M-to-1 Maxpool circuit as shown in Figure 4b which uses $\lfloor M/2 \rfloor$ 3-to-1 max circuits (3-MAX). Each of the 3-MAX circuits consists of three 2-GT and one 8×1 multiplexer (MUX). We can use multi-input AND gates instead of 2-input AND gates in our protocol. Figure 5 shows our 8×1 multiplexer and Figure 6 shows our 8-bit comparator (8-GT) circuit with multi-input AND gates based on [36]. Usage of a 2-MAX or 3-MAX circuit is a trade-off between the round and communication overhead. The Maxpool that uses the 3-MAX components have lower depth, and the Maxpool that uses the 2-MAX components have lower communication.

Table 5 compares the round and communication complexity of MISC (this work), and GMW [16]. MISC has an $8 \log_2 M$ round complexity in the proposed Maxpool circuit due to the using multi-input AND gates, while in the regular Maxpool circuit, it has a $16 \log_2 M$ round complexity. For 2-MAX and 3-MAX, our MISC protocol has up to $1.41\times$ less communication than GMW [16].

5.2 Argmax

The Argmax circuit, similar to the Maxpool circuit in Section 5.1, finds the maximum of its input values. However, the output of this circuit also determines the index of the largest input. This circuit is similar to the Maxpool in Figure 4. The only difference is, in each 2-to-1 and 3-to-1 ArgAmax (2-AMAX, 3-AMAX), two multiplexers are used instead of 1: one for choosing the max value, and one multiplexer for choosing the index of the max input. Figure 7 shows such a 2-AMAX circuit as proposed in [34]. The Argmax circuit has the same depth as the Maxpool circuit and needs to compute one more multiplexer in each 2-AMAX or 3-AMAX. The resulting complexity is given in Table 5. The Argmin circuit has a similar structure to Argmax.

5.3 Equality Check

The equality circuit is a common building block used in circuit-based PSI [40–42]. As shown in Figure 8, this circuit can be built from XNOR and AND gates [23, 34]. For N-bit inputs, we need N XNOR and a tree

Table 5. Communication and round complexity of our MISC protocol in comparison with [16] for different building blocks (M is the number of inputs of bitlength 16 bit. Best values marked in bold)

Building Blocks	MISC (This protocol) Setup [bits]	Online [bits]	# Online Rounds	GMW [16] Setup [bits]	Online [bits]	# Online Rounds
Equality (Section 5.3)	**945**	95	**4**	2 010	60	4
Maxpool use 2-MAX (Section 5.1)	**5 510(M-1)**	392(M-1)	$8 \cdot \log_2 M$	7 772(M-1)	232(M-1)	$6 \cdot \log_2 M$
Maxpool use 3-MAX (Section 5.1)	**26 466(M/2)**	1 896(M/2)	$8 \cdot \log_3 M$	32 844(M/2)	238(M/2)	$8 \cdot \log_3 M$
Argmax/Argmin use 2-AMAX (Section 5.2)	**6 126(M-1)**	432(M-1)	$8 \cdot \log_2 M$	9 108(M-1)	66(M-1)	$6 \cdot \log_2 M$
Argmax/Argmin use 3-AMAX (Section 5.2)	**31 914(M/2)**	2 558(M/2)	$8 \cdot \log_3 M$	37 536(M/2)	272(M/2)	$8 \cdot \log_3 M$

Table 6. Total communication, rounds and time for CNN inference on MNIST dataset using MISC, GMW [16], and ABY2.0 [17] (Best results are marked in bold)

Machine Learning	Total [MB]	Online Rounds	Total LAN [s]
2-MAX in GMW [16]	209.13	**67**	3.82
2-MAX in ABY2.0 [17]	211.49	51	3.82
2-MAX in MISC	**204.24**	83	**3.80**
3-MAX in GMW [16]	218.64	51	3.88
3-MAX in ABY2.0 [17]	227.41	35	3.94
3-MAX in MISC	**213.46**	**51**	**3.84**

of (N-1) two-input AND gates of depth $\lceil \log_2 N \rceil$. We can replace three 2-input AND gates with a 4-input AND gate. As shown in Table 5, for a 16-bit equality circuit, this improves communication over GMW [16] by 2.13× with the same round complexity.

6 Applications and Benchmarking

We evaluate our MISC protocol on three privacy-preserving applications: machine learning, and circuit-based private set intersection. We perform experiments on servers with an Intel Xeon E5-2690 CPU with 64 cores and 64GB RAM. Computation time was measured by the [43] tool. We simulate a network with *1 Gbit/s* bandwidth and *1 ms* RTT for The LAN setting and *100 Mbit/s* bandwidth and *100 ms RTT* for the WAN setting. We ran our simulations ten times and got the average.

6.1 Machine Learning

We evaluate our MISC protocol on Convolutional Neural Networks (CNN) for handwriting recognition on the MNIST dataset used in previous works [5, 46]. This CNN has two Convolutional (Conv) layers using ReLU as an activation function and a Maxpool layer. Also, this CNN has two Fully Connected (FC) layers with the ReLU activation function. We use Arithmetic sharing for the Conv and FC layers, and then we use our conversion from Section 4 to convert to MISC sharing and then use MISC for the ReLU and Maxpool layers. We use a 2×1 Multiplexer from Section 5 for ReLU and 9-to-1 Maxpool from Section 5.1 for the pooling layers. In Table 6, we compare the performance of CNN inference using the 2-MAX or 3-MAX circuit evaluated by MISC, and GMW [16]. We used building blocks shown in Section 5 to evaluate the non-linear CNN layers. However, for GMW [16] we have used a 2-input AND gate. Compared to GMW [16], we improve total communication by 1.02× (non-linear communication by 1.4×) and total time by 1.01× in LAN. When using 3-MAX, we improve communication by 1.02× (non-linear communication by 1.2×), and total time in the LAN setting by 1.01×. Also, compared to ABY2.0 [17], MISC improves communication by 1.06× (non-linear communication by 1.7×), and total time in the LAN by 1.01×.

GALA [47] shows that evaluating the convolutional layers and fully connecting in CNN using arithmetic sharing takes about 98% of the computation time. Therefore, the improvement of MISC is reduced in Table 6. The recent Delphi framework [37] spends 52% of the computation time on the linear layers [47, Table 7]. MISC improves the non-linear layers over other GMW-style protocols [16], which can be used in new PPML frameworks to boost the efficiency of non-linear layers further.

6.2 Circuit-Based-PSI

Circuit-based PSI [14, 40–42, 44] allows two parties to privately compute a function on the intersection of their private input sets. This has several applications, like measuring ad conversion rates, data mining, and social networks. Today's PSI protocols [14, 40–42] first use hash functions to achieve sub-quadratic overhead and then securely evaluate equality circuits for checking the equality of many bit strings in parallel. Most of the computation and communication is spent on this private equality checks [17, 42]. To be precise, when computing the intersection between two sets, 96% of the overall communication (cf. [42, Table 3]) and 34% – 63% of the overall runtime (cf. [42, Table 5]) is spent on Equality Checking. Using our efficient

Table 7. Total communication, round and time for circuit-based PSI protocols on sets with N elements of length 32-bit (Best results are marked in bold)

PSI Protocols	$N = 2^{12}$			$N = 2^{16}$			$N = 2^{20}$		
	Total [MB]	Online Rounds	Total WAN [ms]	Total [MB]	Online Rounds	Total WAN [ms]	Total [MB]	Online Rounds	Total WAN [ms]
SCS [44]	104	60	-	2174	80	-	42976	100	-
Circuit-Phasing [14]	130	5	37380	1683	5	327976	21004	5	4850571
Hashing+SCS [45]	-	-	-	1537	42	-	21207	36	-
2D CH [41]	51	5	22796	612	5	129436	6582	5	1512505
Stash-Less [42]	9	6	5910	149	6	22134	2540	6	261481
Stash-Less [42] **with GMW** [16]	6	6	5727	112	6	19106	1924	6	209877
Stash-Less [42] **with ABY2.0** [17]	5	4	-	94	4	-	1608	4	-
Stash-Less [42] **with GC** [11]	9	**2**	-	149	**2**	-	2540	**2**	-
Stash-Less [42] **with MISC**	**2**	6	**4917**	**48**	4	**15186**	**829**	4	**168647**

Equality Check circuit (Section 5.3) in today's best efficient circuit-based PSI protocol using STPC [42] results in an improvement of 3 × in overall communication, and 1.5× in total time. Moreover, compared to GC [11], MISC improves communication by 3.69×.

Table 7 shows the resulting round, communication, and total time of the circuit-PSI of [42] using MISC and our efficient Equality Check compared to previous works [14, 41, 42]. In this table, the communication is shown on three sets of elements of length 32 bit. The numbers for previous works are taken from [41, 42]. For better comparison we evaluate [42] with GC [11], GMW [16], and ABY2.0 [17] protocols.

7 Conclusion and Future Work

Nowadays, privacy-preserving computing has gained vast attention due to the growth of using machine learning and PSI. Several protocols can be used for secure computation on private data, e.g., Yao's Garbled Circuits [31] and Secret Sharing (GMW [9]). We propose a new protocol MISC that reduces communication overhead with multi-input AND gate evaluation. For better efficiency, we designed conversions between MISC and other STPC protocols. Finally, we showed three examples for applications that benefit from the improvements of our MISC protocol. Our results show that MISC reduces communication complexity, and total time in these applications in the LAN setting. It would be interesting to extend MISC to stronger adversary models and more than two parties in future work.

References

[1] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, 1999.

[2] Oleksandr Tkachenko, Christian Weinert, Thomas Schneider, and Kay Hamacher. Large-scale privacy-preserving statistical computations for distributed genome-wide associati, organization=ACM on studies. In *ASIACCS*. ACM, 2018.

[3] Wilko Henecka, Stefan Kögl, Ahmad Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: Tool for automating secure two-party computations. In *CCS*. ACM, 2010.

[4] Farhad Taheri, Siavash Bayat-Sarmadi, and Shahriar Ebrahimi. Efficient hardware implementations of legendre symbol suitable for mpc applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2021.

[5] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE S&P*, 2017.

[6] Payman Mohassel and Peter Rindal. ABY^3: A mixed protocol framework for machine learning. In *CCS*. ACM, 2018.

[7] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. MP2ML: A mixed-protocol machine learning framework for private inference. In *ARES*. ACM, 2020.

[8] Andrew Chi Chih Yao. How to generate and exchange secrets. In *FOCS*, 1986.

[9] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC*, 1987.

[10] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*. Springer, 2008.

[11] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*. Springer, 2015.

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

[13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE S&P*, 2013.

[14] Benny Pinkas, Thomas Schneider, Gil Segev, and

Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security*, 2015.

[15] Satsuya Ohata and Koji Nuida. Towards high-throughput secure MPC over the Internet: Communication-efficient two-party protocols and its application. In *FC*, 2020.

[16] Ghada Dessouky, Farinaz Koushanfar, Ahmad Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *NDSS*, 2017.

[17] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved mixed-protocol secure two-party computation. In *USENIX Security*, 2021.

[18] D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

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

[20] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Asia conference on computer and communications security*, pages 707–721, 2018.

[21] Hossein Yalame, Hossein Farzam, and Siavash Bayat-Sarmadi. Secure two-party computation using an efficient garbled circuit by reducing data transfer. In *Applications and Techniques in Information Security*. Springer, 2017.

[22] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge University Press, 2009.

[23] Thomas Schneider and Michael Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *FC*, 2013.

[24] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *TC*, 1990.

[25] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryptio. In *CRYPTO*, 2012.

[26] Moni Naor and Benny Pinkas. Computationally secure oblivious transfer. *Journal of Cryptology*, 2005.

[27] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *STOC*, 1989.

[28] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS*. ACM, 2013.

[29] D. Beaver. Precomputing oblivious transfer. In *CRYPTO*, 1995.

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

[31] A. C. Yao. Protocols for secure computations. In *FOCS*, 1982.

[32] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *CCS*. ACM, 2019.

[33] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *CCS*. ACM, 2020.

[34] Vladimir Kolesnikov, Ahmad Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS*, 2009.

[35] Niklas Buescher, Andreas Holzer, Alina Weber, and Stefan Katzenbeisser. Compiling low depth circuits for practical secure computation. In *ESORICS*, 2016.

[36] Juan Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *International Workshop on Public Key Cryptography*, pages 330–342. Springer, 2007.

[37] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *USENIX Security*, 2020.

[38] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *IEEE S&P*, 2020.

[39] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *USENIX Security*, 2019.

[40] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM TOPS*, 2018.

[41] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In *EUROCRYPT*. Springer, 2018.

[42] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT*. Springer, 2019.

[43] OTextention. https://github.com/encryptogroup/ \uppercase {OT}extention. Accessed: October. 2020.

[44] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits bet-

ter than custom protocols? In *NDSS*, 2012.

[45] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. In *WPES*, 2019.

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

[47] Qiao Zhang, Chunsheng Xin, and Hongyi Wu. Gala: Greedy computation for linear algebra in privacy-preserved neural networks. *NDSS*, 2021.

Farhad Taheri Received a B.Sc. degree in computer engineering from Shahid Bahonar University of Kerman (SBUK), Kerman, Iran, in 2016, and an M.Sc. degree in computer engineering from Sharif University of Technology (SUT), in 2018. From 2016 to 2018, he was a member of the Data Storage, Networks, and Processing (DSN) Lab at SUT where he researched on the reliability of Solid-State Drives (SSDs). Currently, he is a Ph.D. candidate at the Smart and Secure Systems (3S) lab at SUT under the supervision of Dr. S. Bayat-Sarmadi. His research interests include Privacy-Preserving Machine learning, Multi-Party Computation, Machine learning acceleration, and System Security.

Siavash Bayat-Sarmadi received the B.Sc. degree from the University of Tehran, Iran, in 2000, the M.Sc. degree from the Sharif University of Technology, Tehran, Iran, in 2002, and the Ph.D. degree from the University of Waterloo in 2007, all in computer engineering (hardware). He was with Advanced Micro Devices, Inc. for about six years. Since September 2013, he has been a faculty member in the Department of Computer Engineering, at Sharif University of Technology. He has served on the executive committees of several conferences. His research interests include hardware security and trust, cryptographic computations, and secure, efficient and dependable computing, and architectures. He is a member of the IEEE.